Chinese to English Machine Translation Using SNePS as an Interlingua

by

Min-Hung Liao

Dec, 1997

A thesis submitted to the
Faculty of the Graduate School of State
University of New York at Buffalo
in partial fulfillment of the requirements for the degree of
Master of Arts

Thesis Committee:
Dr. Matthew S Dryer, Chairman
Dr. Jean-Pierre A Koenig
Dr. William J. Rapaport

Contents

List of Figures

Abstract

An interlingua machine translation is a two-stage operation: from source language to an interlingua, and from the interlingua to the target language. The idea of translating natural-language texts using an interlingua, an intermediate common language, is based on the belief that while languages differ greatly in surface structures, they share a common deep structure.

I propose a way of automatically translating texts using SNePS as an interlingua. The representation of the meaning of the source-language input is intended to be language-independent, and this same representation is used to synthesize the target-language output. As an interlingua, SNePS fulfills the requirements of being formal, language-independent, and a powerful medium for representing meaning; thus, it can handle ambiguities.

SNePS can be used to translate texts automatically as follows. The user inputs a sentence of the source language to a generalized augmented-transition-network (GATN) parser-generator. The parser fragment of the GATN parser-generator updates an existing knowledge base containing semantic networks to represent the system's understanding of the input sentence. The node newly built to represent the proposition is then passed to the generator fragment of the GATN parser-generator, which generates a sentence of the target language expressing the proposition in the context of the knowledge base.

The parsing of Chinese relies more on semantic information than syntactic relations because, first, the word order is determined primarily by semantic factors rather than syntactic ones, second, there is a lack of morphological inflections and syntactic clues. A series of noun phrases and verb phrases can be juxtaposed without syntactic glues such as function words or variation of verb forms to make the linking. These linguistic properties cause lexical and structural ambiguities. Besides being an adequate interlingua representation, SNePS is also a computational environment particularly suitable for processing Chinese because it provides the facilities for building, retrieving, and deducing semantic information that guides the parsing and resolves ambiguities.

Chapter 1

Introduction

This project is a study on the design of an interlingua machine translation system. The system takes single Chinese sentences as input and produces SNePS semantic networks to represent their meanings. The semantic-network representations resulting from analyzing the input sentences are called the interlingua. Taking the interlingua as input, the generator produces English sentences that are the translations of the input Chinese sentences.

Some characteristics of Chinese, such as the lack of morphological inflection and linking devices to identify grammatical relations between words, cause ambiguities and difficulties in analyzing Chinese sentences. An inference mechanism operating on the knowledge base and a set of inference rules is needed to resolve the ambiguities by checking whether the noun phrase in question meets the constraints of semantic role under the subcategorization frame of the verb. The ambiguities we are trying to resolve include those arising from relative clauses, a sequence of nouns, serial verb constructions, and two different functions of 的 *de* for genitive marker and relative clause marker. The subcategorization information for verbs is encoded in the lexicon. The knowledge base is a SNePS semantic network that represents a variety of concepts such as objects, their properties, and propositions.

The result of the source language analysis is a SNePS semantic network that represents the proposition of the Chinese input. The representations for propositions of various grammatical constructions are proposed e.g., indicative clause, relative clause, purpose clause, XCOMPlement and interrogatives etc. Because both the knowledge base and the interlingua are SNePS semantic networks, the interlingua not only serves as the basis for the generation of the English translation but also becomes part of the knowledge base. That is to say, every input sentence translated also updates the current state of the knowledge base. For ambiguity resolution, the knowledge base built in the system initialization stage provides the background knowledge and the interlingua representations built in the course of translation provide the contextual knowledge. Because different contextual knowledge may give different disambiguating results, the previous translations can affect the current one or, in other words, the results of translating identical inputs may not always be the same depending on their previous inputs.

In generation, the SNePS semantic network representing the proposition of the input Chinese sentence is mapped onto the English syntactic structure. For example, an object's properties are realized as a sequence of adjectives according to the English adjective ordering. Complements to the verbs are realized in different forms according to the patterns of the verbs. For example, the complements can be *that*-clauses, infinitives, gerundized, or in genitive form etc. Inference is used in lexical selection. For example, if the generator infers that the head noun of the relative clause is animate then *who* is selected for the relative pronoun; *which* otherwise.

The paper is organized as follows. The second chapter provides typological, syntactic, and morphological descriptions of Chinese. The difficulties in analyzing Chinese will be addressed there. Chapter 3 gives an overview of the SNePS architecture on which the translation system is built. The syntax and semantic of the semantic-network representations used for the interlingua and knowledge base in this project are explained. Chapter 4 describes the parsing grammar and analysis process. The generation grammar and process are illustrated in chapter 6. In chapter 7, the project is summarized, directions of extensions and improvements are suggested. An annotated sample run that illustrates the translation processes is given in the appendix.

Chapter 2

Characteristics of Chinese

In this Chapter, we will describe some special features of Chinese that are different from those of English. Then, we will discuss what difficulties these features cause in the translation processes.

2.1 Typological description

2.1.1 Morphological structure

Chinese characters most nearly correspond to morphemes. Many of the Chinese characters also happen to be words. Chinese has been classified as an *isolating* language whose words are typically composed of a single morpheme without inflection. Compared to *inflecting* languages, Chinese has few affixes and very little morphological complexity. For example, Chinese does not mark nouns for case[1], number, and definiteness. Chinese verbs do not have agreement morphemes to highlight certain properties of subjects or objects such as gender, number etc. Chinese verbs are not inflected for tense either, though they have

[1]In this thesis, the term *case* all refer to the semantic case.

aspect suffixes.

2.1.2 Word structure

In written Chinese, there are no delimiters such as spaces to mark the word boundary. A word can consist of one or more morphemes. Two or more words can form a compound word e.g. a nominal compound. All the morphemes in a sentence run together without a break. For clarity of explanation, the Chinese sample sentences in this thesis have been morphologically broken down and spaces are inserted among words. For example, sentence (2.1) in section 2.2, instead of being displayed like this:

張三	教	英文	日文.
Zhang1 san1	jiao1	Ying1 wen2	Ri4 wen2.
Zhang1san1	teach	English	Japanese

Zhang1san1 teaches English and Japanese.

The usual way to write it is:
張三教英文日文.
Zhang1 san1 jiao1 Ying1 wen2 Ri4 wen2.
It is more or less like reading the "English translation": *Zhang1san1teachesEnglishandJapanese.*

2.1.3 Word order

It is primarily semantic factors rather than syntactic ones which determine word order in Chinese. The "basic" word order is difficult to establish in Chinese; however, a sample text count yields more SVO[2] than SOV sentences.[Li and Thompson, 1981]

Chinese can be termed a *topic-prominent* language because the "topic" in Chinese grammar plays an important role.[Li and Thompson, 1981] The topic, occurring in sentence-initial position, is what the sentence is about. It always refers to something definite or that the hearer already knows about.

[2] S stands for "subject," O for "object," and V for "verb."

2.1.4 Modifier precedes the modified

Chinese exhibits more features of an SOV language than those of an SVO language, although the sample text contains more SVO sentences than SOV ones.[Li and Thompson, 1981] According to Greenberg[Greenberg, 1963], there are some word order parameters that should correlate with the verb and object order. The following properties of Chinese are characteristics of SOV languages:

- Relative clauses and Genitive phrases precede the head noun.
- Certain aspect markers follow the verb.
- Adpositional phrases and certain adverbials precede the verb.
- Question particles are at the end of a sentence.
- “Wh”-phrase occupies the same position in the sentence as the phrase it replaces.
- Articles are absent.
- Suffixes rather than prefixes are used.

The following properties are characteristics of SVO languages:

- Prepositions exist
- The negative markers, some aspect markers, modals, and “want” precede the verb.
- The copula precedes the predicate.
- There is no case on subject/object.

2.2 Linking

In Chinese, phrases and clauses can hold together simply side by side without conjunctions, function words, variation of verb forms, or the like to identify the connections between words as in English. For example, conjoined noun phrases follow each other without conjunctions.

張三	教	英文	日文.
Zhang1 san1	jiao1	Ying1 wen2	Ri4 wen2.
Zhang1san1	teach	English	Japanese

Zhang1san1 teaches English and Japanese. (2.1)

Another example is the serial-verb construction in which two or more verb phrases or clauses are juxtaposed without any marker indicating what the relationship is between them. [Li and Thompson, 1981]

張三	吃	晚飯	讀	書.
Zhang1 san1	chi1	wan3 fan4	du2	shu1
Zhang1san1	eat	dinner	read	book

Zhang1san1 ate dinner and read books. (2.2)

張三	相信	李四	喜歡	老王.
Zhang1 san1	xiang1 xin4	Li3 si4	xi3 huan1	Lao3 wang2.
Zhang1san1	believe	Li3si4	like	Lao3wang2

Zhang1san1 believed that Li3si4 liked Lao3wang2. (2.3)

張三	買	書	給	李四	讀.
Zhang1 san1	mai3	shu1	gei3	Li3 si4	du2
Zhang1san1	buy	book	give	Li3si4	read

Zhang1san1 bought books to give Li3si4 to read. (2.4)

張三	勸	李四	讀	書.
Zhang1 san1	quan4	Li3 si4	du2	shu1.
Zhang1san1	persuade	Li3si4	read	book

Zhang1san1 persuaded Li3si4 to read books. (2.5)

Comparing these Chinese sample sentences with their English translation, we can see that while there are no grammatical linkers in the Chinese serial verb construction, they can be found in the English translation. For example, in (2.2), the coordinating conjunction *and* conjoins two verb phrases. The subordinating conjunction *that* in (2.3) introduces the second clause that serves as the object of the first verb. The *to* infinitive in (2.4) and (2.5) links two verb phrases together.

2.3 Difficulties in processing Chinese

2.3.1 Word segmentation

Because Chinese characters follow one another without delimiters to mark word boundaries, grouping characters into words and deciding whether a certain character should belong to a word or a larger compound word add complexity to parsing Chinese. Consider how it will look like if one were asked to read the following "English sentence": *Thisisasentencewithoutthespacesplacedbetweenwords.*

2.3.2 The importance of semantics in parsing Chinese

Semantics plays important role in parsing Chinese. Syntax does not provide as much information in parsing Chinese as in parsing English. In fact, the grammatical characteristics of Chinese make a pure syntactic parser almost impossible. The characteristics discussed in the previous section will be examined to see what ambiguities and difficulties they cause and why semantics is important in parsing Chinese.

Ambiguities from a series of nouns

Two or more consecutive nouns in a sentence can cause structural ambiguity. Those nouns may form a nominal compound or a conjoined noun phrase without conjunction. Part of them may be conjoined into a noun phrase or combined into a nominal compound while the others remain as individual nouns. To parse the sentence successfully, the parser has to segment those nouns into noun phrases correctly. The following sample "sentences" show why syntax alone does not provide enough information to divide sequences of nouns into noun phrases properly. All five sentences share the same form, i.e. one preverbal noun and four consecutive postverbal nouns. To facilitate the comparison, the Chinese characters are omitted, only their English glosses are juxtaposed in the table below. These pseudo English sentences resemble their Chinese counterparts in most ways, for example, in the absent of morphological inflections, conjunction, or genitive marker etc. These pseudo sentences help demonstrate what ambiguities would occur if syntactic markers were left out.

Noun	Verb	Noun	Noun	Noun	Noun
John	teach	Math	English	Physics	Linguistics.
John	teach	Mary	Mark	Paul	Linguistics.
John	teach	Mary	English	Physics	Linguistics.
John	teach	Mary	Mark	Math	Linguistics.
John	teach	Mary	brother	Math	Linguistics.

Actual English sentences corresponding to each are listed below:

- John taught Math, English, Physics, and Linguistics.
- John taught Mary, Mark, and Paul Linguistics.

- John taught Mary English, Physics, and Linguistics.
- John taught Mary and Mark Math and Linguistics.
- John taught Mary's brother Math and Linguistics.

As we can see, punctuation and conjunction markers group the nouns into phrases in English. With the help of syntactic markers, the computer parser is able to divide the nouns into grammatical units e.g. indirect object and direct object. However, even in the absence of syntactic markers, we human "parser" can still identify which nouns should belong to the direct object and which nouns should be the indirect object, because in addition to syntactic information, semantic information is also available to disambiguate sentences. Due to its lack of syntactic marking, the parsing of Chinese relies more on semantic information e.g., the properties and taxonomic features of nouns, what semantic roles a verb subcategorizes for and the semantic constraints on these roles.

A sequence of preverbal nouns could be a conjunction of nouns or a nominal compound acting as the subject, a sentence-initial topical direct object followed by the subject, or the subject and object in a SOV sentence. Their surface structures are the same i.e. consecutive nouns followed by a verb phrase. To determine their grammatical functions, the parser again has to rely on semantic information.

張三	王五	教	過	英文	了.
Zhang1 San1	Wang2 Wu3	jiao1	guo4	Ying1 wen2	le5
Zhang1san1	Wang2wu3	teach	EXP	English	LE

Zhang1san1 and Wang2wu3 taught English. [3] (2.6)

英文	張三	教	過	王五	了.
Ying1 wen2	Zhang1 San1	jiao1	guo4	Wang2 Wu3	le5
English	Zhang1san1	teach	EXP	Wang2wu3	LE

[3] EXP denotes experiential aspect.

Zhang1san1 taught Wang2wu3 English. (2.7)

Sentences 2.6 and 2.7 have the same sentence form, namely: `noun noun verb EXP noun LE`. In sentence 2.6, the two preverbal nouns form a conjoined noun phrase, *Zhang1san1 and Wang2wu3*. In sentence 2.7, the first noun, 英文 *English*, is the direct object acting as the topic; and, the second noun 張三 *Zhang1san1* is the subject.

In sentence 2.8, the first noun, 書 *book*, is the topic. It is also the direct object. The second noun, 張三 *Zhang1san1*, is the subject.

書	張三	已經	買	了.
shu1	Zhang1 San1	yi3 jing1	mai3	le5
book	Zhang1san1	already	buy	LE

Zhang1san1 has already bought the books. (2.8)

In (2.9), the first noun is the subject and the second noun is the direct object. This SOV structure is used in a contrary-to-expectation situation. For instance, the teacher tells the students not to buy some books since they are available from the library. If Zhang1san1 has already bought the books, the other students will explain the situation to the teacher using the SOV sentence structure.

張三	書	已經	買	了.
Zhang1 San1	shu1	yi3 jing1	mai3	le5
Zhang1san1	book	already	buy	LE

Zhang1san1 has already bought the books. (2.9)

Sentence 2.9 has exactly the same surface pattern as that of (2.8). Only semantic information can tell the difference between the two.

Ambiguities from a series of verb phrases

Sentences 2.2,2.3,2.4, and 2.5 all have the same form, that is: `noun verb noun verb noun`; but, the relations that hold between the two verbs are different. The two verbs phrases may describe two separate events. For example, in (2.2), two actions, eating dinner and reading books, which follow each other. In (2.4), the first event, buying books, is done for the purpose of achieving the second event, giving Li3si4 the books to read. One verb phrase may be the subject or object of another. For example, in (2.3), the second clause, *Li3si4 liked Lao3wang2*, is the object of the first verb, *believe*. Sometimes the first verb controls where the second verb phrase get its missing subject. For example, in (2.5), the first verb "object-controls" the second verb phrase. That is, the second verb phrase, *read books*, gets its unexpressed subject from the object of the first verb.

Although there are no grammatical markers such as those found in English to distinguish the relations between verbs, the semantic and subcategorization information of the verbs help clarify their relationship. For example, the verb 相信 *believe* takes a clause as object and the verb 勸 *persuade* is an object control verb.

Another problem in parsing serial-verb construction is how to resolve the referential relations between the expressed subject/object of the first verb and the unexpressed subject/object of the succeeding verbs. More specifically, what are the subject and the object for the verb 讀 *read* in (2.2), (2.4), and (2.5)? For example, in (2.4), the missing object of the third verb 讀 *read* refers to the the second verb 給 *give*'s missing direct object, which in turn refers to 書 *books*, the object of the first verb *buy*. The missing subject of the third verb 讀 *read* refers to *Li3si4*, the indirect object of the second verb 給 *give*.

Li3 si4	*mai3*	*shu1*	*gei3*	*Zhang1 san1*	*du2.*	
Li3si4	buy	book	give	Zhang1san1	read	

Li3si4 bought a book to give Zhang1san1 to read. (2.10)

The noun *book* is linked semantically to the verb *read* two layers down in the serial-verb sentence. The coreferential relations between the arguments of the first verb with the missing arguments of a verb embedded one or more layers down in a sentence is a kind of *long-distance dependency* i.e., two grammatical elements are non-locally dependent on each other. The phenomenon of long-distance dependencies exists in English too. For example, resolving referential relation for a missing grammatical argument is also a problem in English purpose clause and control sentence. Assigning the correct grammatical function to the head noun of a relative clause is another kind of long-distance dependency problem in both Chinese and English. The resolution of this problem again can use semantic role information.

張三	給	李四	王五	喜歡	李四	的	狗.
Zhang1 San1	gei3	Li3 si4	Wang2 wu5	xi3 huan1	Li3 si4	de5	gou3
Zhang1san1	give	Li3si4	Wang2wu3	like	Li3si4	DE	dog

Zhang1san1 gave Li3si4 and Wang2wu3 the dog that liked Li3si4. (2.11)

張三	給	李四	王五	借	李四	的	狗.
Zhang1 San1	gei3	Li3 si4	Wang2 wu5	jie4	Li3 si4	de5	gou3
Zhang1san1	give	Li3si4	Wang2wu3	lend	Li3si4	DE	dog

Zhang1san1 gave Li3si4 the dog that Wang2wu3 lent Li3si4. (2.12)

Sentences (2.11) and (2.12) are almost identical except for the two verbs, 喜歡 *like* and 借 *lend*. The verb *like* subcategorizes for a subject and an object; the verb *lend* for a subject, a indirect object and an direct object. In the relative clauses, the subject is missing for the verb *like*; the direct object is missing for the verb *lend*. Therefore, the head noun 狗 *dog* refers to different missing grammatical functions in the relative clauses; in (2.12), the head noun refers to the missing direct object; in (2.11), the head noun refers to the missing subject.

Besides resolving the referential relation between the head noun and the missing function in the relative clause, the verb's semantic-role subcategorization information helps establish the boundary of the relative clause. In English, we have to decide where the relative clause ends while, in Chinese, we have to decide where the clause starts. Although the relative clause marker 的 *de* marks the end of the relative clause, there is nothing to mark the beginning of the relative clause. From the surface structure, the parser is not able to tell whether it is parsing an element in the main clause or one in the relative clause. However, with the subcategorization information of the verb, the parser can judge which elements should belong to the relative clause and which elements belong to the main clause. For example, because the verb 借 *lend* in the relative clause of (2.12) needs the propername 王五 *Wang2wu3* to fill its agent case-role but the verb 喜歡 *like* does not need *Wang2wu3* to fill its case frame, the propername *Wang2wu3* belongs to the main clause in (2.11) but it belongs to the relative clause in (2.12). That is to say, the relative clause starts from the verb 喜歡 *like* in (2.11); but, the propername 王五 *Wang2wu3* in (2.12).

(2.11) and (2.12) are also examples of Chinese serial nouns. Native speakers of Chinese would put a 和 *and* between two propernames *Li3si4* and *Wang2wu3* for (2.11). However, since the conjunction is optional in Chinese, the parser still has to use semantic information to rule out the possibility that these two propernames are conjoined in (2.12).

2.4 A computational environment suitable for the processing of Chinese

From the discussion above, we know that semantic information is essential to the computational understanding of Chinese. The parsing of Chinese should be semantically driven instead of being syntax-oriented. Therefore, the computational environment suitable for parsing Chinese is one that provides semantic information and facilities for utilizing semantic information efficiently e.g. performing inference with it. In the next section, we will discuss the features of SNePS, the Semantic Network Processing System, and show SNePS provides a platform that is particularly suitable for parsing Chinese.

Chapter 3

An Overview of the SNePS/CASSIE Architecture

CASSIE, the Cognitive Agent of the SNePS System—an Intelligent Entity, is a particular computational model of a cognitive agent. CASSIE uses SNePS (Semantic Network Processing System) [Shapiro, 1979, Shapiro and Rapaport, 1987, Shapiro and Rapaport, 1992] as the knowledge-representation system, SNIP (the SNePS Inference Package) as the reasoning system, and SNaLPS (the SNePS Natural Language Processing System) to interact with other cognitive agents such as native speakers. There have been a number of CASSIE projects done by the SNePS Research Group, and all the inputs to her are in English. (For a description of the CASSIE projects, please refer to [Shapiro, 1989]) In this project, CASSIE is able to understand one more language, Chinese, thus becoming multilingual. Since CASSIE can understand Chinese and then expresses it in English, she is a Chinese-English translator too. Our version of CASSIE is being implemented in the SNePS-2.3 knowledge-representation and reasoning system, written in Common Lisp and running on Sun SPARCs under the UNIX operating system environment. In this chapter, I describe the systems that make up CASSIE.

3.1 SNePS

A semantic network is a data structure, typically consisting of labeled nodes and labeled, directed arcs, that have associated with them facilities for representing information, for retrieving information from it, and for performing inference with it. A SNePS semantic network can be considered as a labeled, directed graph [Rapaport, 1991] wherein each node represents a concept, and arcs between nodes represent non-conceptual or structural relations. [Shapiro and Rapaport, 1991] The meaning of a node is determined by the structure of the entire network connected to it.

SNePS is an intensional, propositional semantic-network formalism. It is intensional, because the objects of CASSIE's thought may be fictional object (e.g., Sherlock Holmes), non-existent objects (e.g., a golden mountain), or impossible objects (e.g., a round square), and objects not substitutable in intensional contexts (e.g., the morning star and the evening star). The fact that SNePS is an intensional semantic network is embodied in the Uniqueness Principle: there is a one-to-one correspondence between nodes and represented concepts. This principle guarantees that nodes represent intensional objects.

It is propositional in that unlike structured-inheritance networks such as KL-ONE, which represent taxonomic inheritance hierarchies by arcs, SNePS represents all information, including propositions, by nodes. For example, a structured-inheritance semantic network might represent the proposition that Socrates is a man and man is human as in Figure 3.1. One important feature of such networks is that the

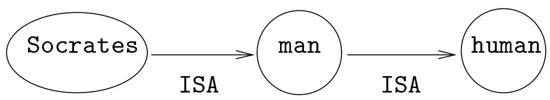

Figure 3.1: An ISA network representation.

properties are allowed to be "inherited," so that the fact that Socrates is human does not have to be stored at the `Socrates` node.

The information represented in the structured-inheritance semantic network of Figure 3.1 could be represented in the propositional semantic network as in Figure 3.2. Now there is a node arbitrarily labeled

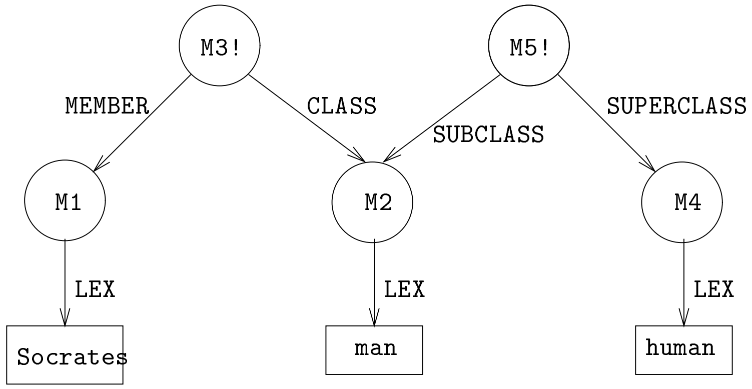

Figure 3.2: A SNePS propositional semantic network.

`M3`, which can represent the proposition that Socrates is a man; and, another node arbitrarily labeled `M5`, which represents the proposition that man is human. The advantage of propositional semantic networks as a knowledge-representation system over structured-inheritance networks is that propositions about propositions can be represented without limit.

3.1.1 Types of nodes

In SNePS, two types of nodes, atomic and molecular, are distinguished. Every node has an identifier, which uniquely identifies it. Atomic nodes, which have only arcs pointing to them, include:

- sensory nodes, which represent CASSIE's interfaces with the external world (e.g., utterances)

- base nodes, which represent some particular entity whose properties are assertionally determined by the arcs coming into it. And

- variable nodes, which represent arbitrary individuals, and propositions.

Molecular and pattern nodes, which have arcs emanating from them, are structurally determined by the arcs pointing out of them and the nodes they go to. Molecular nodes represent structured individuals and propositions, including atomic propositions and rules. Pattern nodes are molecular nodes which dominate variable nodes. Molecular nodes with an exclamation mark at the end of its identifier represents CASSIE's believes and are called asserted nodes.

3.1.2 SNePS as a knowledge representation system

SNePS provides a user interface, the SNePS User Language (SNePSUL), to build, access and retrieve information from the network. Using SNePSUL, we can define a set of relations: individual arc labels and sets of arc labels (or case frames) that will be used to represent various objects and information about them. Below, I give examples of the network representation of a few simple propositions.

Representation of subclass/superclass

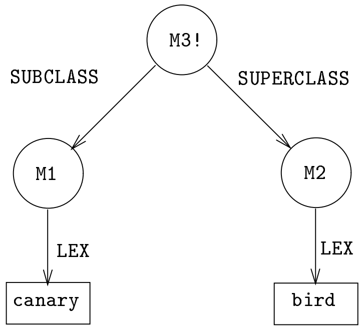

Figure 3.3: Superclass/Subclass

The subclass/superclass case frame represents the subset relation between two classes. In Figure 3.3, `M3` represents the proposition that the class of `canary` is a subclass of the class of `bird.`

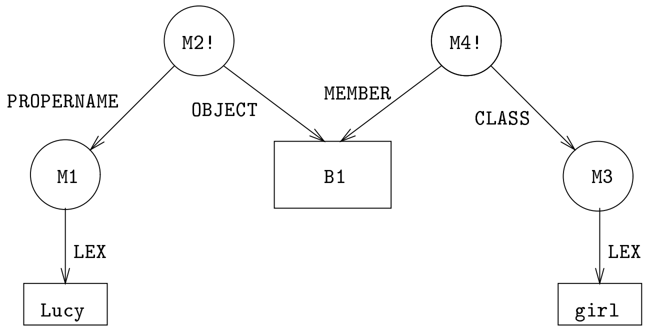

Figure 3.4: Member/Class

Representation of class/membership

Figure 3.4 illustrates the SNePS representation `M4` of the individual Lucy denoted by node `B1` being a member of the class `girl`. Two kinds of *ISA* relations, set membership (member/class) and class inclusion (superclass/subclass), are differentiated. For example, The proposition, *A canary is a bird,* is represented with the `superclass/subclass` arcs as shown in Figure 3.3 while the proposition, *Lucy is a girl,* is represented with the `member/class` arcs as shown in Figure 3.4 since a proper name designates a specific member of a class.

Representation of object/propername

Figure 3.4 illustrates the SNePS representation `M2` of the individual `B1` being named `Lucy`.

Representation of object/property

In Figure 3.5, `M4` represents the proposition that Lucy has the property of being rich.

Representation of ability

In Figure 3.6, `M3` represents the proposition that the `bird` has the ability to `fly`.

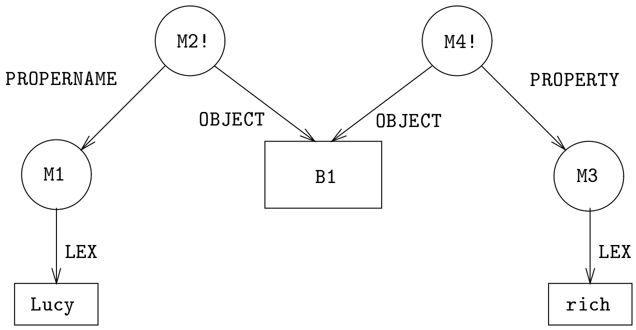

Figure 3.5: Object/Property

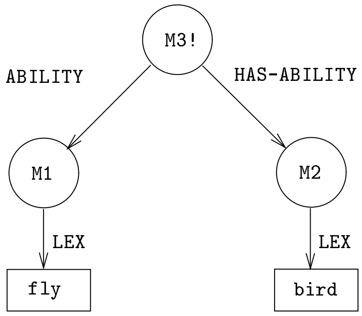

Figure 3.6: Has-ability/ability

Representation of possessive relations

Figure 3.7 illustrates the network representation of *John's book*: `M7` represents that `B1` is a book of `B2`, who is named `John`.

Representation of kinship relations

The representation illustrated in Figure 3.8 is used in our SNePS network to express that Tom is a son of John: `M11` represents that `B1` (who is named `Tom`) is a son of `B2` (who is named `John`).

Representation of part/whole

Figure 3.9 illustrates the network representation of a `trunk` being a part of an `elephant`.

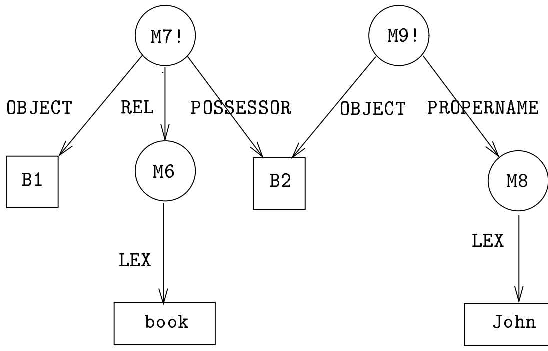

Figure 3.7: Possessor/Rel/Object

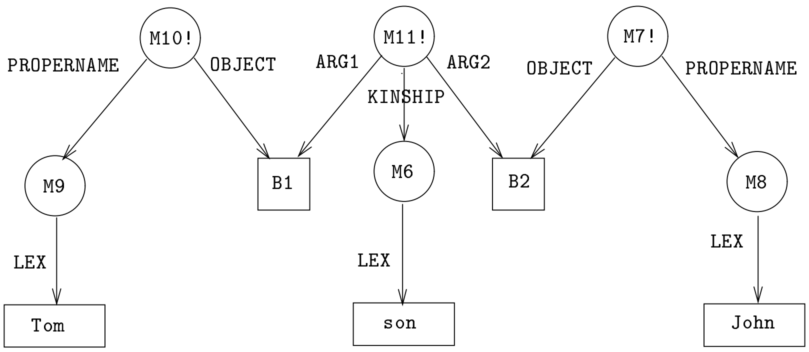

Figure 3.8: kinship relations

Representation of propositional relations

One advantage of SNePS as a propositional semantic network is that propositions about propositions can be represented without limit. Nodes are connected together to form propositions, which in turn can be

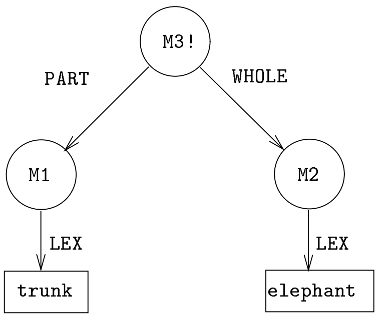

Figure 3.9: Part/Whole

connected together to form propositions of even higher levels. In this section, we show how propositional relations are represented.

- Representation of information in a sentence

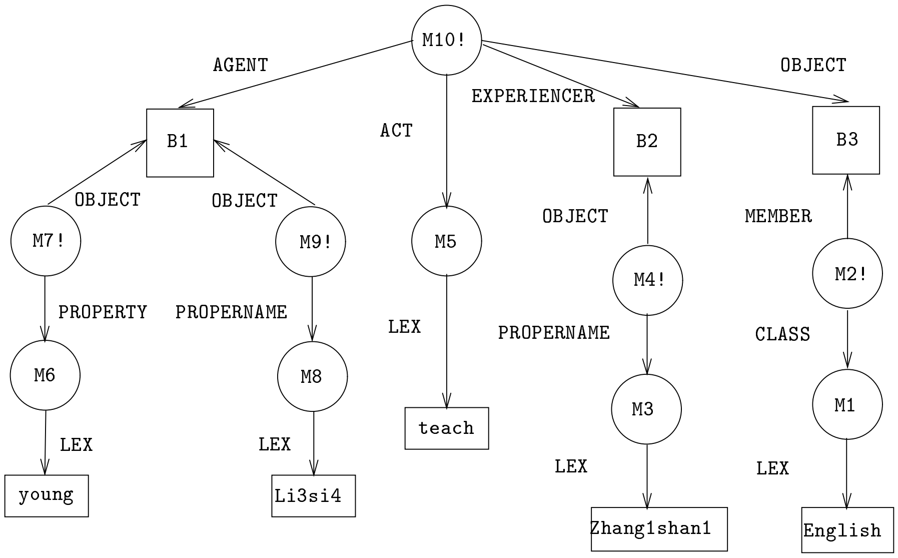

Figure 3.10: *Young Li3si4 taught Zhang1san1 English.*

Figure 3.10 illustrates the SNePS network representation of the information that can be linguistically expressed as *Young Li3si4 taught Zhang1san1 English.*[1] Five basic cases are used to represent the valence structure of a sentence. Detailed descriptions of these cases are deferred to the next chapter.

- Representation of information in an indicative clause

Figure 3.11 illustrates an example of an indicative clause. Here, `M10` represents the proposition that John `B3` believed another proposition `M6`, related to `M10` by a `COMP` arc.

[1]In this and the following examples, the default tense is the past tense. Since there is no tense in Chinese sentences, their English translation will default to the past tense.

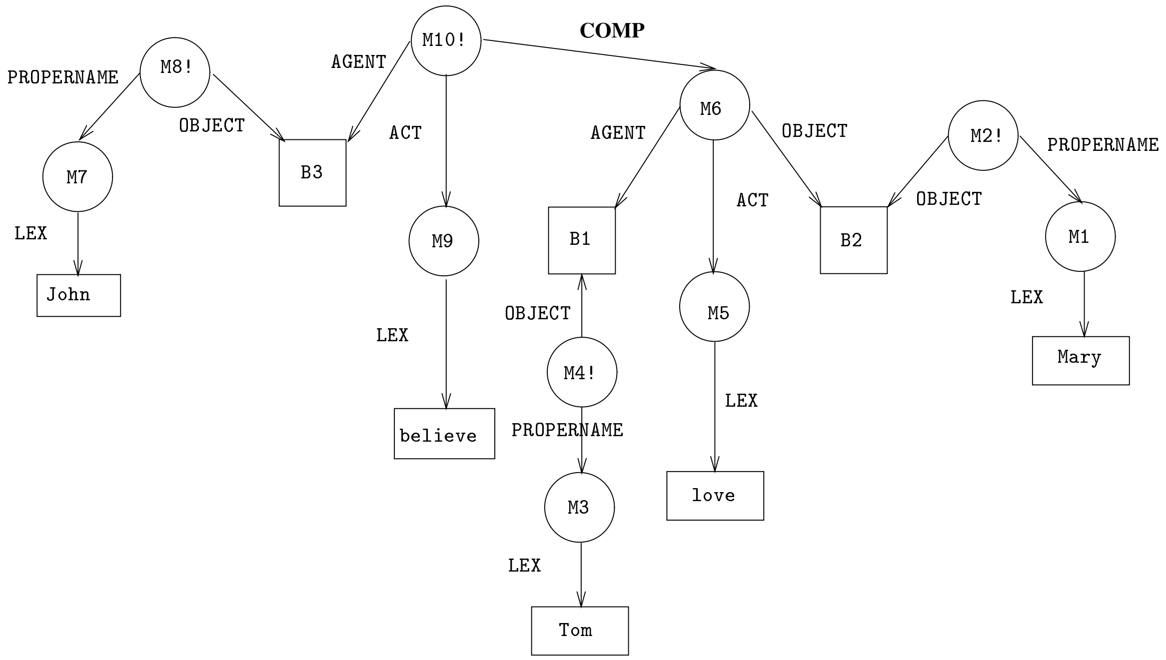

Figure 3.11: *John believed that Tom loved Mary.*

- Representation of main/relative clauses

The relation between a main clause and a relative clause can be represented in SNePS by *main/relc* arcs.[2] For instance, the sentence *Mary liked the flower that John bought,* consisting of a main clause *Mary liked the flower* and a relative clause *John bought the flower,* can be represented (as in Figure 3.12) by a molecular node `M301` that dominates both the propositional node `M300` for the main clause, and the propositional node `M284` for the relative clause via `main` and `relc-o` arcs, respectively. The `relc-o` arc specifies that the head noun of the relative clause plays the `Object` role of the main clause. We can use the `relc-a` arc for the `Agent` role, the `relc-e` arc for the `Experiencer` role, the `relc-l` arc for the `Locative` role, and the `relc-b` arc for the

[2]This representation is far from being adequate as it mixes the syntactic information with the semantic information. Further studies are needed to come up with a better way of representing relative clauses. One possible improvement could be to propose the *new/presupposed* case frame in place of the current *main/relc* one, since the relative clause usually represent a presupposed information whereas the main clause gives new proposition.

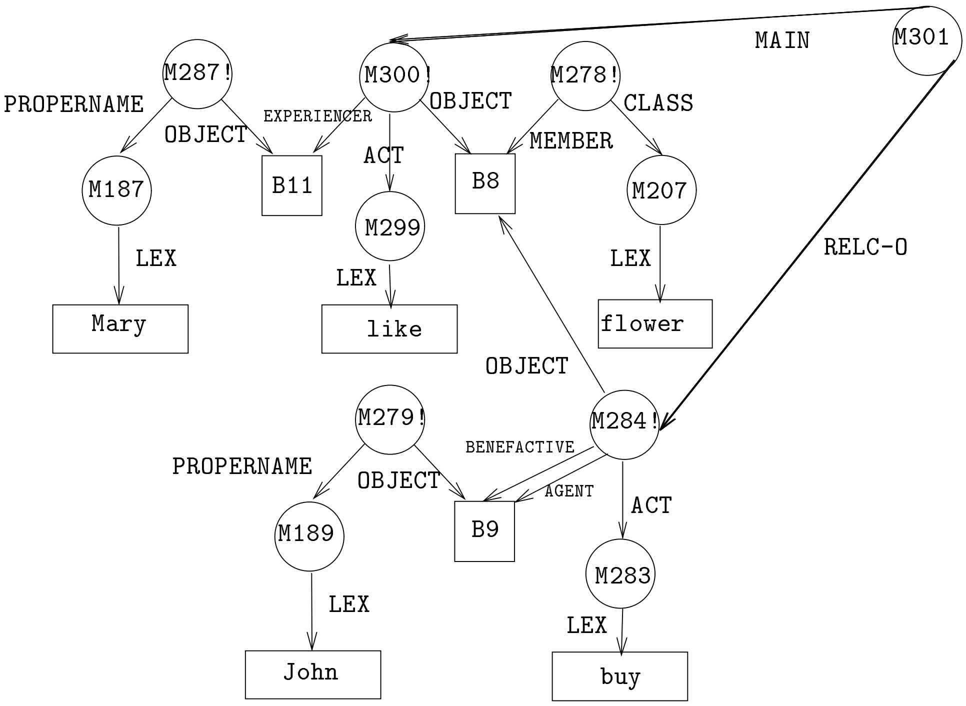

Figure 3.12: *Mary liked the flower which John bought.*

`Benefactive` role.

- Representation of XCOMPlement

  There is a class of verbs that must take (i.e., subcategorize for) an embedded complement, XCOMP, which is a non-finite clause without an overt subject. Either the subject or the object of the matrix verb must functionally control the lower subject of the non-finite clause. Figure 3.13 illustrates the SNePS representation of the knowledge associated with the proposition *John persuaded Tom to study linguistics.*

- Representation of purpose clauses

  There is another type of verb that does not subcategorize for an XCOMP. When taking an embedded complement, the second verb serves as the goal, purpose or intention of the first verb.

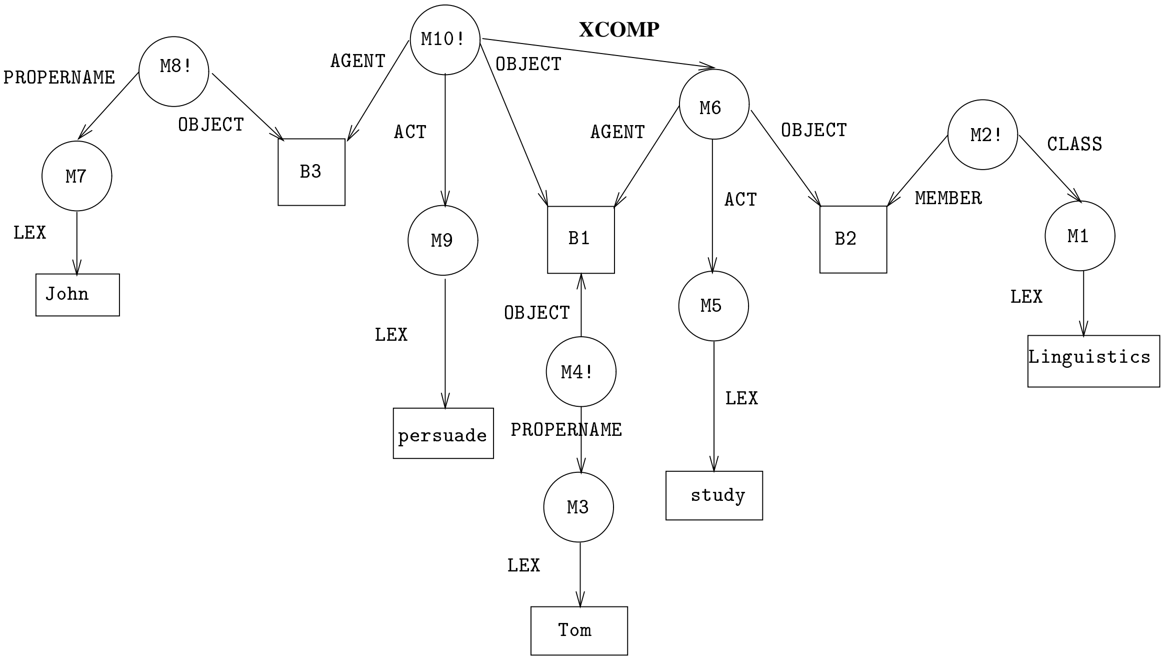

Figure 3.13: *John persuaded Tom to study linguistics.*

Figure 3.14 illustrates how the idea expressed by the English phrase *John bought books to read* can be represented in SNePS.

Please notice that while the propositions for main clauses are asserted, those for complements are not. The three `M6`s in Figure 3.11, 3.13, and 3.14 are not asserted because they are not CASSIE's believes. In the sentence *John believes that Tom loved Mary*, the sentential complement *Tom loved Mary* is John's belief but not CASSIE's. In Figure 3.13, CASSIE believes that John persuaded Tom to do something; however, whether Tom really does it is not certain. In Figure 3.14, CASSIE believes that John bought books. It is also not certain whether he reads them or not.

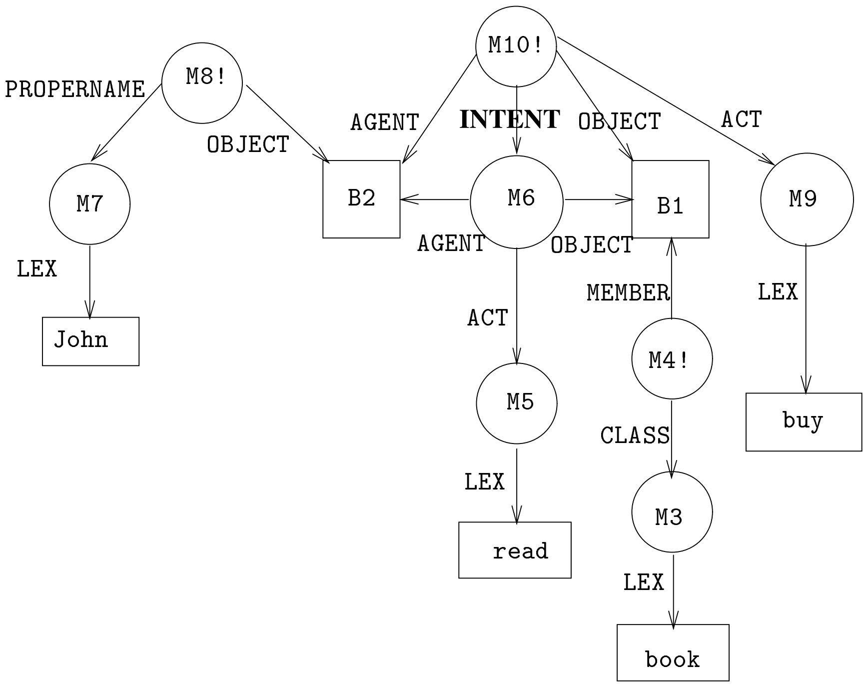

Figure 3.14: *John bought books to read.*

3.2 SNIP

SNePS comes with an “inference engine”, SNIP, the SNePS Inference Package. In our machine translation system, SNIP is a component of the source text disambiguation process. When parsing Chinese, the parser performs semantic role checking and resolves ambiguities by reference to the knowledge base. However, not all semantic information is explicitly stored in the knowledge base. Some semantic information has to be inferred from the existing information.

SNIP provides for node-based reasoning, path-based reasoning, and interactions with SNeBR (SNePS Belief Revision) [Martins and Shapiro, 1988]. Cognitively, node-based reasoning represents conscious reasoning, following explicit “rules” stated in the form of networks. Path-based reasoning, on the other hand, can be thought of as subconscious reasoning; it is a generalization of the notion of “inheritance" found in many other semantic-network systems. [Shapiro, 1978] It is often the case that we combine both types of reasoning. For example, we are given a rule stating that "if X is an animal, then X has a head," and we know that Clyde is an elephant and that all elephants are animals. We would like to know whether Clyde has a head. If we were to do explicit rule-based reasoning only, this information would not enable us to conclude that Clyde has a head. Our subconscious tell us that since Clyde is an elephant and elephants are animals, Clyde is an animal. This implicit (subconscious) reasoning is expressed in SNePS as the path-based inference rule.

```
(define-path class (compose class
                            (kstar (compose subclass- superclass))))
```

We can now go back to explicit reasoning to infer that since Clyde is an animal, Clyde has a head.

The primary advantage that path-based inference has over node-based inference is efficiency. In order to infer that a relation exists between two nodes, one has only to check whether a specified path of arcs goes from one node to the other. Path-based reasoning is just arrow-chasing, while node-based reasoning involves pattern-matching and unification. However, the advantage of node-based inference is that it is more general, since relationships are not restricted to being binary.

3.2.1 Node-based inference

In SNePS' propositional semantic network, all information is stored as nodes. The sentences parsed and new concepts inferred by the system are also stored as nodes. In order to carry out node-based reasoning, we must store rules for deductive inference in the network. Every rule is stored in the system as a rule node. An example of a rule node follows. User input follows the '*' prompt, and subsequent lines show SNePS' output. Some timing and other irrelevant information has been edited out for ease of readability.

```
* (describe (assert forall ($x $y $z)[3]
                    &ant ((build member *x class *y)[4]
                          (build member *y class *z))
                    cq   (build member *x class *z)))
(M1! (FORALL V1 V2 V3)
        (&ANT (P1 (CLASS V2) (MEMBER V1))
              (P2 (CLASS V3) (MEMBER V2)))
        (CQ (P3 (CLASS V3) (MEMBER V1))))
```

The rule node, `M1`, represents the proposition that *if something is a member of a class and this class is contained in another class, then that thing is also a member of that another class.* This rule uses the universal quantifiers. The three universally quantified variables, `x`, `y`, and `z`, are represented by variable nodes, `V1`, `V2`, and `V3` respectively. The antecedents and the consequent of this rule are represented by pattern nodes, `(P1, P2)` and `P3` respectively. The antecedents, `(P1, P2)`, can be read as: `V1` *is a member of the class* `V2` and `V2` *is a subclass of the class* `V3`. The consequent `P3` can be read as: `V1` *is also a member of* `V3`. (Note: For simplicity, in this example, I do not distinguish member-class and superclass-subclass. Superclass-Subclass is represented as `member-class`.)

When trying to infer a consequent of a rule, SNIP tries to find instances of the pattern nodes representing the antecedents. If a match succeeds, then SNIP adds the node representing the consequent to the network. A sample run of the node-based inference follows. Natural language input is on the lines beginning with the ":" prompt. System output is on the other lines. Slight editing is done to remove extra inference reports that are irrelevant.

```
 : Elephants are animals.
   I understand that the object, elephant, is a(n) animal.
 : A circus elephant is an elephant.
```

[3] The `$` macro creates a new variable node.

[4] '`*`' is a macro command function which returns the set of nodes in the value of the SNePSUL variable.[Shapiro and Group, 1994]

```
   I understand that the object, circus elephant, is a(n) elephant.
 : Are circus elephants animals?
I know
((M6! (CLASS (M2 (LEX (elephant))))
      (MEMBER (M5 (LEX (circus elephant))))))
I know
((M4! (CLASS (M3 (LEX (animal))))
      (MEMBER (M2 (LEX (elephant))))))
Since
((M1! (FORALL (V1 <-- M5) (V2 <-- M2) (V3 <-- M3))
         (&ANT (P1 (CLASS (V2 <-- M2)) (MEMBER (V1 <-- M5)))
               (P2 (CLASS (V3 <-- M3)) (MEMBER (V2 <-- M2))))
         (CQ (P3 (CLASS (V3 <-- M3)) (MEMBER (V1 <-- M5))))))
and ((P1 (CLASS (V2 <-- M2)) (MEMBER (V1 <-- M5))))
and ((P2 (CLASS (V3 <-- M3)) (MEMBER (V2 <-- M2))))
I infer ((P3 (CLASS (V3 <-- M3)) (MEMBER (V1 <-- M5))))

I know
((M7! (CLASS (M3 (LEX (animal))))
      (MEMBER (M5 (LEX (circus elephant))))))
Yes.
```

In the above output, `M6` represents the proposition that *a circus elephant is an elephant.* `M4` represents the proposition that *elephants are animals.* From nodes `M1, M4`, and `M6`, SNePS is able to deduce (using node-based inference) that *circus elephants are animals* (node `M7`). Notice that node `M7` was not in the original network. This node has just been created by node-based inference. This was done by the network-match function. The inference is node-based, because it proceeds according to the existence of instances of patterns of nodes: `P1`, which is in antecedent position of a rule, matches `M6` with the substitution `V1/circus elephants`[5]`, V2/elephants`; `P2`, which is also in antecedent position of the rule, matches `M4` with the substitution {`V2/elephants, V3/animals`}. Thus we may infer an instance, node `M7` of the pattern `P3` with the substitution {`V1/circus elephants, V3/animals`}.

[5]This notation reads: substitute `circus elephants` for `V1`

3.2.2 Path-base inference

Path-based inference allows the existence of a "virtual" arc between two nodes to be inferred whenever a path of arcs exists between those nodes. That is, a path of arcs can be defined as a "virtual" arc. Then, this "virtual" arc is treated like a real arc in network pattern-matching to allow property inheritance within generalization hierarchies. The structural inheritance networks have automatic inheritance features; in a SNePS propositional semantic network, inheritance is generalized to path-based inference.

We can, for example, implement the inference involved in the previous example as an instance of path-based inference as follows:

```
(define-path member (compose member (kstar (compose class- ! member))))
```

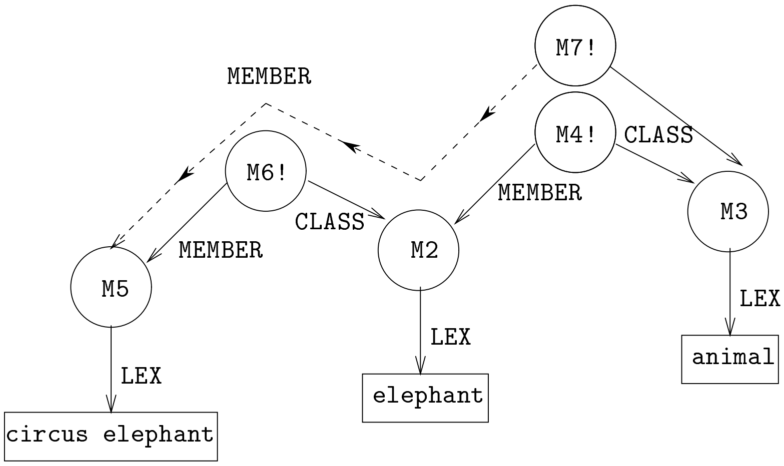

Figure 3.15: *The dashed line is a virtual* `member` *arc composed of the sequence of arcs* `member/class-/member` *from node* `M4` *to node* `M5`*. Note: "/" represents composition of arcs. An arc appended with "-" represents a converse arc.*

This rule has the following interpretation: a virtual `member` arc is equivalent to a path of arcs consisting of a `member` arc followed by zero or more occurrences of converse `class` arcs followed by a `member` arc. Using the previous example, a virtual `member` arc can be inferred since there exists a path of arcs: the `member` arc, from node `M4` to node `M2`, followed by the converse `class` arc, from node `M2` to node

M6, followed by the member arc, from node M6 to node M5. Following along this virtual member arc to node M5, SNePS derives a new node M7 from the previous example. However, the way M7 is derived is very different from the previous derivation. Rather than relying on the existence of instances of patterns of nodes, this relies on the existence of a sequence of arcs i.e., a path from one node to another.

During parsing, we use path-based inference to deduce whether a case slot satisfies the required selectional restrictions. For an example on how path-based inference is used for case-role checking, please refer to section 4.4.1. Besides property inheritance, a path can be defined to help the parser find certain kind of nodes in the network.

After a semantic network is built for a sentence just parsed, we need to find appropriate nodes among the network and assert them. To assert a node is to make CASSIE believe it. An asserted node has an exclamation point appended to its identifier. Let us look at the semantic network in Figure 3.12. Not all molecular nodes are asserted. For example, nodes M300!, M284!, and M278! are asserted whereas nodes M207 and M301 are not. Basically, nodes representing the propositions that CASSIE believes are asserted. Therefore, the molecular nodes representing the main proposition e.g. M300!, relative clauses e.g. M284!, and attributive propositions e.g. M278! are asserted. The molecular nodes representing structured individuals e.g. M207, and the propositions for complements e.g. M6 in Figure 3.11 and Figure 3.13 are not asserted. The following paths allow SNIP to search the network generated by the final parse so that the appropriate propositions can be found, asserted, and described.

```
(define-path describe_assert
    (compose
     (or member (compose (or (compose property property-)
                             (compose propername propername-)
                             (compose adposition adposition-))
                         object))
     (kplus (or agent- object- benefactive- experiencer- locative-))
```

```
    (kstar (compose (kstar (compose
                            (or relc-a- relc-o- relc-e-
                                relc-b- relc-l-)
                            main))
                    (kstar (or comp- intent- xcomp-))))))
```

```
(define-path relative-clauses
    (kstar (compose
            (kplus (compose
                    (or relc-a- relc-o- relc-e-
                        relc-b- relc-l-)
                    main))
            (kstar (or comp- intent- xcomp-)))))
```

In order to handle a variety of network structures and because any network can be nested inside another network, the paths look complicated. For example, we do not assert the node representing a complement; however, the nodes, embedded inside it, representing attributive propositions and relative clauses will still be found and asserted. We are not going to explain the paths above, because complicated illustrations would be involved to properly explain them and a huge semantic network needs to be drawn on the paper. Interested readers can refer to the SNePS User's Manual ([Shapiro and Group, 1994]) for the description of the path defining notations such as `compose`, `and`, `or`, `kstar`, and `kplus`.

We could have asserted the nodes along the way of parsing a sentence as other CASSIE projects did. When parsing a sentence, the parser has the knowledge of currently what grammatical component e.g. the main clause, a relative clause, or a complement, is being parsed; therefore, right after building the SNePS representation for it, the parser can decide whether to assert it or not. However, in this project, we are trying to analyze complicated Chinese sentence structures such as relative clauses, serial nouns, serial-verb construction, which could be highly ambiguous; therefore, our parsing here involves a lot of backtrackings in which some nodes can be generated from invalid parses. The fact that we do not assert nodes until the final parse is available prevents temporary nodes generated in the invalid parse from being asserted. As a result, CASSIE will not have false belief.

```
(define-path head_noun
   (and
      (or agent- object- experiencer- benefactive- locative-)
      (or (agent- (kstar (or xcomp- intent-)) main- relc-a)
          (object- (kstar (or xcomp- intent-)) main- relc-o)
          (experiencer- (kstar (or xcomp- intent-)) main- relc-e)
          (benefactive- (kstar (or xcomp- intent-)) main- relc-b)
          (locative- (kstar (or xcomp- intent-)) main- relc-l))))
```

During generation, the path-based inference rule above helps locate the base node that represents the head noun of a relative clause. For example, Figure 3.12 depicts the network that represents the sentence, *Mary liked the flower which John bought.* Node `M284` represents the proposition for the relative clause, *John bought the flower.* The head noun of the relative clause is *the flower* represented by the node `B8`. This node needs to be found and translated first. Instead of its original surface form *flower*, it is generated into the relative pronoun *which*. The noun phrase *the flower* appended with the relative clause, *which John bought,* becomes *the flower which John bought*.

Between `B9`/*John* and `B8`/*flower*, the generator has to find which node represents the head noun. We find it using a SNePSUL command like this: `(find head_noun M284)`, which returns `B8`. The virtual arc `head_noun` is a path from the node representing the head noun to the node `M284` representing the relative clause. The decision about which relative pronouns we choose is also based on the result of the path-based inference. We deduce whether the head noun of the relative clause is a human being. If yes, then we use *who*; otherwise, we use *which*.

3.3 SNaLPS

The SNePS Natural Language Processing System consists of a Generalized Augmented Transition Network (GATN)[Shapiro, 1982, Shapiro and Group, 1994, Shapiro, 1989] grammar interpreter and compiler for user-defined grammars for natural or formal language interfacing, an English morphological analyzer for handling morphological inflections, and an English morphological synthesizer for constructing inflected words. In this project, since we parse Chinese input, the English morphological analyzer is not used.

3.3.1 Grammar automata

Chomsky [Chomsky, 1963] identified four types of grammars: type-0, type-1, type-2, and type-3. Type-0 grammars have the greatest generative power, while type-3 grammars are the most constrained and thus are the least powerful.

Finite-state automata

A finite-state grammar corresponds to a type-3 grammar in Chomsky's hierarchy. The number of distinguishable states is finite. The language processing starts from an initial state. During language processing, the system is in any one of a finite number of states, and the only thing that matters is which state we are in. The portion of a sentence that we have already processed is not relevant to the correct continuation of the process. At the end of the process, if we can reach a terminal state, then the input string is accepted. Finite-state grammars provide a simple mechanism for language analysis and synthesis.

Recursive transition networks

Recursive transition networks (RTN) are the extension of finite-state grammars. RTN grammars are the equivalent of type-2 grammars in Chomsky's hierarchy. An RTN is like a finite-state grammar except that RTNs can have registers and labeled arcs whose labels are the same as state names. Since they are implemented on push-down automata allowing non-terminal symbols to be the arc labels, they are more powerful than finite-state grammars. Non-terminal symbols, such as NP, name the subnetworks of the grammar. When an PUSH arc with a non-terminal symbol is encountered, the subnetwork named by the non-terminal symbol is pushed onto the stack, and the parse continues at the subnetwork. It is recursive, because any subnetwork can call any subnetwork, including itself, recursively without limit. When a final state in a subnetwork is reached, the subnetwork is popped out from the stack through the POP arc and control is handed back to the calling state. A sentence is said to be accepted if a final state at the top level is reached, and the input buffer is empty.

Besides PUSH and POP arcs, there are CAT arcs, which are matched against the lexical category of any sense of the current word; WRD arcs, which compare the current word to the word list; and JUMP arcs, which succeed without consuming the input buffer.

The RTN formalism also provides a backtracking mechanism. If parsing fails at some arc, the system backtracks to the point where the previous choice was made, takes an alternative arc, and parsing proceeds from there. Backtracking assures a complete search. If there is a plausible reading of the input sentence, the system is guaranteed to find it. If there are multiple readings, the system will find all of them. However, backtracking can be very costly. It is up to the grammar writer to order the arcs carefully and put proper constraints on the tests of an arc so as to minimize unnecessary backtrackings and enhance efficiency.

Augmented transition networks

An augmentation of the RTN is the augmented transition networks (ATN). ATNs are RTNs extended with the abilities to test and to perform actions[Allen, 1987]. With this extension, ATN grammars are able to deal with certain forms of relationships between constituents; thus, they are context-sensitive grammars, or type-1 grammars in Chomsky's hierarchy. The test part of an arc controls the transition. If the test fails, returning nil, the arc can not be reached; otherwise, destination of the parsing proceeds as the grammar specifies. In actions, information about the sentence being examined are saved in registers, and parse trees are built. Actions also provide the means for passing information, stored in the registers, between different levels. Whenever a subnetwork is pushed, a new set of registers in the form of an associative list of key/value pairs is created. The value recorded in a register can be any grammatical information or semantic-network nodes. When the subnetwork pops back, some values of the registers are recorded in the registers of the higher-level networks. The linguistic features at the lower-level networks can then be accessed at the higher-level networks to check for grammatical dependency or agreement in number, gender or, person. Also, the semantic nodes built at the lower levels can be connected together at the higher levels to form bigger semantic networks.

Generalized augmented transition networks

Generalized augmented transition networks (GATN;[Shapiro, 1982]) allow a single ATN interpreter and grammar to be used both for standard ATN parsing and for generation. It is made possible by supplying consistent semantics for both parsing and generation grammars. In fact, in a GATN, both parsing and generation are forms of parsing. During parsing, we parse natural-language surface strings into semantic networks; during generation, we "parse" semantic networks into natural-language surface strings. With a

GATN, we can write grammars for parsing into, and generating from, semantic networks. In this manner, we can express in natural language the concepts represented by the semantic-network nodes. Usually a single lexicon may be used for both parsing and generating a single language; however, in this machine-translation project, two lexicons are used, one for parsing Chinese and the other for generating English.

3.3.2 The organization of the lexicon

All the grammatical or functional information in a sentence comes from the words in it. The lexicon thus plays an essential role. A lexicon consists of several lexical entries. A lexical entry contains not only static syntactic information such as the lexeme, syntactic category, and grammatical constraints but also functional control[6]. The encoding of a lexical entry is composed of two parts: a lexeme and feature lists. In our system, the lexeme of each Chinese lexical entry is a string of Chinese character(s). A feature list is a list of feature-value pairs, each of which comprises a feature name and one or more corresponding values. For example, the syntactic category of a lexical entry is represented as a feature-value pair whose feature is `ctgy` and whose value is the syntactic category of the lexeme.

```
("給" ((ctgy . v)                                                    (3.1)
       (case_frame (Agent animal)
                   (Object stuff power title time nonhuman)
                   (Benefactive animal))
       (surface_arguments . ("AVBO" "OAVB"))
       (benefactive . positive)
       (sense . "give")))
```

```
("語言學" ((ctgy . n)                                                 (3.2)
           (superclass . knowledge)
           (sense . "linguistics")))
```

[6]The *functional control* here is a LFG notion.

The first lexical entry is the lexeme 給. Its syntactic category `ctgy` is `v`, a verb. The value of the `case_frame` feature is an associative list, which associates each case role with its semantic constraints.

```
((Agent animal)
 (Object stuff power title time nonhuman)
 (Benefactive animal))
```

The fillers of the `Agent` role and the `Benefactive` roles should belong to the class of `animal`. The filler of the `Object` role should be something in the classes of `stuff`, `power`, `title`, `time`, or `nonhuman`. The `surface_arguments` has the list `("AVBO" "OAVB")` as its value. This list of possible surface orderings of the semantic cases carries the syntactic as well as the semantic constraints for the verb. The capital letter `V` is the abbreviation for `Verb`, `A` for the `Agent` role, `O` for the `Object` role, and `B` for the Benefactive role. While the Agent role must precede the verb and the Benefactive role always follows the verb. The `Object` role can be either in sentence initial or sentence final positions. The value of the feature `benefactive` is binary, `positive` or `negative`. The feature `sense` gives the lexeme's meaning in English. The second entry is a noun entry. Its `superclass` feature shows that this noun is under the class `knowledge`.

A description of the standard lexical features and their values can be found in the SNePS 2.3 User's Manual [Shapiro and Group, 1994]. I will only describe the additional features and their values used in this project.

Multi-character words and multi-word phrases

In addition to a usual lexical entry, a compound, multi-character lexeme, has its last character as the head of another lexical entry whose lexical category is `multi-end`. The compound has the `multi-head` feature, whose value is a list of the rest of the characters (before the last one) that form the compound. For example, the compound, 語言學 *linguistics*, has two lexical entries as follows:

```
("語言學" ((ctgy . n)
          (superclass . knowledge)
          (sense . "linguistics")))
("學" ((ctgy . multi-end) (multi-head . ("語" "言"))))
```

Beyond the word level, multi-character lexical entries also facilitate the analysis of multi-word phrases such as "Dian4 Gi1 Gon1 shen2 si4" (*Electrical Engineering Department*). In fact, this algorithm is modified from the SNaLPS built-in function `get-senses`, which can handle English multi-word phrases such as *State University of New York*.

To show how the system establishes the word or phrase boundary, let us examine the following sentence:

張三	教	李四	語言學
Zhang1 san1	jiao1	Li3 si4	Yu3 yang2 xue2
Zhang1san1	teach	Li3si4	linguistics

Zhang1san1 teaches Li3si4 linguistics. (3.3)

The parser takes the words of the input sentences from right to left. The last character, 學 *xue2*, can be found in several lexical entries. For example:

```
("學" ((ctgy . v)                                                    (3.4)
      (case_frame (Agent human person)
                  (Experiencer human person)
                  (Object knowledge))
      (surface_arguments . ("ACVO"))
      (sense . "learn"))) ;'C' denotes Agent/Experiencer coreferential

("學" ((ctgy . multi-end) (multi-head . ("語" "言"))))               (3.5)
```

The second lexical entry is then mapped to its multi-character counterpart:

```
("語言學" ((ctgy . n)                                                (3.6)
          (superclass . knowledge)
          (sense . "linguistics")))
```

Both senses, 3.4 and 3.6, will be tested accordingly. The parser first takes the string, 學 *learn*, as a verb. When the input sentence is exhausted, the parser will get the word segmentation as follows:

張三	教	李四	語言	學	(3.7)
Zhang1 san1	jiao1	Li3 si4	Yu3 yang2	xue2	
Zhang1san1	teach	Li3si4	language	learn	

This path proves to be wrong, since no parse can be produced. Then the parser backtracks and tests the second sense, 語言學 *linguistics.* Following this path, the parser gets the following word segmentation:

張三	教	李四	語言學	(3.8)
Zhang1 san1	jiao1	Li3 si4	Yu3 yang2 xue2	
Zhang1san1	teach	Li3si4	linguistics	

This time, the parse succeeds with the reading *Zhang1san1 teaches Li2si4 linguistics.*

It is not the case that the parser parses sentence after segmenting the whole sentence. Morphological and syntactical parsing are mingled together. SNaLPS embeds the above morphological analysis within the `cat` arc, so backtracking is fully enforced over all possible word segmentations. If there are several valid segmentations, the parser will return as many parses.

Noun and propername (NPR) entries

The `superclass, property, ability`, and `part` features indicate the lexeme's parent, properties, abilities, and parts respectively.

```
("李四" ((ctgy . npr)                                    (3.9)
        (superclass . man)
        (sense . "Li3si4")))

("牛奶" ((ctgy . n)                                      (3.10)
        (superclass . food)
```

```
         (property . liquid)
         (sense . "milk")))
```

```
("鳥" ((ctgy . n)                                                    (3.11)
       (superclass . animal)
       (ability . fly)
       (sense . "bird")))
```

```
("電視" ((ctgy . n)                                                  (3.12)
         (superclass . equipment)
         (part antenna screen)
         (sense . "television")))
```

The example 3.9 indicates the Chinese `npr`, 牛奶, is a `man`'s propername. 3.10 shows that the noun 牛奶 meaning `milk` is a kind of `food` with the `property` of the `liquid`. 3.11 shows that 鳥 meaning the `bird` is an `animal` with the `ability` to `fly`. The noun entry 電視 *television* in 3.12 is an `equipment` with the parts `antenna` and `screen`.

Unlike other features whose values are accessed from the lexicon via the SNaLPS `getf` (get feature) function, these pieces of information are retrieved from the SNePS semantic network by the SNePSUL `find` command or via SNIP's path-based inference. Before parsing starts, we read the `KB` file (Knowledge base; Appendix A) and build the SNePS knowledge base according to this specially formatted file. The semantic network thus built is an inheritance hierarchy, and it forms the trunk and branches of the hierarchy. Then, we open the Chinese `lexicon` file, read the above feature/value pairs in the noun and NPR entries, and extend the knowledge base. The nodes so built are automatically attached to the inheritance hierarchy. They form the leaves of the inheritance hierarchy. Let us take a mini version of the KB file for illustration:

```
(noun (concrete abstract))
(concrete (animate inanimate))
(animate (plant (animal (property mobile))))
```

```
(animal (human nonhuman))
(human (woman man))
```

The system starts off by processing this file and building the following semantic networks as the background information:

```
(M3!  (SUPERCLASS (M2  (LEX noun)))    (SUBCLASS (M1  (LEX concrete)))
(M5!  (SUPERCLASS (M2  (LEX noun)))    (SUBCLASS (M4  (LEX abstract)))
(M7!  (SUPERCLASS (M1  (LEX concrete)))(SUBCLASS (M6  (LEX animate)))
(M9!  (SUPERCLASS (M1  (LEX concrete)))(SUBCLASS (M8  (LEX inanimate)))
(M11! (SUPERCLASS (M6  (LEX animate))) (SUBCLASS (M10 (LEX plant)))
(M13! (SUPERCLASS (M6  (LEX animate))) (SUBCLASS (M12 (LEX animal)))
(M15! (OBJECT     (M12 (LEX animal)))  (PROPERTY (M14 (LEX mobile))))
(M17! (SUPERCLASS (M12 (LEX animal)))  (SUBCLASS (M16 (LEX human)))
(M19! (SUPERCLASS (M12 (LEX animal)))  (SUBCLASS (M18 (LEX nonhuman)))
(M21! (SUPERCLASS (M16 (LEX human)))   (SUBCLASS (M20 (LEX woman)))
(M23! (SUPERCLASS (M16 (LEX human)))   (SUBCLASS (M22 (LEX man)))
```

Then the Chinese lexicon is processed. The information in the lexicon is added to the knowledge base. For example, the following semantic network built according to the lexical feature `(superclass . man)` in example 3.9 will be attached to the inheritance hierarchy above through the node `(M22 (LEX man))`.

```
(M25! (SUPERCLASS (M22 (LEX man))) (SUBCLASS (M24 (LEX Li3si4)))
```

In a pure ISA hierarchy, to represent common properties among various objects, one has to invent a virtual superclass. For example, cars, ships, and dogs are subordinated under a virtual class such as movable-objects. Therefore, there will be as many top level superclasses as the number of properties or attributes. Our taxonomy contains more than just a superclass-subclass relation. The knowledge base is also composed of *semantic features* such as locative, *relations* such as partship and kinship, *properties* such as edible, and *abilities* such as flying. Rather than being subordinated under the superclass, movable-object, as the pure ISA hierarchy does, cars, dogs, and any new members are tagged with the property *mobile*. Since SNePS is a propositional semantic network system, any knowledge needed to enforce the semantic restriction can be utilized. By inference, path-based or node-based, all properties, relations, abilities, and features can be inherited down or up the hierarchy just as properties in an inheritance semantic networks can be.

By inheritance, common features are passed on from superclass to subclass. Features do not need to be stored with every object in the knowledge base. Rules and paths can be defined and SNePS is able to derive the features. Therefore, we only have to record new or unique features for the objects in the lexical entries. The inheritance mechanism ensures knowledge consistency, saves memory space, and provides an efficient mode of representation. That SNIP is able to supply information that is not specifically stored in the memory simulates a desirable property of human cognition i.e., the capability of using the memory inferentially. From a cognitive point of view, SNePS provides a framework in which the semantic information is stored as networked concepts and a mechanism to use these concepts for inferences. As a computer model for natural-language understanding, CASSIE uses the semantic network to represent linguistic properties such as word features and their relations efficiently.

The knowledge base is dynamic. A modifiable knowledge base is essential to natural-language understanding [Rapaport, 1988]. Taxonomic hierarchies should also be acquired by intelligent systems through natural-language interaction. Since both the inheritance hierarchy and the intermediate result of translations are part of the semantic network, we can expand or revise the noun hierarchy through natural-language interaction with SNePS. The newly asserted properties can be passed down through the hierarchy immediately. The dynamics of the knowledge base are reflected in that the same Chinese input may be translated into different English sentence(s) due to a change of the knowledge base.

For example, consider the following Chinese input:

李四	給	志成	兒子	一	本	書	(3.13)
Li3 si4	gei3	Zhi4 cheng2	er2 zi2	yi1	ben3	shu1.	
Li3si4	give	Zhi4cheng2	son	one	CL	book	

Before CASSIE is told that Zhi4cheng2 has a son, this Chinese sentence will be translated as: *Li3si4 gave Zhi4cheng2 and son a book.* This *son* is an arbitrary one whose relations with Li3si4 or Zhi4cheng2 are unspecified. This *son* can belong to Li3si4, Zhi4cheng2, or an unspecified third person. After we tell CASSIE the fact that Zhi4cheng2 has a son, this Chinese sentence will also be translated as *Li3si4 gave Zhi4cheng2's son a book*, in addition to the first translation. The "son" in *Zhi4cheng2 has a son* and the "son" in *Zhi4cheng2's son* are denoted by the same SNePS base node. That our translations vary with a change in the knowledge base is one of the most notable characteristics of the system developed here.

Verb entry

The `case_frame` template, an association list, indicates, for each predicate, possible slots in its frame structure and associates each case role with its filler constraints. The case frame utilizes the case system proposed by Walter Cook [Cook, 1989]. The verb 教 *teach*, as in the following sentence:

張三	教	李四	英文
Zhang1 san1	jiao1	Li3 si4	Ying1 wen2
Zhang1san1	teach	Li3si4	English

Zhang1san1 teaches Li3si4 English. (3.14)

would have the following entry:

```
("教" ((ctgy . v)                                          (3.15)
        (case_frame (Agent animal)
                    (Object knowledge skill)
                    (Experiencer animal))
        (surface_arguments . ("AVEO" "AVO" "OAVE"))
        (sense . "teach")))
```

The verb 教 *teach* subcategorizes for an Agent, an Object and an Experiencer. The Agent, who does the teaching, should be an animal. The Object of the teaching should be some knowledge or skills. The Experiencer, who undergoes the cognitive process, should be an animal as well. Note that we can specify more than one selectional restriction for a case role as in the case of `(Object knowledge skill)` in our example. The parser will check an argument against them one by one. If the argument meets any one of these filler constraints, then this argument is assigned the corresponding case role.

The selectional restrictions are not limited to the object's categories. They can include any properties, features, abilities, and other attributes. For example,

```
("飛" ((ctgy . v)                                                    (3.16)
       (case_frame (Agent can_fly)
                   (Object aircraft)
                   (Locative property_locative))
       (surface_arguments . ("AV" "AVO" "AVL" "ALV"))
       (sense . "fly")))
```

The verb 飛 *fly* requires its Agent to be capable of flying and its Locative role to have the property of locative.

```
("吃" ((ctgy . v)                                                    (3.17)
       (case_frame (Agent animal)
                   (Object food meal property_edible))
       (surface_arguments . ("AVO" "AOV" "OAV" "UOV"))
       (sense . "eat")))
```

The Object role of the verb 吃 *eat* should be the food, meal, or something edible.

The feature `surface_arguments` describes the predicate-argument structure. The argument structure provides two types of information: the legal order of the surface realizations of its case frame and the number of arguments required by the predicate.

Chinese is a relatively free-word-order language. In other words, a verb may be realized with different orderings of its case roles. However, Chinese word order is not so free as to allow every distribution. The possible case sequences of a verb are specified in the feature `surface_arguments`. The syntactic orders of the case roles are abbreviated. For the verb 教 *teach*, three grammatical orders, i.e., `AVEO`, `AVO`, and `OAVE`, are allowed. "AVEO" is the abbreviation of `Agent Verb Experiencer Object`, "AVO" of `Agent Verb Object`, and "OAVE" of `Object Agent Verb Experiencer`. Besides the different orderings of semantic cases, a verb may have different numbers of semantic cases. For example, for the verb *fly*, we can say *Dumbo flies* or *Dumbo flies in the sky.* Sometimes, different numbers of arguments represent different senses. For example, the *fly* in *Mary flies a fighter* means something differently from *flies* in *Dumbo flies.* The second *fly* means *to move in the air by flapping the body parts, e.g., wings or, in this example, ears*, while the first *fly* means *to operate an aircraft*.

The values, "AV", "EV", or "BV" tell the parser that the predicate needs one argument syntactically; the values, "AVO" and "AVL" imply that the number of arguments is two; "AVEO" implies that the number of arguments is three and so on. When case checking is performed, the parser first checks if the number of the elements in the argument stack [7] matches the number of arguments implied by the value of `surface_arguments`. If the number matches, then the parser goes on mapping the syntactic arguments onto the corresponding semantic case roles. The sentence is accepted when the mapping succeeds. Finally, the case roles of the syntactic arguments are assigned according to the mapping. This feature of the lexical entries of English verbs also guides the generation of English sentences. For example, the value "AVO" tells the generator first to generate a noun phrase for the Agent role, then the verb, and at last a noun phrase for the Object role.

[7] For a discussion on how the argument stack is filled, please refer to section 4.3

`Surface_arguments` also has to specify covert case roles that are not realized in the surface string. Covert case roles include the unspecified Agent in a passive sentence and coreferential case roles. For example, the "U" in "UOV" stands for the unspecified Agent in a passive sentence. "C" denotes the covert Experiencer coreferential with the Agent, as in the sentence *John studies linguistics.* Since John is the agent who initiates the action of studying and he is also the experiencer who undergoes the cognitive process of learning; therefore, the verb *study* has "ACVO" as the value of its `surface_arguments`. The parser has to be notified so that when the parser checks the syntactical well-formedness of the surface string, those covert case roles will not conflict with the surface structure and will be built in the semantic network. Therefore, the sentence *John studies linguistics* will have the following SNePS representation:

```
(M2! (PROPERNAME (M1 (LEX John))) (OBJECT B1))
(M4! (CLASS (M3 (LEX linguistics))) (MEMBER B2))
(M6! (AGENT B1)
     (EXPERIENCER B1)
     (ACT (M5 (LEX study)))
     (OBJECT B2))
```

We see, from above, that the covert Experiencer role is represented explicitly with an `EXPERIENCER` arc. `EXPERIENCER` arc and `AGENT` arc point to the same node `B1` because the two roles, Agent and Experiencer, are coreferential.

The feature `Xcomp` defines the *coreferential* relation between the missing subject of the controlled non-finite clause, designated by XCOMP, and its antecedent. Three types of relation are defined, namely *ObjControl, SubjControl* and *NoObj*. Basically, these are the same as the functional control proposed by Lexical-Functional Grammar (LFG) [Kaplan and Bresnan, 1982]. *ObjControl* stands for "Object Control," *SubjControl* for "Subject Control," and *NoObj* for "No Object." *Object Control* means that the *object* of the main verb functionally controls the subject of the predicate of the subordinate clause. As for both *Subject Control* and *No Object*, the subject of the matrix verb and the subject of the controlled clause are coreferential. The difference between the two relations is that the main verb has an object for *Subject Control,* while in the latter case no object is found in the main clause.

Consider the sentence whose control verb 勸 *persuade* would have the following entry:

張三	勸	李四	去
Zhang1 san1	quan4	Li3 si4	qu4
Zhang1san1	persuade	Li3si4	go

Zhang1san1 persuaded Li3si4 to go. (3.18)

```
("勸" ((ctgy . v)                                              (3.19)
       (case_frame (Agent human person)
                   (Object human person))
       (Xcomp ObjControl)
       (surface_arguments . ("AVO"))
       (sense . "persuade")))
```

The value of the feature `Xcomp` being "ObjControl" means that the object of the main clause *Zhang1san1* controls the unexpressed subject in the controlled clause. The subject of *go* is therefore to be identified with *Li3si4*. The verb 勸 *persuade* subcategorizes for an Agent, an Object, and an Xcomp. The Agent and Object roles have the properties of `human` (being) or `person`(or pronoun) as their selectional restrictions.

The SNePS network representation for the control sentence roughly corresponds to an LFG f-structure representation. There are some differences in the actual representations. Compare the f-structure in LFG and the semantic-network representation in SNePS:

f-structure in LFG:

```
SUBJ  PRED 'Zhang1san1'
PRED  'persuade<(↑SUBJ)(↑OBJ)(↑XCOMP)>'
OBJ   PRED 'Li3si4'
XCOMP SUBJ [^OBJ]
      PRED 'go<(↑SUBJ)>'
```

SNePS semantic network representation:

```
(M2! (PROPERNAME (M1 (LEX Zhang1san1))) (OBJECT B1))
(M4! (PROPERNAME (M3 (LEX Li3si4)))     (OBJECT B2))

(M8! (AGENT B1)
     (ACT (M5 (LEX persuade)))
     (OBJECT B2)
     (XCOMP (M7 (AGENT B2)
                (ACT (M6 (LEX go))))))
```

The major difference between these two representations is that instead of using syntactical notation, e.g., SUBJ and OBJ, as in LFG, this SNePS representation uses case roles to denote the coreferential relation. The syntactic notations in f-structure are language dependent; thus, it is not "deep" enough to capture the underlying structure of a sentence. In this respect, this representation goes one step further to represent the referential relation between the controller and the controlled with semantic cases that are found to be universal across languages.

Now we come to see another sentence:

張三	想要	李四	買	書
Zhang1 san1	xiang3 yao4	Li3 si4	mai3	shu1
Zhang1san1	want	Li3si4	buy	book

Zhang1san1 wanted Li3si4 to buy books. (3.20)

f-structure in LFG:

```
SUBJ  PRED 'Zhang1san1'
PRED  'want<(↑SUBJ)(↑OBJ)(↑XCOMP)>'
OBJ   PRED 'Li3si4'
XCOMP SUBJ [^OBJ]
      PRED 'buy<(↑SUBJ)(↑OBJ)>'
      OBJ  PRED 'book'
```

SNePS semantic network representation:

```
(M2! (PROPERNAME (M1 (LEX Zhang1san1))) (OBJECT B1))

(M4! (PROPERNAME (M3 (LEX Li3si4))) (OBJECT B2))

(M6! (CLASS (M5 (LEX book))) (MEMBER B3))

(M10! (EXPERIENCER B1)
      (ACT (M7 (LEX want)))
      (OBJECT B2)
      (XCOMP (M9 (AGENT B2)
                 (BENEFACTIVE B2)
                 (ACT (M8 (LEX buy)))
                 (OBJ B3))))
```

This example shows that the controller is not necessarily the Agent. It can be any role, i.e., Agent, Experiencer, or Benefactive, in the subject position of the main clause. And the controlled element is not limited to one role. Being coreferential, both the Agent and the Benefactive of the XCOMP are controlled by the Object role of the matrix verb 想要 *want*. The *Object Control* in this project can be defined as: The case roles in the *object* position of the main verb functionally controls the case roles in the *subject* position of the lower clause. Similarly, *Subject Control* and *No Object* can be defined as: The case roles in the *subject* position of the main verb functionally controls the case roles in the *subject* position of the embedded nonfinite clause. The verb 想要 *want* in the example above has the following lexical entry:

```
("想要" ((ctgy . v)                                                    (3.21)
         (case_frame (Experiencer animal)
                     (Object noun))
         (surface_arguments . ("EVO" "OEV" "EV"))
         (Xcomp ObjControl NoObj)
         (sense . "want")))
```

We may find it redundant to use two names for similar referential relations, i.e., *Subject Control* and *NoObject*. The above lexical entry shows that if we use *SubjControl* in place of *NoObj* then the parse cannot decide whether the controller of the XCOMP is in the subject position or in the object position of the higher clause. *NoObj* is a special case of subject control, occurring only when the object of the verb is not present, as in the sentence:

張三	想要	去
Zhang1 san1	xiang3 yao4	qu4
Zhang1san1	want	go

Zhang1san1 wanted to go. (3.22)

f-structure in LFG:

```
SUBJ  PRED 'Zhang1san1'
PRED  'want<(↑SUBJ)(↑XCOMP)>'
XCOMP SUBJ [^SUBJ]
      PRED 'go<(↑SUBJ)>'
```

SNePS semantic network representation:

```
(M2! (PROPERNAME (M1 (LEX Zhang1san1))) (OBJECT B1))

(M6! (EXPERIENCER B1)
     (ACT (M3 (LEX want)))
     (XCOMP (M5 (AGENT B1)
                (ACT (M4 (LEX go))))))
```

The feature `obl` is used in generation to specify the marker of the oblique function. For example, the verb *give* has the feature pair `(obl . "to")` as in the sentence “John gave the money to Mary,” and the verb *steal* has the feature pair `(obl . "from")` as in the sentence “John stole money from Mary.”

The generator consults this feature to know which preposition to use when generating a prepositional phrase. For example, the verb *congratulate* has “on” as the value of this feature, as in the sentence *Li3si4 congratulated Zhang1san1 on getting a job.*

```
("adorn" ((ctgy . v)
          (case_frame (Agent human person)
                      (Benefactive animal)
                      (Object jewelry flower))
          (Vpattern V+N+P+N)
          (prep . "with")
          (surface_arguments . ("AVBO" "AVB"))))
```

The feature `vpattern` tells the generator the pattern of the verb or how to generate its XCOMP. A verb with the “V+N+P+N” pattern, e.g., *adorn*, will have an object followed by a prepositional phrase, as in the sentence *Mei3hua2 adorned herself with jewels.* The verb pattern “gerund” tells the generator to gerundize the XCOMP, as in the sentence *Li3si4 considered changing jobs.* If the generator finds the feature `prep` in this verb lexical entry, then the value specified by the feature `prep` will be generated first, and then the XCOMP is gerundized. For example, *insist* requires a preposition “on,” as in the sample sentence *Li3si4 insisted on paying bills.* If it is an object-control sentence, then the generator will turn the object into the genitive and then gerundize the XCOMP, as in the sentence *Li3si4 disliked Zhang1san1’s stealing money.* Another kind of object-control verb does not possessivize the object before gerundizing the XCOMP; instead, it puts a preposition before the gerundized XCOMP. For example, the verb *congratulate* has the verb pattern “V+N+P+Ving” and requires the preposition “on”. Rather than saying **Li3si4 congratulated Zhang1san1’s getting a job*, we generate *Li3si4 congratulated Zhang1san1 on getting a job.* Noted the difference between these two verb patterns: `V+N+P+Ving` and `gerund` with the feature `prep`. The latter is for sentences like: *Officers insisted on soldier’s obeying orders* instead of being: **Officers insisted soldier on obeying orders.* There is a class of verbs called perceptive verbs, such as *see or hear,* which can be followed by either object + infinitive (without *to*) or object + -ing. For example:

Mary saw John cross the road. (3.23)

Mary saw John crossing the road. (3.24)

There is a difference in meaning between the two sentences. The former means that Mary saw the whole action of John's crossing the road. The latter means Mary saw a part of the action. Since both correspond to one Chinese translation, we do not make the distinction in this project. Thus, all perceptive verbs will have the same verb pattern, "perception," and all will be followed by the gerundized verb phrase. The other class of object-control verb does not gerundize the XCOMP. They turn the XCOMP into a that-clause and keep the infinitive form (without *to*) of the verb. For example, the verb *suggest* has the verb pattern "clause," as in the sentence *Li3si4 suggested that Zhang1san1 study linguistics.*

Chapter 4

The Parsing of Chinese

4.1 Parsing strategies

4.1.1 Left to right vs right to left

The default parsing order for a GATN is from left to right. For this project, we take the words of the input sentences from right to left. This decision is based on the observation that in Chinese the modifiers precede the modified. The genitive noun occurs to the left of the possessed noun. Adjectives and relative clauses precede the head noun they modify. If we parse a sentence from left to right, first we have to hold the modifiers in a register; after seeing the head noun, then we start to process the modifiers in the hold register. By parsing sentences from right to left, we can identify the head noun first and build a SNePS base node to represent the entity. As we encounter its modifiers, we build the SNePS network representations for them, e.g., the `object/property` case frame for the adjectives, and simply connect them to the base node. The right-to-left parsing scheme processes left-branching structures, such as Chinese relative clauses, in a straightforward manner; however, it may have difficulty in processing right-branching structures. Our bottom-up parsing strategy, which is also motivated by Chinese grammatical characteristics, overcomes the difficulty. We will discuss the bottom-up strategy in the next section. Right-to-left parsing is done by reversing the input string before we start the parsing.

4.1.2 Top-down vs. bottom-up strategy

Top-down parsing is goal-directed. Two key operations here are *predict* and *match*. It begins at the top level Sentence network and predicts the lower-level constituent structures (e.g., rewriting S to the non-terminal symbols NP and VP). These intermediate constituents are further expanded to constituents of even lower level. The process repeats until it reaches a pre-terminal symbol such as *noun*. This pre-terminal symbol is then used to matched the lexical categories of the lexical item at the front of the input buffer, say *saw*. The match succeeds by taking the *noun* sense of *saw*. The procedure continues testing the rest of the hypotheses against the input sentence until every element of the input sentence is assigned a value.

A bottom-up parser, which is data-driven, does the opposite. Two key operations here are *shift* and *reduce*. It shifts the input tokens onto a stack, assigns them pre-terminal symbols (e.g., noun, verb), combines, i.e. reduces, these symbols into non-terminal symbols (e.g., NP, VP), and then assembles these non-terminal symbols into the highest sentence structure, S.

Top-down parsing requires prediction. Consider two rules as follows:

$\mathtt{A} \rightarrow \mathtt{bca}$
$\mathtt{A} \rightarrow \mathtt{abc} \mid \mathtt{acb} \mid \mathtt{bac} \mid \mathtt{bca}$

To predict is to select one of the right-hand sides and replace the left-hand side with it. Given an input string, say bca, for the first rule, the parser predicts just once to get the match, whereas, for the second rule, the parser has to select four times to find the match. The top-down method will be more efficient if there are less right-hand sides to select from. Therefore, a top-down parser is more suitable for parsing languages with fixed word order such as English. However, in Chinese since word order is relatively loosening, furthermore, phrases and clauses can hold together in simple juxtaposition without grammatical linkings to indicate the syntactic relationship between them, there are much more possibilities lying ahead of the parser. To put it another way, there are fewer syntactic clues for a top-down parser to predict what would be the next constituent to parse. Thus, a top-down parser cannot efficiently parse Chinese. In view of this, at sentence and clause levels, we adopt a bottom-up strategy, which only shifts the constituents parsed (e.g., NP, PP) onto a stack and the decision on what rule applies is postponed to the latest possible moment so that decision can be based on the fullest possible information. For example, the good moment would be the time after the verb is parsed because the verb provides lexical information on how its arguments are ordered.

The bottom-up strategy also make it easy for a right-to-left parser to parse right-branching constructions such as sentential complements. Right-branching structures lead a right-to-left-ordered top-down parser into an endless right-recursion. Consider the processing of the indicative clause, e.g., *John believes Mary saw Pat,* whose phrase-structure rules can be described as follows:

$$
\begin{array}{lll}
\texttt{S} & \longrightarrow & \texttt{NP VP} \\
\texttt{NP} & \longrightarrow & \texttt{S} \mid \texttt{noun} \\
\texttt{VP} & \longrightarrow & \texttt{verb} \mid \texttt{verb NP}
\end{array}
$$

First, `S` is replaced with `NP` and `VP`, which in turn is replaced with `verb` and `NP`. Then, `NP` is replaced with `S`, which again is replaced with `NP` and `VP`. The recursion goes on and on infinitely. The ATN parser cannot tell that the clause *Mary saw Pat* is an indicative clause until it finds the verb *believes*. However, bottom-up parsing allows the parser to "look ahead" an arbitrary distance. Before the verb *believe* is encountered, the parser pushes the constituents in the clause *Mary saw Pat* onto an argument stack. When the verb *believes* is recognized, the elements in the stack are retrieved for semantic case checking and are reduced to a semantic node representing the proposition that *Mary saw Pat*. The node is then returned to the higher sentence-level in which the node becomes the indicative clause of the matrix verb *believes*.

However, bottom-up processing requires more run-time memory to implement the stack. When parsing languages with fixed word order, it is better to employ the top-down method. Chinese word order within the phrase level (e.g., NP, PP, VP) is fixed; therefore, our parser switches to the top-down mode in the `NP`, `PP`, and, `VP` subnetworks of the argument network.

4.2 The sentence network

The parsing process starts from the sentence state `S`, in which the parser does some initial setups such as variable initialization, selecting or switching back to the Chinese lexicon, input-string reversal, and setting sentence mood.

- Variable initialization: In order to facilitate the parsing process, we keep some variables for book-keeping and table look-up. When parsing a new sentence, we have to reset these variables.

- Selecting Chinese lexicon: There are two lexicons: one is the Chinese lexicon for parsing; the other is the English lexicon for generation. In parsing a Chinese sentence, we select the Chinese lexicon. When generating an English sentence, we switch to the English lexicon. We switch the lexicon back to the Chinese lexicon before parsing each new sentence.

- Input string reversal: The GATN interpreter takes the input string from left to right. Since we want to parse the sentence in the right-to-left direction, we first reverse the input sentence. On reversing the input string, we JUMP to the SP (Sentence Parsing) state to find the mood of the sentence.

- Setting sentence mood: The mood of the input sentence is partly determined by the end punctuation. A *period* indicates the *declarative mood.* A *question mark* sets the *interrogative mood*. If both the question mark and the Chinese question marker *ma1* is found at the end of the sentence, then the sentence mood is set to *Yes-No-Q mood* (Yes-No Interrogative mood).

After these initializations, the sentence network calls the argument network, ARGS, to find out the arguments or constituents (including the verbs) of the input sentence. After PUSHing down to the argument network, the parser arrives at the ARGS state. The mood of the sentence is sent down to the ARGS state.

4.3 The argument network

Because of the relatively free word order of Chinese, the parser operates in the bottom-up mode at the ARGS state. There are three subnetworks in the argument network, namely, the NP network (noun phrase), the PP network (adposition network), and the VP network. Without making any hypothesis or prediction,[1] the ARGS network calls its three subnetworks in the order of NP network then Verb network and then PP network to search for the predicate of the sentence or the arguments of the predicate. If any one of these three subnetworks succeeds, whatever popped back from it, an NP, a PP, or a verb, is stored in the argument stack. The parser then loops to the ARGS state to invoke its three subnetworks in the same order to

[1] In the ARGS state, we arrange the PUSH arcs in the order of NP network then V network, and then PP network. If it traverses one arc successfully, the parser will skip the remaining arcs in a state. In the input sentence, there are higher percentages of noun phrases than the combination of the verb phrases and the adposition phrases. The way we order the arcs makes parsing more efficient, since the parser will waste less time on taking wrong arcs.

search for the next constituent on the remaining input string. The parser iterates over the three subnetworks in the `ARGS` state until all of them fail to find any constituents or the input string is exhausted. All the arguments and the predicate successively recognized are pushed onto the argument stack. The parser then `JUMP`s to the `S/END` (sentence end) state where the arguments in the stack will be tested against the case frame of the predicate for their semantic well-formedness. If the case checking procedure proves the argument stack semantically well-formed, then each argument is assigned a proper case. According to the result of case assignment, the SNePS node representing the input sentence is built and stored in the `*` register. Then, the generation of the English translation begins from the G network taking the content of the `*` register as its input.

In the following subsections, we discuss the three subnetworks, the `NP` network, the `PP` network, and the `Verb` network, of the argument network. Before we go over these subnetworks, I will talk about the word-segmentation problem, which is relevant to all three subnetworks.

4.3.1 The word segmentation problem

Since there are no delimiters between two Chinese words, the parser has to find word boundaries while parsing. In Chinese, the differences among one-character words, multi-character words, and noun compound are not clear[Tang, 1989]. Therefore, we must determine the boundaries of compound words and multi-character words. Right before the parser tries to find out the lexical category of the current word, the GATN interpreter calls the function `GET-SENSES` to search for all possible word boundaries from the current point of the input string. The function will return as many words as the number of word boundaries found. Then the parser begins going through the network. If later it fails, then the parser backtracks to take the next word and tries again. The word-segmentation mechanism is built in with the `CAT` arc, all possible word boundaries in the input string will be tried through backtracking. For an example on how the function `GET-SENSES` finds out the word boundaries, please refer to section 4.3.3.

4.3.2 The verb network

We put the verb network in the `ARGS` state, although the verb is the predicate rather than the argument of a sentence or a clause. Like the arguments of sentences, i.e., noun phrases etc., the Chinese verb can also appear in sentence initial, middle or final positions. Due to the free word order among the arguments and the predicate, together with the noun phrase network and the adposition network, we include the verb network in the argument state where the parser operates in bottom-up mode.

Besides parsing the main verb of a sentence, the `Verb` network also parses the modality of the verb, i.e., auxiliary, time, aspect, negation, etc.

- auxiliary: *ying1gai1* (should), *hui4* (will), *kei3yi3* (can)
- time: *tian1tian1* (everyday), *zuo2tian1* (yesterday), *jin1tian1* (today)
- aspect: *guo4* (experiential suffix), *zhe* (durative suffix), *le*[2] (perfective suffix), *zai4* (progressive prefix)
- negation: *bu4*, *mei2*

Our `Verb` network is built according to the network diagram in Figure 4.1. The negative *mei2* does not occur with the future tense. When the parser sees the negative *mei2*, a test is performed on the `tense` register. If it is set to the value `future`, the `Verb` network blocks.

[2] When *le* appears in the sentence-final position, it is a perfective particle.

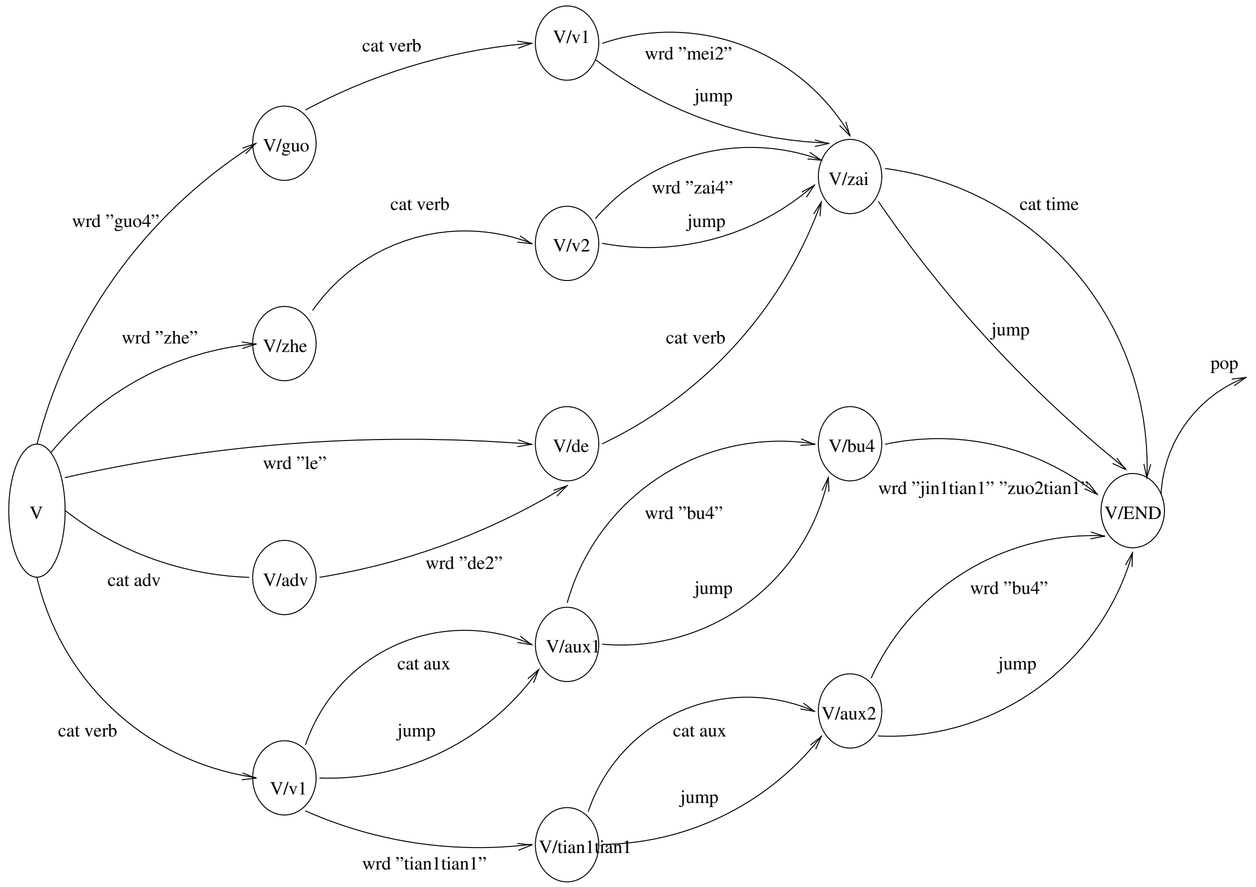

Figure 4.1: Verb Network

There are no tenses in Chinese comparable with those found in European languages. However, what the Chinese aspects express can correspond semantically to English tenses or aspect. Like the English past tense, *guo4* indicates the action took place in the past. *le* indicating completion is roughly equal to the English perfect aspect. *zhe* signals a continuing action. It is similar to the English progressive aspect. When parsing a Chinese aspect, the parser sets the `TENSE` register to the English tense comparable to the Chinese aspect. Later on when the parser builds the SNePS representation, we will have a `TENSE` arc pointing to the value of the `TENSE` register.

The way we handle Chinese aspect is far from being satisfactory though. There have been extensive researches done in the SNePS Research Group on building SNePS language-independent temporal structures for English narrative. It would be better if our parse could utilize them to build similar case frames that capture the meaning of the Chinese aspects. Since this project is to build a prototype machine translation system, the above issue is not our top concern. We leave it to further research.

4.3.3 The noun phrase network

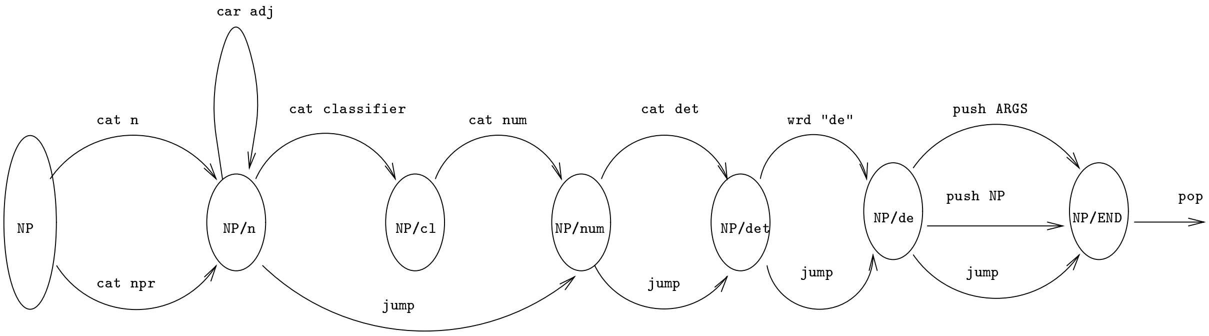

Figure 4.2: The Noun Phrase Network

The noun-phrase network, (NP network), is initiated through a PUSH arc from the ARGS state. The NP network tries to find deixis, numerals, classifiers, and one or more adjectives, as well as the head nouns. Since the word order within the Chinese noun phrase is fixed, the parser returns to top-down mode, when entering the NP network. Scanning the input from right to left, the parser first tries to identify the head noun, which can be a noun or a proper name.

We use the CAT arc to find out the lexical category of the current word. If it is a noun, then the parser builds a member/class case frame to represent it; the parser builds an object/propername case frame for a proper name or a variable node for a question pronoun; if the current word does not belong to any noun category, then the NP subnetwork blocks and is popped back to the ARGS state, where the Verb network and the PP network will be called to see if the current word is a verb or an adposition.

After identifying the head noun, the parser goes on to find the modifiers of the head noun. We build the object/property case frame for each adjective or object/quant case frame for the numeral. We also point the `object` arcs to the base node which represents the intensional entity denoted by the head noun. Notice that the case frames are only built but are not asserted. All nodes thus built are not in CASSIE's belief space until a final valid parse is found, at which time, we use path-based inference to walk down the sentence node tree to assert them. Before we reach the end of the `NP` network, where the parser

Relative clause starts here.

張三	喜歡	李四	買	的	書
Zhang1 san1	xi3 huan1	Li3 si4	mai3	de5	shu1
Zhang1san1	like	Li3si4	buy	DE	book

Zhang1san1 liked the books which Li3si4 bought.

Figure 4.3: An example of *de* as a relative clause marker.

張三	喜歡	李四	的	書
Zhang1 san1	xi3 huan1	Li3 si4	de5	shu1
Zhang1san1	like	Li3si4	DE	book

Zhang1san1 liked Li3si4's books.

Figure 4.4: An example of *de* as a genitive noun phrase marker.

pops back to the `ARGS` state, we check if the current word is a 的 *de*, the relativizer (as in Figure 4.3) or the genitive-noun-phrase marker (as in Figure 4.4). The parser first tests whether the rest of the input sentence can form a relative clause of the head noun just found. If it is not a relativizer, then it should be a genitive-phrase marker; then, the `NP` network recursively calls itself to retrieve the genitive noun. The SNePS nodes representing the relative clause, the genitive, or the head noun and all relevant syntactic information are stored in the `NP` register, which will be sent up to the argument-level network and pushed into the argument stack when the `NP` network pops back to it.

Parsing relative clauses

After the relativizer *de* is identified by the `NP` network, it calls its parent, the argument network, to find the relative clause. Since relative clause possesses every element that constitutes a sentence, the argument network, normally used to parse a sentence, is reused to parse the relative clause. Our grammar is a truly recursive network grammar in which a network calls its subnetworks, which in turn can call its parent network; therefore, deeply embedded relative clauses can be parsed without any problems.

There are two differences in parsing relative clauses and sentences. First, the relative clause is a "sentence" inside the whole sentence. The top sentence spans the whole input string, while a relative clause is part of the input string. That is, we can tell we are at the end of a sentence when no more elements are found in the input string; however, this cannot serve as a clue for a relative clause. We do not know whether we are still parsing a constituent in the relative clause or whether we have already passed the relative clause and have started parsing a constituent in the higher-level clause or sentence. Our right-to-left parsing does not have any problem identifying a relative clause; however, we still have to find a way to tell where this relative clause starts[3] so that we will not mistake a constituent at some higher level for one in this relative clause and so that we can pop back from a relative clause at the right place and return to the higher level.

Second, the syntactic structure of the relative clause is different from the canonical sentence structure. The head noun of a relative clause is at the right end of the clause. Before we can do role assignment on the arguments of a relative clause, we must insert the head noun into the argument stack in the proper place so that syntactic well-formedness can be met.

To decide the starting point of a relative clause, we follow this algorithm: When we push down the

[3] In Chinese, the relative clause comes before the relative clause marker. So we have to find where the relative clause starts.

argument network to find the arguments of the relative clause, we send down the head noun and the *relc* register, whose value is true, to the lower-level argument network. The *relc* register informs the parser that this level of the argument network is for parsing relative clauses and is thus different from the top-level argument network, which is for parsing the whole sentence. Unlike the top-level argument network, which is popped back when the input string is empty, the argument network for relative clauses is popped back when the case assignment for the arguments in the relative clause succeeds. We start to do case assignment as soon as the predicate of the relative clause is found. The predicate provides enough information for us to judge if at this point we have picked up enough arguments for this relative clause. We also do semantic checking on these arguments to know whether this relative clause is grammatical and, more importantly, whether an argument belongs to this relative clause. An argument that does not fit in the predicate's case frame may belong to a clause at some other level. With this method, we are able to handle any long-distance dependency that occurs in the case of sentences deeply embedded within relative clauses whose arguments are separated by arguments at other clause levels. If the `role-assignment` procedure finds there are insufficient arguments or the argument found does not fit the predicate's case frame, then the argument network goes on to take one more. We repeat the `role-assignment` each time when a newly parsed argument is pushed onto the argument stack, until the `role-assignment` procedure succeeds. When it succeeds, we know we have reached the starting point of the relative clause; thus, the SNePS node representing the proposition of the relative clause can be built. Then the lower-level argument network together with the SNePS node is popped back to the `NP` network where this argument network was called.

To decide the syntactic position of the head noun in a relative clause, we use the following rules: If the relative clause starts with a predicate or adposition phrase then the head noun is the subject of the relative clause. In this case, we put the head noun on top of the argument stack for this relative clause. For example,

mai3	*shu1*	*de*	*Zhang1 san1*	
buy	book	DE	Zhang1san1	
Zhang1san1 who bought books.				(4.1)

Zai4	*yi1*	*ge5*	*yin2 hang2*	*zuo4*	*shi4*	*de5*	*Zhang1 san1*	
in	one	CL	bank	do	thing	DE	Zhang1san1	
Zhang1san1 who worked in a bank.								(4.2)

When this relative clause is transformed into the canonical sentence structure, the head noun *Zhang1san1* is the subject of these relative clauses:

Zhang1 san1	*mai3*	*shu1*	
Zhang1san1	buy	book	
Zhang1san1 bought books.			(4.3)

Zhang1 san1	*zai4*	*yi1*	*ge5*	*yin2 hang2*	*zuo4*	*shi4*	
Zhang1san1	in	one	CL	bank	do	thing	
Zhang1san1 worked in a bank.							(4.4)

If the last argument of the relative clause is an adposition phrase, then we insert the head noun right in front of the adposition phrase.

John	*qi2*	*dao4*	*xue2 qiao4*	*de*	*na4*	*tai2*	*dan1 che1*	
John	ride	to	school	DE	Det	CL	bicycle	
the bicycle that John rode to school								(4.5)

The underlying representation of this relative clause is:

John	*qi2*	*na4*	*tai2*	*dan1 che1*	*dao4*	*xue2 qiao4*
John	ride	Det	CL	bicycle	to	school

John rode the bicycle to school. (4.6)

In all other cases, we append the head noun to the bottom of the argument stack. For example,

Zhang1 san1	*mai3*	*de*	*shu1*
Zhang1san1	buy	DE	book

the book that Zhang1san1 bought (4.7)

Zhang1 san1	*gei3*	*Li3 si4*	*de*	*shu1*
Zhang1san1	give	Li3si4	DE	book

the book that Zhang1san1 gave Li3si4 (4.8)

The underlying representations of these relative clauses correspond to these canonical sentence structures:

Zhang1 san1	*mai3*	*shu1*
Zhang1san1	buy	book

Zhang1san1 bought the book. (4.9)

Zhang1 san1	*gei3*	*Li3 si4*	*shu1*
Zhang1san1	give	Li3si4	book

Zhang1san1 gave Li3si4 the book. (4.10)

Note that although we can relativize the direct object, it sounds odd to relativize the indirect object.

*	*Zhang1 san1*	*gei3*	*shu1*	*de*	*Li3 si4*
	Zhang1san1	give	book	DE	Li3si4

Li3si4 whom Zhang1san1 gave book to (4.11)

Since the head noun cannot be the indirect object, it is safe to append the head noun to the end of a ditransitive relative clause.

Parsing the genitive construction

The Chinese character 的 *de* can serve as a genitive marker, a relativizer, or an adjectival marker. When our right-to-left parser encounters the Chinese character 的 *de*, the parser first looks up the dictionary to see whether the next few characters in the input buffer can form an adjective. If not, the parser tests whether this 的 *de* is a relativizer, using the method described in the preceding section. If this try fails i.e., it does not mark a relative clause, we test whether it is a genitive marker. The Chinese genitive construction has this order:

`genitive noun + 的 de + possessed noun`

The noun phrase preceding *de* is the possessor and the noun following *de* is the possessed. We have parsed the possessed noun and 的 *de*. Now we `PUSH` to the noun phrase network to get the genitive noun.

Both the genitive construction of Chinese and that of English are left branching as in *Mary's friend's father's books*. Our right-to-left parser is capable of handling deep left-branching structure. After recognizing the first genitive noun, the parser loops to the same state to look for another genitive marker *de*. If it is found, we `PUSH` to the noun phrase network again to get the next genitive noun. The parser iterates over the same state until no other genitive marker is found.

On collecting all genitive nouns and the possessed noun, the parser builds the SNePS representation for this noun phrase. The semantic relations between the genitive noun and the possessed noun include possession (John's book), kinship relations (John's father), part-whole relations (John's arm), location (John's birthplace) and various other abstract relations (John's birthday, John's success, etc). In our implementation, the default relation for the genitive construction is the possession relation. The parser builds the *kinship-arg1-arg2* case frame for the genitive construction with a kinship semantics, the *part-whole* case frame for that with a part-whole semantics and *possessor-rel-object* for that with a possession semantics.

We use the SNePS Inference Package (SNIP) to deduce which meaning relation holds between the two nouns. In the knowledge base, the kinship terms, e.g., *father, mother, daughter, son, etc* are linked to the kinship property like the following:

```
(M145! (OBJECT (M142 (LEX father)))   (PROPERTY (M144 (LEX kinship))))
(M148! (OBJECT (M146 (LEX mother)))   (PROPERTY (M144 (LEX kinship))))
(M151! (OBJECT (M149 (LEX daughter))) (PROPERTY (M144 (LEX kinship))))
(M154! (OBJECT (M152 (LEX son)))      (PROPERTY (M144 (LEX kinship))))
```

We can deduce from the knowledge base whether a noun is a kinship term. If the possessed noun is a kinship term, then we build the *kinship-arg1-arg2* case frame for the genitive construction as the one in Figure 3.8. We can also deduce whether the genitive noun and the possessed noun are in a part-whole relation. If yes, then we build the *part-whole* case frame for them as the one in Figure 3.9. If all the deductions fail, then we build the *possessor-rel-object* case frame as the one in Figure 3.7.

Parsing conjoined NPs without conjunction

The noun-phrase network only takes care of the syntactic parse. Because there is no conjunction present in case of conjoined noun phrases, at the syntactic level, the parser is unable to determine the boundary of noun phrases. The segmentation of serial nouns[4] is again relegated to the case assignment procedure at the higher-argument network.

When the argument network calls the NP network, the argument stack is sent down to the NP network. If the NP network finds that the element on top of the argument stack is a noun phrase and the current word is a noun, then the NP network concatenates the current word with the noun phrase. The

[4]Please refer to section 2.3.2 for discussions and examples on the serial nouns.

argument stack containing the concatenated NP is then lifted up to the argument network where whether this concatenated NP meets its predicate's semantic case requirements is judged. For the two complements of *give*, the concatenated NP will fails the case-checking. Let us trace the parse of the "Chinese" sentence *John teach Mary Allen linguistics.*

The argument network calls the `NP` network again and again to form concatenated NPs. The elements in the argument stack accumulate in this manner:

(linguistics)

(Allen linguistics)

(Mary Allen linguistics)

((teach) (Mary Allen linguistics))

((John) (teach) (Mary Allen linguistics))

This is a simplified argument stack. In the real implementation, there is syntactic and semantic information, such as lexical category, included with each surface substring. The case-assignment procedure at the end of the argument network performs syntactic and semantic checking on this stack. The predicate 教 *teach* has the lexical entry.

```
("教" ((ctgy . v)                                              (4.12)
       (case_f (Agent animal)
                 (Object knowledge skill)
                 (Experiencer animal))
       (surface_arguments . ("AVEO" "AVO" "OAVE"))
       (sense . "teach")))
```

The lexical entry illustrates that its lexical category `ctgy` is a verb. The `case_frame` feature specifies that the predicate subcategorizes for an Agent role with `animal` selectional restriction, an Object role with `knowledge` or `skill` properties, and an Experiencer role with `animal` property. The `surface_arguments` feature reveals the correlation between the semantic subcategorization and the syntactic structure. The value of the `surface_arguments` feature is a list of possible surface distributions among the predicate and its case frame. The first member of the list `AVEO` indicates one possible ordering: **A**gent + **V**erb + **E**xperiencer + **O**bject. The second member of the list `AVO` indicates another ordering of the roles: **A**gent + **V**erb + **O**bject. The other legitimate ordering is `OAVE`: **O**bject + **A**gent + **V**erb + **E**xperiencer. The `surface_arguments` also implicitly specify the number of argument required by the predicate. For example, `AV` and `EV` means the predicate needs one argument, `AVO` and `OAV` two arguments and `AVEO` and `OAVE` three arguments.

We take the surface arguments one by one to match the stack. The first match fails due to the mismatch between the number of elements, three, in the stack and the number four, in the surface argument "AVEO". The next match succeeds, since both the stack and the surface argument "AVO" have three elements. This match completes the syntactic checking. Next we perform the semantic case checking. The string "AVO" means the predicate should be preceded by an Agent and followed by an Object. The first element of the stack, *John,* an animal, satisfies the verb *teach*'s requirement for Agent, i.e., John is an animal. The third element of the stack, (Mary Allen linguistics), does not satisfy the semantic constraint imposed on the Object role. The case assignment fails, and the parser backtracks to the next arc in the noun-phrase network, where *Mary* will be taken as a single noun phrase. The argument stack will be:

((John) (teach) (Mary) (Allen linguistics))

This time, the surface argument "AVEO" will match the stack syntactically, i.e., John matches A, teach V, Mary E and (Allen linguistics) O. However, the last element of the stack, (Allen linguistics), still does not satisfy the semantic constraint for the Object role. The propername, Allen, is not either a knowledge or skill, although the noun linguistics is a knowledge. The case assignment fails again, and the parser backtracks. The argument stack is now:

((John) (teach) (Mary Allen) (linguistics))

This stack, matching the surface argument “AVEO”, is syntactically well-formed. Its third element, (Mary Allen), meets the semantic restriction of the Experiencer role of *teach* because both Mary and Allen are animals. The last argument, (linguistics), being the knowledge, is a legal slot filler for the Object role. Therefore, the semantic well-formedness of this argument stack is also granted. The SNePS representation is then built and generation follows.

Parsing noun compounds

Two or more nouns may be put together to form a compound. For example,

檸檬	凉糕
ning2 meng2	liang2 gao1
lemon	jelly

lemon jelly (4.13)

The way we handle noun compounds is the same as the way we handle multi-character words. Every word and compound has its own lexical entry. The function `GET-SENSES` looks up the lexical entries in the lexicon for words and noun compounds. Because the interpreter can only see current character from the window of the input string, and we parse sentences from the end, to enable `GET-SENSES` to retrieve the lexical entries of compound, each multi-character lexical entry should have its last character as a separate entry whose lexical category is *multi-end*. The `GET-SENSES` function uses the *multi-end* lexical entries as the key to find the whole compound. For example, to find the current word (or noun compound) in the string 張三吃檸檬凉糕, the GATN interpreter looks for this lexical entry:

```
("糕" ((ctgy . multi-end) (multi-head . ("凉")))                    (4.14)
      ((ctgy . multi-end) (multi-head . ("檸" "檬" "凉"))))
```

With the feature pair, `(multi-head . ("凉"))`, the GATN interpreter identifies the two-character word 凉糕 *jelly*. Through the feature pair, `(multi-head . ("檸" "檬" "凉"))`, the noun compound 檸檬凉糕 *lemon jelly* is retrieved. After the word boundaries are determined, there are separate entries for the words identified. For example, the two-character word 凉糕 *jelly* has the lexical entry:

```
("凉糕" ((ctgy . n)                                                  (4.15)
               (superclass . food)
               (sense . "jelly")))
```

The noun compound 檸檬凉糕, *lemon jelly*, has the entry:

```
("檸檬凉糕" ((ctgy . n)                                              (4.16)
                (superclass . food)
                (sense . "lemon jelly")))
```

The character 糕 is also a word by itself:

```
("糕" ((ctgy . n)                                                    (4.17)
      (superclass . food)
      (sense . "cake")))
```

Therefore, when the interpreter points at the last character 糕 of the input string 張三吃檸檬凉糕, three possible senses are available for disambiguation: two words, 糕 *cake* and 凉糕 *jelly*, and one for the noun compound, 檸檬凉糕 *lemon jelly*.

Part-whole/kinship relations without genitive marker

The Chinese genitive marker DE between two nouns can be used to denote possessive, part-whole, and kinship relations. In colloquial Chinese, sometimes the genitive marker DE is left out. The DE in a possessive phrase is usually preserved, while that in part-whole/kinship relations is more likely to be omitted. Without the genitive, the two nouns can be two independent nouns or one noun phrase. Natural language users can make a judgment based on discourse context and background knowledge. For example:

dian4 shi4	*bing1 xiang1*	
television	refrigerator	
television and refrigerator		(4.18)

dian4 shi4	*de*	*tian1 xian4*	
television	DE	antenna	
television's antenna			(4.19) [5]

dian4 shi4	*tian1 xian4*		
television	antenna		
television's antenna		* 'television and antenna'	(4.20)

Zhang1 san1	*de*	*er2 zi2*	
Zhang1san1	DE	son	
Zhang1san1's son			(4.21)

Zhang1 san1	*er2 zi2*	
Zhang1san1	son	
Zhang1san1's son		(4.22)

Without the genitive marker, there is no syntactic clue to help group the nouns. We have to use the semantic information to find the relation between them. The semantics here is based on our knowledge base. We search the knowledge base; if the two nouns in question are connected through a part-whole path, then they are in a part-whole relation and therefore can be combined into one noun. For example,

dian4 shi4	*tian1 xian4*	(4.23)
television	antenna	

Parsing these two nouns, we use the SNePSUL `find` command to find out whether there exists a part-whole relation between them:

[5] This phrase sounds a bit awkward. *television antenna* sounds better. As for now, our parser translates all part-whole relation with the genitive construction. This needs improving in future implementations.

```
(find (lex- whole- part lex) television)
```

This command will return "antenna" as the result of the search, since we have already built the following node in the knowledge base:

```
(M222! (PART  (M221 (LEX antenna screen)))
       (WHOLE (M219 (LEX television))))
```

Based on this semantic information, we decide that the two words, "television" and "antenna" are actually one noun "television's antenna." The same method is applied to other relations, such as the kinship relation. Since our parsing is based on semantic information, if the knowledge base changes over the translation process, the parse results may change. For example,

Zhang1 san1	*er2 zi2*	(4.24)
Zhang1san1	son	

Our parser will translate this as "Zhang1san1 and son." Then we input the Chinese sentence:

Zhang1 san1	*you3*	*er2 zi2*	(4.25)
Zhang1san1	have	son	

this sentence will get translated as "Zhang1san1 has sons." The SNePS representations for it are built as follows:

```
(M7! (OBJECT B2) (PROPERNAME (M8 (LEX Zhang1san1))))
(M10! (CLASS (M9 (LEX son))) (MEMBER B1))
(M11! (ARG1 B1) (ARG2 B2) (KINSHIP (M9 (LEX son))))
```

"Zhang1san1" now is connected with "son" via the `arg1-arg2-kinship` case frame. Notice that both this representation and our knowledge base are SNePS representations. The newly-created SNePS representation built along with the parsing process is integrated into the knowledge base; therefore, our knowledge base grows with the translation process. The new knowledge base becomes the reasoning

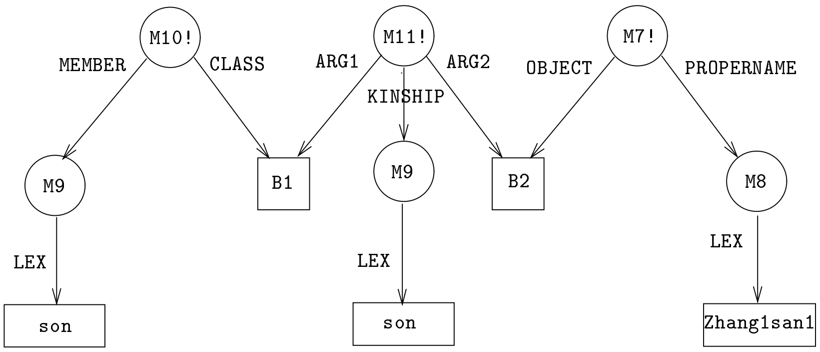

Figure 4.5: Zhang1san1 has sons.

space for future inferences. For identical inference requests, different reasoning space may result in different answers. Based on the responses to the inference, the parser builds the interlingua representation for the input. As a result, our translations are not always fixed. We may get different English outputs for identical Chinese inputs. The translation changes according to the context. This is how the context affects our translation.

Now we ask CASSIE to translate sentence 4.24 again. This time because the knowledge, Zhang1san1 has sons, is in the knowledge base, through the SNePS `find` command, the parser is able to infer that the two nouns, "Zhang1san1" and "son," are in the kinship relation. Thus, besides the conjunctive reading of "Zhang1san1 and son," we get another reading, "Zhang1san1's son."

4.3.4 The adposition phrase network

The adposition phrase in Chinese is usually concerned with time, location, direction, or purpose, and is usually composed of a preposition and/or postposition and a noun phrase.

Here we use the `CAT` arc to recognize the preposition and the postposition. To find the noun phrase, we simply `PUSH` to the noun network. If either the preposition or the postposition is present with the noun phrase, we have found an adposition phrase. The `adpos` register is set to the preposition or the postposition.

Sometimes both the preposition and the postposition are present in a Chinese adposition phrase. The *adpos* register is set to the meaning jointly conveyed by the two Chinese adpositions. For example, the *adpos* register for the Chinese adposition phrase *zai4 ... xia4* is set to the meaning expressed by the English preposition *under*. The preposition and postposition are stored in separate registers, `PREP` and `POST` respectively. The parser looks up a table to find the meaning corresponding to both registers.

4.4 Semantic case checking and role assignment

Case checking is particularly important in parsing Chinese. Because Chinese word order is relatively free, syntactic parsing alone does not impose enough constraints on the grammaticality of a sentence. Our parser traverses the ATN arcs to identify the syntactic arguments and to perform syntactic checking within each argument. Case checking is then implemented to enforce case agreement among these syntactic items. Finally, before the generator takes control of the task, the parser assigns them proper case roles.

The role assignment is a function placed near the end of the parsing process. This function is invoked as the test of a `POP` arc where the input buffer is exhausted, all the arguments have been collected, and the argument network is about to `POP` back to the sentence network. The function checks each argument against the verb's case frame until all the case slots are successfully filled. If the role assignment succeeds i.e., the test returns true, the SNePS node representing this sentence is built and popped up to the sentence network. The parser passes the node to the generation part of the grammar as the input and the generation process follows. Otherwise (the test fails), the `POP` arc is blocked and the parser begins to backtrack for the correct sense to fill out the case frame.

4.4.1 Mapping predicate arguments to case roles

In this section, I will demonstrate the `role-assignment` procedure by tracing the Chinese sentence 甜啼飛到水牛城 *Tweety flew to Buffalo.*

At the end of the argument network, the parser identifies the verb 飛 *fly* and two arguments: 甜啼 *Tweety* and 水牛城 *Buffalo*. They are stored in the argument stack like the following:

((NP 甜啼 *Tweety*) (Verb 飛 *fly*) (PP 到 to (NP 水牛城 *Buffalo*)))

First, the lexical information for the verb is retrieved:

```
("飛" ((ctgy . v)
       (case_frame (Agent can_fly)
                   (Object aircraft)
                   (Locative property_locative))
       (surface_arguments . ("AV" "AVO" "ALV" "AVL"))
       (sense . "fly")))
```

There are four possible syntactic orderings among the verb and its arguments; namely, "Agent Verb", "Agent Verb Object", "Agent Locative Verb", and "Agent Verb Locative". We check each ordering one after one against the argument stack. The first ordering, "Agent Verb", fails the preliminary test because the number of arguments, one, is not equal to that, which is two, in the argument stack. The second ordering "Agent Verb Object" is taken for further consideration because both argument numbers are two.

Now we want to check whether the first item in the argument stack, the propername 甜啼 *Tweety*, can fill the role of being an Agent. The `case_frame` lexical feature of 飛 *fly* specifies that the case filler for the Agent role should be someone who can fly (`can_fly`). The `Role-assignment` function calls SNePS `deduce` to infer whether Tweety can fly. The inference could fail if we had not told SNePS what 甜啼 *Tweety* is. Suppose previously our system has been asked to translate the sentence, 甜啼是一隻

金絲雀 *Tweety is a canary.* The two facts that a canary is a bird and birds can fly are in our knowledge base and have been built in the SNePS semantic network. Through inheritance and path-based inference, SNePS is able to deduce that Tweety can fly. Therefore, Tweety can fill the Agent case slot.

Our semantic restriction for a case role includes diverse properties, features, and relations. One advantage of our taxonomy over a pure ISA one is the simplicity of specification. In an ISA taxonomy, instead of listing `(Agent can_fly)` as we have, one has to list all the classes that meet the constraint, for example, `(Agent bird airplane fighter bee butterfly ...)`. It is also less efficient when the case checking function has to check through all the items to determine whether one argument can fill the role. We simply need to deduce whether one specific item has the ability to fly. The other advantage is the ease of lexicon maintenance. When some categories have to be added to, or removed from, the semantic restrictions for a role, in an ISA taxonomy one has to adjust all the lexical entries affected by the changes. In large lexicons, one cannot be sure whether all the adjustments have been made accordingly. In our case, we just add or remove the properties for these specific categories in the knowledge base. More importantly, because our knowledge base and the translation results are all SNePS semantic networks, the adjustments can be made through natural language. The change of the knowledge base made by the previous translation (natural language) will take effect immediately in the current translation process. In most other systems, the contextual knowledge of the texts previously translated is not utilized. This is one reason that our system claims to have real world knowledge in the sense that it has an ever-changing knowledge acquired from context as opposed to a fixed knowledge predefined in the lexicon found in other systems.

Having successfully identified the Agent role, now we are going to check the Object role. The verb 飛 *fly* needs an `aircraft` sense to fill its object role. Our parser deduces whether 水牛城 *Buffalo* is an aircraft. Because *Buffalo* is not an aircraft, the inference fails. The second ordering "Agent Verb Object" is now ruled out.

The third ordering "Agent Locative Verb" is eliminated from consideration because the verb *fly* occurs second in the argument stack, while the ordering specifies that it should be in the sentence final position.

The Agent role in the last ordering "Agent Verb Locative" is filled by 甜啼 *Tweety* immediately without further verification. When working on the second ordering, we have already called the SNePS `deduce` to infer whether *Tweety* can fill the Agent role. The results of the previous inferences are stored in a hash table,[6] so that the same deduction will never be made twice. Two hash tables, `argument-list` and `infer-list`, are utilized to reduce the queries for knowledge base. The `argument-list` is used to store the constructions of arguments, well-formed or ill-formed, created by the parser. Therefore, in case the parser backtracks, the parser can check the lists. If the syntactic construct has been parsed before, then the result of the role assignment is returned directly; thus, duplication of efforts is avoided. The `infer-list` is used to store a noun's superclasses obtained from querying the knowledge base. Given that each verb usually has more than one case ordering, and each case role could have many candidates for a case filler, there is a good chance that a candidate will be checked over and again. Before making the inference, we look up the hashtable first. If the answer is not in the hash, the inference is made; otherwise, we reuse the answer. This also facilitates the parsing in situations where identical noun phrases tend to recur when we translate a narrative.

The feature-value pair, `(Locative property_locative)`, specifies that the Locative role for 飛 *fly* should have the `locative` property. In our taxonomy, 水牛城 *Buffalo* is a place and all places have

[6] Hash table is a data structure whose elements can be located efficiently using memory access functions called hashing functions.

the `locative` property. By property inheritance and path-based inference, the parser infers that 水牛城 *Buffalo* has the locative property; therefore, it can fill the Locative role. Now all slots have been filled. Each argument is assigned a semantic role as in the following list:

((Agent 甜啼 *Tweety*)
(Locative 水牛城 *Buffalo*))

From this, the SNePS semantic network is then built. The semantic node representing the proposition is stored in a register and is returned to the higher (sentence) level for the generator.

4.4.2 Covert case roles

From the example above, we see that the `role-assignment` function first checks the argument number and the verb position. If both are fine, it then deduces whether the argument string conforms to the semantic restrictions imposed by the case frame. Because making deductions is generally more costly than number checking, the preliminary test facilitates the case checking procedure.

The unspecified Agent in a passive sentence and coreferential case roles are covert case roles. They are specified in the `surface_arguments` lexical feature but do not really exist in the input string. Some care has to be taken before we can go on for the case checking procedure; otherwise, the role assignment will fail because the number of arguments does not match that in the `surface_arguments` lexical feature.

The `surface_arguments`, UOV, "Unspecified_Agent Object Verb", signifies one pattern of Chinese passive sentences. At the start of the `role-assignment` procedure, the parser pushes a dummy string, "#unspecified", for the unspecified Agent role, into the argument stack so that the argument number will agree with that in the `surface_arguments.` The string "#unspecified" always fills the unspecified Agent role.

As an example of covert coreferential role, the letter C in ACVO "Agent Experiencer Verb Object" denotes the covert Experiencer coreferential with the Agent as that in the sentence *John studies linguistics.* We duplicate a copy of the Agent role including its surface string, the SNePS node representing this Agent etc., and put it in the position of the Experiencer role.

4.4.3 Role assignment for relative clauses

Role assignment for relative clauses also needs some pretreatment. Usually the head noun of a Chinese relative clause is not in the canonical position. The head noun is marked by the relativizer 的 DE and always appears at the end of a relative clause. First, the relative clause has to be transformed to the conventional clause through head noun relocation. Then, we can perform case checking on relative clauses the same way as we do it on other sentences.

`subject`	讀	語言學	的	張三
	verb	object	DE	head noun
	study	linguistics	DE	Zhang1san1

Zhang1san1 who studied linguistics (4.26)

`subject`	在	美國	讀	語言學	的	張三
	prep	phrase	verb	object	DE	head noun
	in	the US	study	linguistics	DE	Zhang1san1

Zhang1san1 who studied linguistics in US (4.27)

張三	騎	`object`	到	學校	的	單車
subject	verb		prep	phrase	DE	head noun
Zhang1san1	ride		to	school	DE	bicycle

the bicycle which Zhang1san1 rode to school (4.28)

張三	騎	$\boxed{\texttt{object}}$	的	單車
subject	verb		DE	head noun
Zhang1san1	ride		DE	bicycle

the bicycle which Zhang1san1 rode (4.29)

張三	給	李四	$\boxed{\texttt{IO}}$	的	書
subject	verb	DO		DE	head noun
Zhang1san1	give	Li3si4		DE	book

the book which Zhang1san1 gave Li3si4 (4.30)

⋆張三	給	$\boxed{\texttt{DO}}$	書	的	李四
subject	verb		IO	DE	head noun
Zhang1san1	give		book	DE	Li3si4

Li3si4 to whom Zhang1san1 gave books (4.31)

In Chinese, preverbal adposition phrases follow the subject (as in 4.27) and postverbal adposition phrases follow the object (as in 4.28). If a relative clause begins with an adposition phrase (4.27), it is a preverbal adposition phrase and the missing subject becomes the head noun of the relative clause. We check the relative clause's argument stack. If the relative clause starts with a verb or adposition phrase i.e., the subject is missing (4.26, 4.27), then the head noun is moved to the top of the argument stack to serve as the subject. Otherwise (4.29, 4.30), the head noun should be appended to the end of the argument stack to serve as the object, but, with one exception. Given that postverbal adposition phrases follow the object, if a preposition phrase appears at the end of the argument stack (4.28) then the head noun should be inserted in front of the preposition phrase rather than being appended to it. In addition, because the indirect object of Chinese ditransitive verb is not relativized (4.31), that is, the head noun cannot be the indirect object, we can safely append the head noun to the end of argument stack to serve as the direct object (4.30).

4.4.4 Parsing the serial-verb construction

As summarized by [Li and Thompson, 1981], the serial-verb construction is ”a sentence that contains two or more verb phrases or clauses juxtaposed without any marker indicating what the relationship is between them.” In English, there exist syntactic markers such as conjunctions (*and, but*), markers of subordinate clauses (*because, if*), infinitive markers (*to*), prepositions (*for, on*) to mark the interrelation of verb phrases in the same sentence. The absence of markers in the Chinese serial-verb construction results in ambiguities like those found in the following pseudo English multiple-verb sentences with the omission of syntactic and morphological markers.

	Noun	Verb	Noun	Verb	Noun
1.	John	eat	dinner	read	book
2.	John	believe	Mary	read	book
3.	John	buy	book	give	Mary
4.	John	persuade	Mary	read	book
5.	John	promise	Mary	read	book

Their corresponding English sentences are as follows:

1. John eats dinner and reads books.
2. John believes that Mary read books.
3. John buys books to give Mary.
4. John persuades Mary to read books.
5. John promises Mary to read books.

All the above "sentences" share the same sentence pattern, i.e., `(noun verb noun verb noun)`. However, the relations between the two verbs are all different. In sentence (1), the coordinate conjunction *and* and the tense inflection (-*s*) indicates that the two verb phrases represent two independent events. In sentence (2), the complementizer *that* indicates the second verb is the predicate of the indicative clause of the first verb. In sentence (3), the infinitive *to* indicates the purpose of John's buying books is to give them to Mary. In the last two sentences, the first verb is a control verb and the second verb is "controlled" by the first one i.e., in LFG terms, the first verb subcategorized for an XCOMP, a non-finite clause without an overt subject. In sentence (4), "Mary", the object of the first verb, functionally becomes the subject of the XCOMP; while in sentence (5), "John" becomes the subject of the XCOMP.

From the examples above, we see that the syntactic and morphological markers help clarify the sentences in English. However, we can also observe that native speakers of Chinese are still able to understand corresponding sentences without any markers. The markers serve as clues for machine parsers. For traditional parsers that solely depend on syntactic and morphological information, the markers are the only clues. Native speakers or our parser draw on semantic information to cope with the deficiency of the syntactic and morphological information.

In this project, we classify the Chinese serial verb construction into four types namely indicative clause, control construction, infinitives of purpose and coordinate compound sentences. The semantic relationships between the verbs in the Chinese serial verb construction are very complicated. The classification here is certainly not complete.

- Indicative clause: The first verb subcategorizes for an indicative clause. For example, sentence (2).

- Control construction: The first verb is a control verb, i.e., the first verb stipulates where the subject of the second verb comes from. Sentence (4) and (5) are examples of this construction. In sentence (4), the first verb *quan4* "persuade" is an *object control verb* that identifies its object to the subject of the second verb. Sentence (5) is a subject control sentence. The missing subject of the second verb is the subject of the first verb.

- Coordinate compound sentence: Two or more clauses, each standing for a separate event, are conjoined by the coordinate conjunction *and*. All clauses except for the first one, are without subject and they share their subject with the first clause.

- Infinitives of purpose: The first act is done in order to achieve the second. For example, in sentence (3), the act of *John's buying books* is done for the purpose of *giving them to Mary*.

To determine which type of serial verb construction a sentence belongs to, our parser draws on lexical and case information. First, we make a dictionary lookup to check whether the first verb contains lexical information that decides the type of serial verb construction. Two types of serial verb construction namely indicative clause and control construction are lexically-determined.

In the lexicon, the verb subcategorizing for an indicative clause has a letter `S` in its `surface_arguments` feature. For example, the `surface_arguments` of the verb *believe* is `AVS`, which means the `V`erb follows an `A`gent role and precedes a `S`entential complement.

There is an `Xcomp` lexical feature for control verbs. For example, the verb *persuade* has an `(Xcomp ObjControl)` feature-value pair in its lexical entry. The value `ObjControl` means *persuade*, which subcategorizes for an `Xcomp`lement, is an object control verb. The subject control verb, for example *promise*, has the `(Xcomp SubjControl)` feature-value pair in its lexical entry.

If the type of serial-verb construction cannot be determined from the lexical information, then we use the `role-assignment` procedure I describe below to find out the type of the serial-verb construction involved. Let us take the following Chinese sentence as examples.

Zhang1 san1	*mai3*	*shu1*	*du2*	
Zhang1san1	buy	book	read	
Zhang1san1 bought books to read.				(4.32)

First, the role assignment is performed on the first verb phrase.

Zhang1 san1	*mai3*	*shu1*	(4.33)
Zhang1san1	buy	book	

The role assignment succeeds because the case slots of the verb "mai3" *buy* can be properly filled. The verb *buy* requires an Agent and an Object. "Zhang1san1" is assigned the Agent role; "shu1" *book* the Object role. And then we go on to do role-assignment on the second verb phrase. At first, it is only a verb "du2" *read*; then, it gets its unexpressed subject from that of the first verb phrase. The second verb phrase is now *"Zhang1san1 read"*. The role assignment fails because the required Object role for the verb *read* is missing. The parser then takes the Object of the first verb phrase as that of the second verb phrase. It now becomes:

Zhang1 san1	*du2*	*shu1*	(4.34)
Zhang1san1	read	book	

This time the role assignment succeeds. When the second verb phrase misses two case role values that are shared with the first verb phrase's arguments, this sentence is an infinitives of purpose.

If the second verb does not require an Object role, i.e., there is only a missing subject then the parser is not able to tell whether this sentence is an instance of the coordinate compound or the infinitives of purpose. For now, this kind of sentences will all be parsed into coordinate compound. For example,

Zhang1 san1	*chi1*	*ang1mian2yao4*	*shui4jiao4*	(4.35)
Zhang1san1	eat	sleeping pill	sleep	

Zhang1 san1	*chi1*	*ping2guo3*	*shui4jiao4*	(4.36)
Zhang1san1	eat	apple	sleep	

The two sentences above will be translated into:

Zhang1san1 ate a sleeping pill and slept.

Zhang1san1 ate an apple and slept.

The first sentences, however, could be better translated as:

Zhang1san1 ate a sleeping pill to sleep.

The second verb in a coordinate compound either already has its own Object or does not require an Object role. The only syntactic clue to identify an infinitive of purpose is the missing Object in the second verb phrase. There is no clue to the infinitives of purpose when the second verb phrase does not require an Object role. It is therefore easy to confuse this kind of infinitives of purpose with the compound sentences. The sentence below is another example:

Zhang1 san1	*hui2*	*jia1*	*shui4 jiao4*	(4.37)
Zhang1san1	go	home	sleep	

This sentence can mean “John went home and slept.” or “John went home to sleep.” The decision on one over the other has to be based on the context and real world knowledge. It involves inferences at the narrative level which are beyond our current working level, the sentence. Therefore, we do not take pains to solve this problem here. As of now this kind of sentences will all get translated into English coordinate compound[7]. However, because SNePS has the capability to make inferences on all already built networks which to a great extent constitute the context, this task could become one of the focuses of our further research.

[7]It would be better if both translations are given.

To perform case-checking and role-assignment on infinitives of purpose, we have to obtain the unexpressed grammatical functions first. In the succeeding verb phrase(s), besides the missing subject, some have a missing object or direct object e.g. sentence (3). In the absence of control relations between the arguments of the first verb and the arguments of the succeeding verb(s), the following are some rules for the parser to obtain the unexpressed functions. The missing subject will get its function from the subject of the first verb; the missing object or direct object from the object of the first verb. However, if the first verb is a ditransitive verb, the undergoer will be the controller of the missing object; and the beneficiary the controller of the missing subject.

Zhang1 san1	*gei3*	*Li3 si4*	*qian2*	*yong4.*
Zhang1san1	give	Li3si4	money	use

Zhangshan gave Lisi money to use. (4.38)

Zhang1 san1	*tou1*	*Li3 si4*	*qian2*	*yong4.*
Zhang1san1	stole	Li3si4	money	use

Zhangshan stole Lisi money to use. (4.39)

Let us take the above two sentences for examples. The undergoer in both sentences is the *money*. It will fill the function of the missing object. The beneficiary of the money, i.e. the object of 4.38 and the subject of 4.39, will fill the missing subject.

A right-to-left parser may find difficulty in parsing the serial-verb construction, because some critical parsing information are stored in the first verb phrase, which is located at the left of the sentence. For example, the lexical features of the first verb decide whether the sentence is an instance of sentential complement, a control construction or other types of serial-verb construction. And the arguments of the first verb provide grammatical functions for the missing arguments of the succeeding verb. Parsing sentences from the end, our right-to-left parser is at a disadvantage since it has insufficient information.

Our bottom-up parsing strategy compensates for this disadvantage by pushing all the constituents onto a stack, postponing case-checking and role-assignment until enough information is available i.e., after the first verb phrase is parsed.

4.5 Parsing interrogative sentence

The question mark "?" marks the interrogative sentences. Two types are handled here, namely, yes-no questions and wh-questions. If the question marker, ma1, is found at the end of a question, it is a yes-no question. If successfully parsed, it will be represented by a molecular node, either asserted or unasserted. A wh-question is a question sentence that contains a wh-pronoun e.g. 誰 *who* and 什麼 *what*. A variable node is used to represent the wh-pronoun. Wh-questions, yes-no questions, and declarative sentences are all represented by molecular nodes. However, because a node that governs variable nodes is a pattern node, the molecular nodes for wh-questions are pattern nodes, whereas molecular nodes for yes-no questions and declarative sentences are constant nodes.

Parsing interrogative sentences and parsing declarative sentences require different treatments of the noun objects. When parsing declarative sentences, a new base node is created to represent every noun being parsed. When parsing interrogative sentences, a new base node is created only when the noun is first being introduced because an interrogative is a query to the existing knowledge base as opposed to a declarative which is to build new knowledge. The parser deduces whether in the knowledge base there exist nodes that represent the noun objects. If yes, then the parser reuses the latest node. And for reason of efficiency, on entering the parser, one variable node is created for human wh-pronoun, *who*, and another variable node for the nonhuman wh-pronoun, *what*. The parser reuses these two variable nodes when parsing the wh-pronoun in a wh-question.

In SNePS, the meaning of a node is structurally determined by the arcs emanating from it. The two variable nodes just mentioned are the only two nodes that carry semantic information by themselves. The use of two variable nodes instead of one or many is meant to load each variable node with specific semantic information. For some verbs, both humans and nonhumans are qualified for a case role; for example, the agent role of the verb *move*. Both *Who moves?* and *What move?* are possible. We designate one variable node for human and the other for nonhuman objects so that when generating we have the clue about what is meant in the source language.

Being a query to CASSIE's knowledge, the interrogative itself should not be committed to CASSIE's belief space. Hence, the molecular node representing the interrogative is simply built whereas the node for its declarative counterpart is built and then asserted. This does not mean that all nodes built for the interrogative are unasserted. Given the SNePS Uniqueness Principle that no two nodes represent the same proposition, before SNePS builds a node, it searches for CASSIE's belief space. If there exists a node in the current context that represents the same proposition; then, SNePS retrieves that node rather than building a new one. That is, the retrieved node could have already been asserted if it is the result of the SNIP inference or the parsing of the previous declarative input.

Besides translating the interrogative, the parser uses SNePS Inference Package (SNIP) to `DEDUCE` the answer. For a yes-no question, if the molecular node which is just built to represent the question has been asserted, the answer to the question is an affirmative; "I don't know", otherwise. For the answer to a wh-question, the parser deduces on the pattern node that represents the question. The variable nodes under the pattern node will unify with any nodes that match the case frame structure. SNIP returns the nodes that matches the pattern of case structure. If both human and nonhuman can satisfy the case role constraints in the wh-pronoun position, the nodes returned by SNIP inference have to be further distinguished between theses two classes. Let's look at the following examples.

大華	喜歡	念	語言學.
Da4 hua2	xi3 huan1	nian4	yu3 yan2 xue2
Da4hua2	like	study	Linguistics

Da4hua2 liked to study Linguistics. (4.40)

大華	喜歡	美華.
Da4 hua2	xi3 huan1	Mei3 hua2
Da4hua2	like	Mei3hua2

Da4hua2 liked Mei3hua2. (4.41)

大華	喜歡	冰淇淋.
Da4hua2	xi3 huan1	bing1 qi2 lin2
Da4hua2	like	ice cream

Da4hua2 liked ice cream. (4.42)

The simplified case frames of these three sentences are as follows:

((ACT "like") (EXPERIENCER "Da4hua2") (XCOMP "study Linguistics")) 4.40

((ACT "like") (EXPERIENCER "Da4hua2") (OBJECT "Mei3hua2")) 4.41

((ACT "like") (EXPERIENCER "Da4hua2") (OBJECT "ice cream")) 4.42

After CASSIE translates these three sentences, we ask the question:

大華	喜歡	誰.
Da4 hua2	xi3 huan1	shei2
Da4hua2	like	whom

Whom did Da4hua2 like? (4.43)

The parsing of this question results in the pattern structure below:

((ACT "like") (EXPERIENCER "Da4hua2") (OBJECT HUMAN-VARIABLE))

SNIP then maps this pattern structure against the case frames of the previous inputs. The pattern matches that of examples 4.41 and 4.42. `HUMAN-VARIABLE` can unify with “Mei3hua2” and “ice cream”. Since 誰 “whom” is a human wh-pronoun, SNIP deduces again which candidate is human. “Mei3hua2” is human; therefore, input 4.41 becomes the final answer.

Chapter 5

Generation of English from the Semantic Network

5.1 Overview

The sentence generation is here, in a sense, another form of parsing. When generating English sentences, instead of parsing natural language into SNePS semantic network, the parser "parses" the SNePS semantic network into another natural language. The Generalized Augmented Transition Networks (GATN [Shapiro, 1982]) allows a single grammar to be written for both parsing and generating, which are actually two sub-networks in a GATN grammar. The semantic networks, outputs of parsing the sentence sub-network `S`, are inputs to the generation sub-network `G`.

The differences between parsing natural language and parsing SNePS semantic network are as follows. Parsing a natural language sentence consists in resolving it into syntactic and/or semantic components while parsing semantic networks consists in taking the resolved grammatical components and assemble them into a natural language sentence. There are ambiguities in natural language. Resolving grammatical components involves the resolution of ambiguities, which is nondeterministic. Backtrackings may occur. On the other hand, parsing semantic network is deterministic. For one thing, semantic networks are formal and unambiguous. Furthermore, with the resolved grammatical components in hand, either stored in the registers or built as semantic networks, constructing a sentence from its grammatical components does not incur backtrackings. Because it is better informed and thus spared backtrackings, parsing semantic networks takes a much smaller portion of processing time than parsing natural language does.

For this MT system, lexical information plays a very important role in both parsing and generating. As described before, at the end of sentence parsing, the verb lexical features, `surface_arguments` and `case_frame` are used by the `role-assignment` procedure to check whether the sentence is syntactically and semantically grammatical. When generating a sentence, the parser first consults the lexicon then puts the verb and its arguments in the order specified by the `surface_arguments` feature. Unlike the typical ATN grammars whose transition diagrams depict all possible sentence structures they can handle, in this grammar, there are no presumed structures at the clause or sentence level. The sub-networks in our grammar only describe how phrasal constituents, e.g. NP, PP, VP, should look like. The overall orderings among them are prescribed in the lexicon. The state-arc design is used for processing phrases. At the clause or sentence level, LISP procedures based on lexical information are used.

There are several sub-networks for sentence parsing, e.g. NP network, Verb Network, and PP network ...etc. However, there is only one sub-network `G` for sentence generation. The `G` network iterates over itself until the whole semantic network representing the sentence to be generated is traversed. As we can see, the sentence generation relies even more on LISP procedures than on the ATN states and arcs. The reasons the LISP procedures are used in preference to the ATN states and arcs is that parsing semantic networks is much more informed and deterministic than parsing natural languages. The ATN state-arc design, equipped with recursive and backtracking mechanisms, is good for resolving grammatical components and ambiguities; however, the generator has no need of resolving anything because all the syntactic and semantic information of the sentence to generate are in hand. Furthermore, the ATN built-in backtracking mechanism so useful for parsing natural languages is of no help in parsing formal and unambiguous languages like semantic networks. Not being much good at parsing semantic networks, the state-arc design puts the overhead of interpreting the ATN arcs and states on the interpreter. The LISP procedures spare it the overhead.

Two arcs emanate from the `G` sub-network. Each arc process one of the two types of SNePS nodes: atomic nodes and molecular nodes. All SNePS nodes can be categorized into these two types. To be more precise, the molecular nodes here refer to one specific kind of molecular nodes, the structured proposition nodes; and, the atomic nodes refer to all three kinds of atomic nodes, i.e. sensory nodes, base nodes, and variable nodes. The structured proposition nodes include both the constant nodes and pattern nodes. Constant nodes represent the propositions of the sentences or clauses. Pattern nodes represent the propositions of the interrogative sentences. Base nodes represent individual objects. Variable nodes represent arbitrary individuals. Both base nodes and variable nodes represent the head noun of a noun phrase or prepositional phrase. The sensory nodes here represent the verb strings. The molecular nodes in question correspond to the sentence or clause level whereas the atomic nodes correspond to the phrase or argument level.[1] To put it another way, the arc processing the molecular nodes is to handle the whole sentence or clause structure; and, the arc processing the atomic nodes is to handle the phrase or arguments.

[1]In the input buffer, different node types represents different syntactic constituents. Molecular node represents a sentence or a clause. Base node represents a phrase. Sensory node represents a verb. Therefore, the node type tells the parser which syntactic constituent it is dealing with.

Although the `G` sub-network simply iterates over itself again and again, it still has the recursive power to handle deeply embedded syntactic structure. One of the characteristics of the ATN grammar is the ability to call any sub-networks within any sub-networks recursively to handle embedded syntactic structures. Our generator does not utilize this recursive characteristic of the ATN grammar; instead, the structural properties of SNePS semantic networks provide it with the capacity to generate embedded structures recursively. SNePS is a propositional semantic network in which propositions about another propositions are allowed to be represented in a nested fashion. Therefore, the structure of SNePS is, in nature, recursive. Propositions are related to each other through the links of arcs and path-based inference.

The `G` sub-network accepts a molecular node as the main proposition of the input statement. The arc handling molecular nodes gets activated at that time. It finds all the nodes along the arcs emanating from the molecular node. They include several atomic nodes (for verb, NP or PP etc) and, in some cases, the other molecular nodes (for SCOMP, XCOMP, infinitives etc). These nodes are to be arranged in syntactic order. The arc labels provide the semantic case information; the GATN registers provide those information not represented on the semantic networks such as the sentence mood and the determination of a noun phrase etc; and, the English lexicon provides syntactic information. The `surface_arguments` lexical feature in the verb entry specifies what the syntactic order of these cases should be. The atomic nodes, each representing one case role, take the place of the main molecular node in the input buffer. And then, the parser loops back to the `G` sub-network.

From now on, the parser consumes one node at a time from the input buffer and processes it. This time an atomic node is retrieved. The other arc handling atomic nodes gets to process it. Atomic nodes are to be synthesized into English strings. If the atomic node is a base node representing the head noun of a noun phrase, then the parser will also find out whether there is a relative clause modifying this noun through path-base inference. If a molecular node is found representing the noun's relative clause, then this molecular node is pushed onto the front of the input buffer. Control returns to the `G` sub-network. Again, the molecular node is consumed. The arc handling molecular nodes expands it into several nodes in the way we describe in the previous paragraph. The newly expanded atomic nodes are pushed onto the input buffer. If another level of relative clause is found while processing a relative clause, another molecular node is pushed onto the front of the input buffer. After the nodes for the relative clause are synthesized into English strings, the generator continues processing the nodes for the main sentence. From molecular node to atomic nodes, then from atomic node to molecular node, and then from molecular node to atomic nodes, in this way over and over, our generator captures the recursive nature of the SNePS structure. Therefore, with one `G` sub-network and two arcs emanating from its start state, the generator can produce complex sentence structures and deeply embedded clauses. The LISP procedures in the action part of each arc do a big chunk of the work needed for node expanding, ordering, and sentence synthesis.

5.2 Generation of simple sentences

At the beginning of the generation, the lexicon is switched from Chinese to English. In the input buffer, there may be one or several molecular nodes representing the proposition(s) of the input Chinese sentence. Multiple molecular nodes represent a sentence with conjoined clauses. We first describe the generating process with one single molecular node in the input buffer. We will discuss the generation of conjoined clauses later. The `G` Network takes one node out of the input buffer and starts to process it.

Following the arcs emanating from the node, its immediate subordinate atomic nodes are retrieved and each of them is paired with the label of the arc coming to them. Because the arc labels illustrate the predicate-argument (semantic) structure, the arc-node pairs give a semantic mapping between nodes and the predicate-argument structure. This semantic structure will be further mapped onto the syntactic structure.

The parser first searches the arc-node pairs for the predicate. Using the key `ACT`, the arc label for the verbs, the atomic node for the predicate is retrieved. The syntactic structure among the predicate and its arguments is designated in the `surface_arguments` feature of the predicate's lexical entry. The parser looks up the lexicon for this feature then arranges the nodes in the order specified in it. After these nodes are pushed onto the input buffer, the parser loops `TO G`.

The parser again takes one node at a time from the input buffer and starts to process it. This time, it is an atomic node that represents either the verb predicate or a noun argument. The parser finds the English expressions for the nodes and synthesizes them into surface strings. The synthesis module assigns inflections for case, number, and definiteness of nominals, and tense, aspect, mood, and voice of verbs based on the information derived from the SNePS node structure, registers, together with the syntactic and morphological requirements of English. When all the atomic nodes in the input buffer are exhausted, the generation process ends and the English output representing the proposition of the input nodes are displayed on the terminal. Since the input nodes represent the proposition of the Chinese input, this English output is the translation of the correspondent Chinese input.

Let's take an example for illustration:

那	本	書	張三	讀	了.
Nei4	ben3	shu1	Zhang1 san1	du2	le5.
that	CL	book	Zhang1san1	read	LE

Zhang1san1 read that book. (5.1)

The following SNePS semantic networks are built for the above sentence. Node `M265!`, representing the main proposition of the above Chinese input, is stored in the input buffer for English generation. `M259!`

and M261! are not in the input buffer but is connected with M265! through arc and node as shown in Figure 5.1.

```
(M259! (CLASS (M165 (LEX book))) (MEMBER B1))
(M261! (OBJECT B2) (PROPERNAME (M207 (LEX Zhang1san1))))
(M265! (ACT (M264 (LEX read) (TENSE PAST))) (AGENT B2) (OBJECT B1))
```

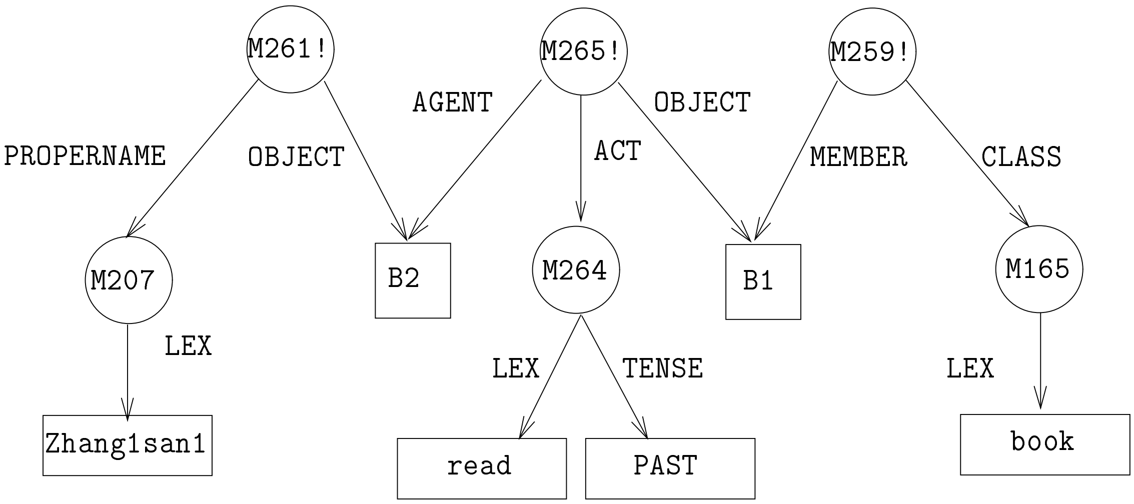

Figure 5.1: *Zhang1san1 read that book.*

The generation starts in the G sub-network with the input buffer being (M265). The generator takes M265 out of the input buffer. The arc handling the molecular node is followed since M265 is a molecular node. Each immediate subordinate atomic node of M265 is paired with the label of the arc coming to it. The list of arc-node pairs looks like this: ((ACT read) (AGENT B2) (OBJECT B1)).

Then the generator looks up the features in the lexical entry for the verb *read*. The value of its surface_arguments feature is ("AVO"), which means, for the English verb *read*, there is exactly one possible surface realization of the <Agent, Object> predicate-argument structure. That is, the verb read follows the agent role B2 and precedes the object role B1. According to this specification, the nodes are arranged into the order of B2, read, and then B1. These three nodes are pushed onto the input buffer. The generator loops TO G with the input buffer being (B2 read B1).

The node `B2` is popped out from the front of the input buffer. After handling `B2`, the generator loops back `TO` state `G`, consuming another node and process it until the input buffer is empty. `B2` and the subsequent nodes `read` and `B1` are all taken care of by the same arc from state `G` that deals with the atomic nodes. From `B2`, following the path, `(OBJECT- PROPERNAME LEX)` [2], the parser `FIND`s the sensory node, `Zhang1san1`. It is `WORDIZE`d[3] into the propername `"Zhang1san1"` and is added to the register `SENTENCE` that collects the surface string being built.

The generator loops back `TO G` taking the second node `read`, a sensory node for a verb. To determine the tense of the verb, the generator follows the path, `(LEX- TENSE)`. A `PAST` node is found at the end of the path. Therefore, the node `read` is `VERBIZE`d[4] into the third person singular past tense, which is *read*. This is added to `SENTENCE`, forming `("Zhang1san1" "read")`. The generator loops `TO` state `G`, taking the last node, `B1`, and emptying the input buffer. From `B1`, following the path, `(MEMBER- CLASS LEX)`, the sensory node `book` is found. To synthesize a noun object, the generator has to determine its number first. Because the register `DET` contains `that`, a singular determiner, the noun should be singular. The generator adds the determiner "that" and the noun "book" to `SENTENCE`, which is now `("Zhang1san1" "read" "that" "book")`. Seeing an empty input buffer, the `POP` arc at state *G* adds a period to the end of `SENTENCE` and `POP`s the contents of `SENTENCE`, which is finally printed by the system, and the interaction is complete.

---

[2] `OBJECT-` is the converse of `OBJECT`. In SNePS, arc labels ending in the character '-' are reserved for this reverse arc or converse relation labeling.

[3] WORDIZE is a LISP function that does the morphological synthesis for nouns. It returns the singular or plural form of a string, symbol, or node. The singular form will use the ROOT form or lexeme itself. Its pluralization operates on the ROOT form according to an extensive set of rules built into it.

[4] VERBIZE is a LISP function that does morphological synthesis for a verb. Its parameters are the person, number, tense, aspect, and the verb to be used.

5.3 Generation of coordinate compound sentences

A coordinate compound sentence comprises several independent clauses that are conjoined by coordinating conjunctions e.g. *and*, for example, *John ate an apple, read a book, and slept.* The first clause shares its subject with the other clauses. Usually a sentence is represented by one SNePS node; however, a coordinate compound sentence is represented by as many nodes as the independent clauses it has i.e. each individual clause is represented by a different node. Therefore, the generation of coordinate compound sentences is generating from multiple nodes instead of from one single node as we usually do for other types of sentences.

Let's take an example for illustration:

張三	吃	蘋果	讀	書	睡覺.
Zhang1 san1	chi1	ping2 guo3	du2	shu1	shui4 jiao4.
Zhang1san1	eat	apple	read	book	sleep

Zhang1san1 ate an apple, read a book and slept. (5.2)

The following SNePS semantic networks are built for the above sentence. Nodes, `M420!   M421!   M423!`, representing the main proposition of the above Chinese input, are stored in the input buffer for English generation. The `G` sub-network starts with an input buffer of `(M420!   M421!   M423!)`. These three

```
(M261! (OBJECT B2) (PROPERNAME (M207 (LEX Zhang1san1))))
(M418! (CLASS (M257 (LEX apple))) (MEMBER B51))
(M420! (ACT (M419 (LEX eat))) (AGENT B2) (OBJECT B51))
(M417! (CLASS (M207 (LEX book))) (MEMBER B50))
(M421! (ACT (M272 (LEX read))) (AGENT B2) (OBJECT B50))
(M423! (ACT (M422 (LEX sleep))) (AGENT B2))
```

Figure 5.2: *Zhang1san1 ate an apple, read a book, and slept.*

nodes represent the three independent clauses in the above sample sentence. The arc that deals with molecular nodes expands these three molecular nodes into their immediate atomic nodes. `M420!` is first expanded to `(B2 eat B51)` in the order of `(AGENT VERB OBJECT)` according to the `surface_arguments` feature of the verb *eat*. In the same manner, the second molecular node, `M421!`, is expanded to `(B2 read B50)` in `(AGENT VERB OBJECT)` order; and then, the last node, `M423!`, is expanded to `(B2 sleep)` in `(AGENT VERB)` order. Before all these atomic nodes get conjoined into one sequence, some form of conjunction will replace the first node `B2` in the second and the last clauses because it is identical in all three clauses and should be shared with the first clause. A comma replaces the node `B2` in the second clause; and, a comma followed by the coordinate conjunction, *and*, replaces `B2` in the last clause. At last, the conjoined atomic nodes are stored in the input buffer, which contains `(B2 eat B51 "," read B50 ", and" sleep)`. The generator loops `TO` the state `G`.

Now the arc dealing with atomic nodes takes controls. In the same way we described in the previous example, English strings are found and synthesized to express these atomic nodes. Through the path `(OBJECT- PROPERNAME LEX)`, the propername `"Zhang1san1"` is found to express `B2`. Through the path `(MEMBER- CLASS LEX)`, the nouns, *apple* and *book*, are found to express `B51` and `B 50` respectively. Since Chinese does not have number and definiteness markers, these two English nouns default to singular and indefinite. And since there is no tense marker in Chinese either, the three English verbs *eat, read*, and *sleep*, default to the past tense. The two strings "," and ", *and*" are spliced onto the noun in front of it. Thus, the sentence *Zhang1san1 ate an apple, read a book, and slept.* is generated.

5.4 Generation of complementized clauses

Three types of complementized clause are dealt with, namely, indicative clause, control constructions, and purpose clauses. Coordinate compound and indicative clauses comprise finite clauses whereas control construction and purpose clauses comprise infinitive clauses. They are all represented as serial-verb constructions in Chinese.

An indicative clause is an embedded finite clause with all its subject and object surfacing. It is a sentence per se except that it is embedded under another sentence. Control constructions and purpose clauses are infinitive clauses without overt subject (and object in some case); the missing subject is shared with its parent clause. The generations of an indicative clause and a sentence are similar; however, generating infinitive clauses is more complicated. When generating infinitive clauses, care has to be taken to determine whether its subject (and object) should be shared with the parent clause or they should surface. If shared, then it has to be further decided from where the embedded infinitive clause acquire its unexpressed grammatical functions; from the subject, from the object, or from the indirect object of the parent clause.

5.4.1 Generation of indicative clauses

There are a group of verbs that require an indicative clause, for example, *think, believe, tell* etc. These verbs take a subject, sometimes an object and an indicative clause. The generation of indicative clauses follows the regular processes for the generation of a sentence. No particular care is taken. Here is an example of operation:

張三	相信	李四	親	了	美華.
Zhang1 san1	xiang1 xin4	Li3 si4	qin1	le5	Mei3 hua2.
Zhang1san1	believe	Li3si4	kiss	LE	Mei3hua2

Zhang1san1 believed that Li3si4 kissed Mei3hua2. (5.3)

The semantic network representation for this sentence is built as follows:

```
(M261! (OBJECT B2) (PROPERNAME (M207 (LEX Zhang1san1))))
```

```
(M262! (OBJECT B3) (PROPERNAME (M208 (LEX Li3si4))))
(M401! (OBJECT B46) (PROPERNAME (M400 (LEX Mei3hua2))))
(M402 (ACT (M354 (LEX kiss))) (AGENT B3) (OBJECT B46))
(M405! (ACT (M404 (LEX believe))) (AGENT B2) (COMP (M401)))
```

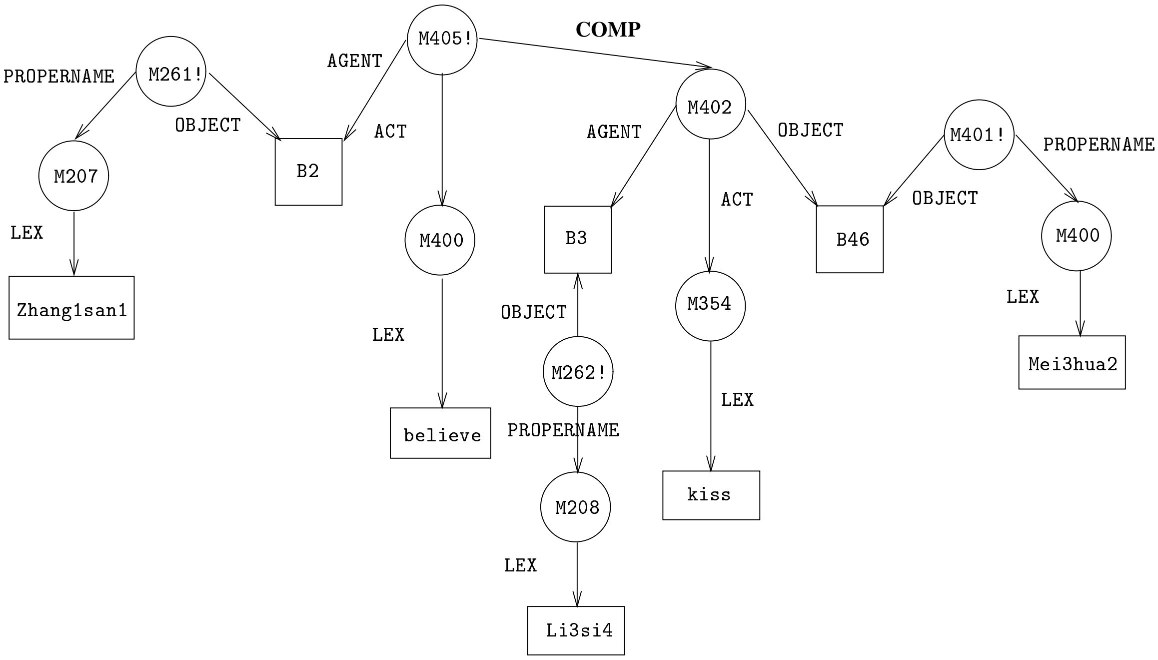

Figure 5.3: *Zhang1san1 believed that Li3si4 kissed Mei3hua2.*

The node `M405` representing the proposition of the whole sentence is first expanded to `(B2 believe M401)`. Then the first node, `B2`, is surfaced as the propername `"Zhang1san1"`. The second node `believe` is synthesized into its past tense `"believed"`. The third node `M401`, being a molecular node, is expanded to `(B3 kiss B46)`. Now the input buffer contains `(B3 kiss B46)` with the register `SENTENCE` being `"Zhang1san1" "believed"`.

The first node, `B3`, is surfaced as the propername `"Li3si4"`. The second node, the verb `kiss`, is synthesized into its past tense `"kissed"`. The last node, `B46`, is surfaced as the propername `"Mei3hua2"`. Since the input buffer is empty, the generation is done. The contents of `SENTENCE`, `("Zhang1san1" "believed" "Li3si4" "kissed" "Mei3hua2")`, is finally printed to the screen.

5.4.2 Generation of the control construction

In control constructions, the main verb, e.g. *persuaded*, subcategorizes for an infinitive clause i.e. XCOMP. The main verb controls from where, either the subject or the object, the XCOMP acquires its missing subject. Therefore, when generating the embedded clause XCOMP, the subject is omitted and the verb takes the infinitive form. Let us look at an example of such operation:

張三	勸	李四	念	語言學.
Zhang1 san1	quan4	Li3 si4	nian4	yu3 yan2 xue2.
Zhang1san1	persuade	Li3si4	study	Linguistics

Zhang1san1 persuaded Li3si4 to study Linguistics. (5.4)

```
(M261! (OBJECT B2) (PROPERNAME (M207 (LEX Zhang1san1))))
(M262! (OBJECT B3) (PROPERNAME (M208 (LEX Li3si4))))
(M408! (MEMBER B53) (CLASS (M407 (LEX Linguistics))))

(M427! (ACT (M411 (LEX persuade))) (AGENT B2) (OBJECT B3)
       (XCOMP (M425 (ACT (M339 (LEX study)))
                    (AGENT B3)
                    (EXPERIENCER B3)
                    (OBJECT B53))))
```

The node M427!, representing the input sentence, is first expanded to its immediate subordinate atomic nodes, (B2 persuade B3). Then following the arc XCOMP, the subordinate molecular node, M425, is retrieved and added to the previous list, (B2 persuade B3). These nodes replace M427! in the input buffer, which becomes (B2 persuade B3 M425).

The first node, B2, surfaces as the propername "Zhang1san1". The verb, persuade, is synthesized into the past tense, "persuaded". B3 surfaces as the propername "Li3si4". The input buffer now has only (M425) left with SENTENCE being ("Zhang1san1" "persuaded" "Li3si4").

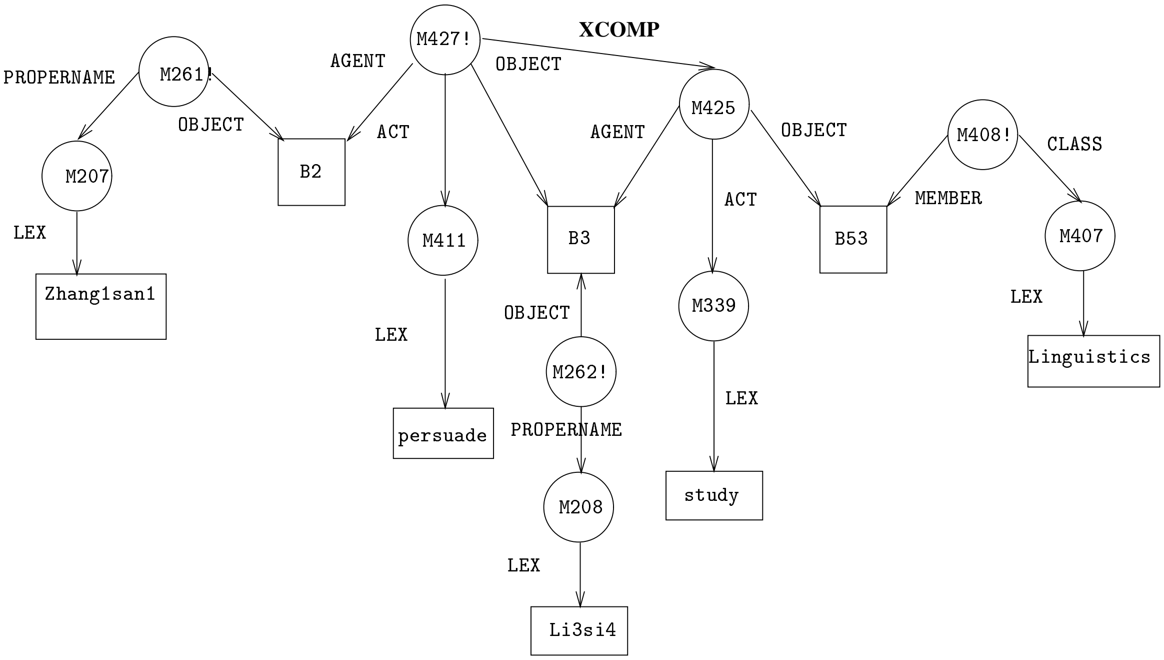

Figure 5.4: *Zhang1san1 persuaded Li3si4 to study Linguistics.*

Being a molecular node, `M425` is expanded to `(study B3 B3 B53)`. They are to be arranged according to the order, `ACVO`, which is the `surface_argument` lexical feature of the verb `study`. `ACVO` means the `AGENT` should go first, followed by the `EXPERIENCER` Coreferential with the agent, then `VERB`, and then `OBJECT`. After sorting, the order of these atomic nodes is: `B3 B3 study B53`, which are `AGENT, EXPERIENCER, VERB, and OBJECT` respectively. Because coreferential roles never surface together, the second `B3`, an `EXPERIENCER` coreferential with the `AGENT`, is dropped. Note that before a molecular node is expanded, it is examined whether it represents the proposition of the main sentence or that of an embedded clause. The generator `FIND`s that `M425` is an `XCOMP`lement governed by the main proposition `M427` through the path `XCOMP`.

The generator loop `TO` state `G` with the input buffer being `(B3 study B53)`. Knowing that the nodes in the input buffer are derived from `M425`, an `XCOMP`lement, the generator drops the `XCOMP`lement's subject node, B3, and adds the string "to" to SENTENCE. The second node, study, is synthesized to its infinitive form "study" because the last string in SENTENCE is the "to" indicating a to-infinitive is to be formed. The last node, B53, is generated to the noun, "Linguistics". "study" and "Linguistics" are appended to the register SENTENCE making it ("Zhang1san1" "persuaded" "Li3si4" "to" "study" "Linguistics").

5.4.3 Generation of the purpose clause

The generation of the purpose clause is similar to that of the control construction except that, in addition to the constantly unexpressed subject, the object and/or the indirect object of the purpose clause may be omitted as well. The generator FINDs whether a given grammatical function of the embedded purpose clause is shared with its parent clause. If shared, the grammatical function is omitted from the purpose clause. The example 5.5 below illustrates the generation of the purpose clause.

The main proposition M481! is first expanded to (buy B3 B3 B62). The surface_argument lexical feature of the verb buy, ADVO, dictates that the AGENT goes first, followed by D (the BENEFACTIVE coreferential with the agent), then the VERB, and then the OBJECT. The sorted node list, (B3 B3 buy B62), are AGENT, BENEFACTIVE, VERB, and OBJECT respectively. The second B3, being coreferential to the first node, is dropped. M479 is also found subordinate to the root node M481! through the arc INTENT. That is added to the input buffer, forming (B3 buy B62 M479). B3, buy, and B62 surface. The resulting strings, ("Li3si4" "bought" "a book"), are put in SENTENCE.

M479 is expanded to (give B3 B2 B62). It is sorted into (B3 give B2 B62), which are AGENT, VERB, BENEFACTIVE, and OBJECT respectively. Then following the arc INTENT, another purpose clause, M477, is found one layer down. It is appended to the input buffer, making it (B3 give B2 B62 M477).

李四	買	書	給	張三	讀
Li3 si4	mai3	shu1	gei3	Zhang1 san1	du2
Li3si4	buy	book	give	Zhang1san1	read

Li3si4 bought a book to give Zhang1san1 to read. (5.5)

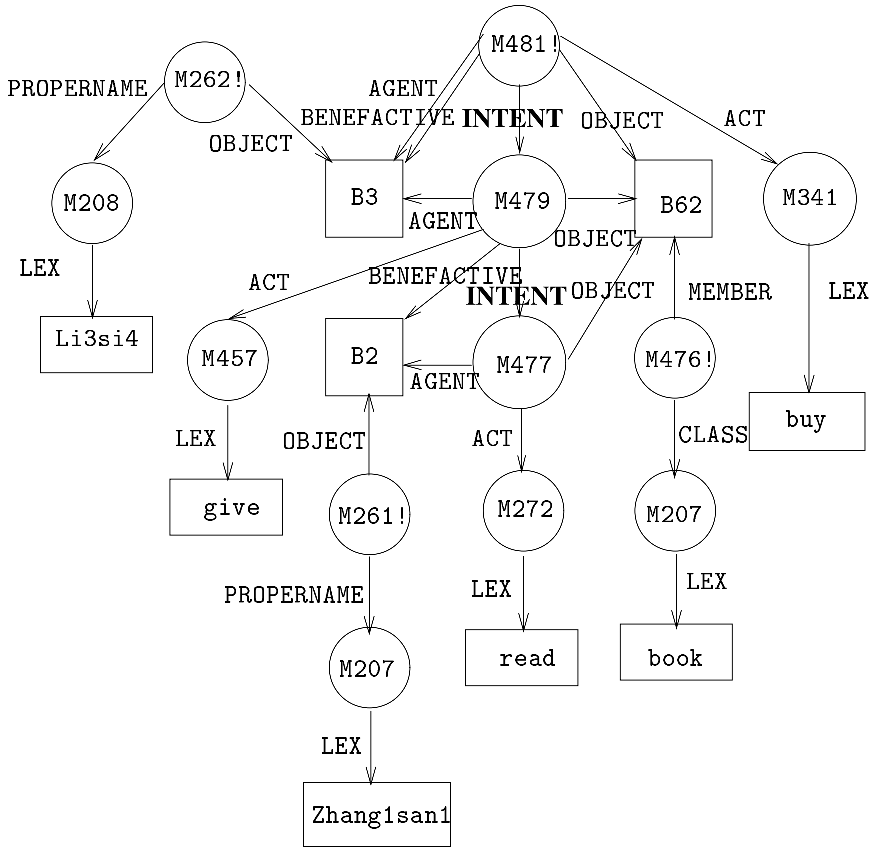

Figure 5.5: *Li3si4 bought a book to give Zhang1san1 to read.*

The generator `FIND`s an `INTENT` arc pointing to `M479`, that is, `M479` represents the proposition of a purpose clause; therefore, its case roles are examined to determine whether an individual role should be shared with the parent clause or should surface. An individual role is found to be shared with the parent clause if, besides an arc from it to the parent clause, there exists a path, `(INTENT- (OR AGENT OBJECT EXPERIENCER BENEFACTIVE))`, that links the atomic node representing the case role to the molecular node representing the purpose clause. Let's take `B62` for example. Besides the `OBJECT` arc that links `M479` with `B62`, there is a path `(INTENT- OBJECT)` that goes from `M479` to `B62`. Therefore, `B62` is shared with the parent node and should not surface in the purpose clause. `B3` is also shared; so, it does not surface either. The verb `give` is synthesized to the to-infinitive form, `"to give"`. `B2` gets surfaced to the propername `"Zhang1san1"` since there is no path going from `M479` to it in addition to the arc `BENEFACTIVE`, meaning it is not shared with the parent clause. The `SENTENCE` is now `("Li3si4" "bought" "a book" "to give" "Zhang1san1")`.

The last node `M477` is expanded to `(B2 read B62)`. Because `M477` is a purpose clause again, in the same manner described above, `B2` is found shared with the parent purpose clause, `M479`; and, `B62` with the main clause, `M481!`. Therefore, neither `B2` or `B62` surface. Only the verb `read` is synthesized to `("to read")`, which is added to `SENTENCE`, forming `("Li3si4" "bought" "a book" "to give" "Zhang1san1" "to read")`.

5.4.4 Verb patterns in the complementized clauses

Not all verbs in the complementized clauses carry the to-infinitive form. Some are in the gerund form; some are preceded by a preposition. For example:

(5.6) *John enjoys playing badminton.*

(5.7) *Mary insists on paying the bill.*

Some complementized clauses are in the form of a that-clause whose subject surfaces rather than being left unexpressed. For example:

(5.8) *Mark assures that he will pay the bill.*

(5.9) * *Mark assures to pay the bills.*

Which pattern a particular verb licenses is specified in the verb's lexical entry. The generator forms the sentences according to the pattern the verb licenses. For example, the value of the `Vpattern` lexical feature of the verb *suggest* is `that-clause`. The `Vpattern` for *insist*, *consider*, and *enjoy* is `gerund`. For those verbs preceded by the preposition, in addition to `Vpattern`, there is a lexical feature `prep`. For example, the verb *insist* has a `prep` lexical feature whose value is `on`.

5.5 Generation of the relative clause

Whenever a noun phrase is generated, the generator checks to see whether there is a relative clause modifying it. If yes, then the node representing the proposition of the relative clause is pushed onto the input buffer and then the relative clause is generated. The path through which the generator `FIND`s the relative clause is as follows:

```
(and (or (relc-a- main (kstar (or xcomp intent)) Agent)
         (relc-o- main (kstar (or xcomp intent)) Object)
         (relc-e- main (kstar (or xcomp intent)) Experiencer)
         (relc-b- main (kstar (or xcomp intent)) Benefactive)
         (relc-l- main (kstar (or xcomp intent)) Locative))
     (or Agent Object Benefactive Experiencer Locative))
```

The node representing the noun phrase is at the end of the path. If a molecular node is found at the beginning of the path, that node represents the relative clause of the noun phrase.

The head noun of the relative clause should be replaced by the relative pronoun and then moved to the front of the clause. However, in this implementation, the relative pronoun is generated right after generating the head noun; then, the head noun in the relative clause is simply omitted. The generator `INFER`s whether the noun phrase is a human being. If the noun phrase is human then the relative pronoun, "who", is used; "which" otherwise. The generator is able to FIND the node representing the head noun of the relative clause through the following path:

```
(and
 (or agent- object- experiencer- benefactive- locative-)
 (or (agent- (kstar (or xcomp- intent-)) main- relc-a)
     (object- (kstar (or xcomp- intent-)) main- relc-o)
     (experiencer- (kstar (or xcomp- intent-)) main- relc-e)
     (benefactive- (kstar (or xcomp- intent-)) main- relc-b)
     (locative- (kstar (or xcomp- intent-)) main- relc-l)))
```

The node representing the relative clause is at the end of the path. The node representing the head noun, if found, is at the beginning of the path. Let us look at an example of relative clause generation:

教	英文	的	張三	喜歡	美華.
jiao1	ying1 wen3	de5	Zhang1 san1	xi3 huan1	Mei3 hua2.
teach	English	DE	Zhang1san1	like	Mei3hua2

Zhang1san1 who taught English liked Mei3hua2. (5.10)

```
(M261! (OBJECT B2) (PROPERNAME (M207 (LEX Zhang1san1))))

(M308! (MEMBER B30) (CLASS (M307 (LEX English))))

(M375! (ACT (M229 (LEX teach))) (AGENT B2) (OBJECT B30))

(M366! (OBJECT B36) (PROPERNAME (M191 (LEX Mei3hua2))))

(M377 (MAIN (M376! (ACT (M354 (LEX like))) (EXPERIENCER B2) (OBJECT B36)))
 (RELC-E (M375!)))
```

The main proposition, representing the whole sentence, M376! is first expanded to (B2 like B36). The first node, B2, represents the propername "Zhang1san1". When generating it, the generator FINDs

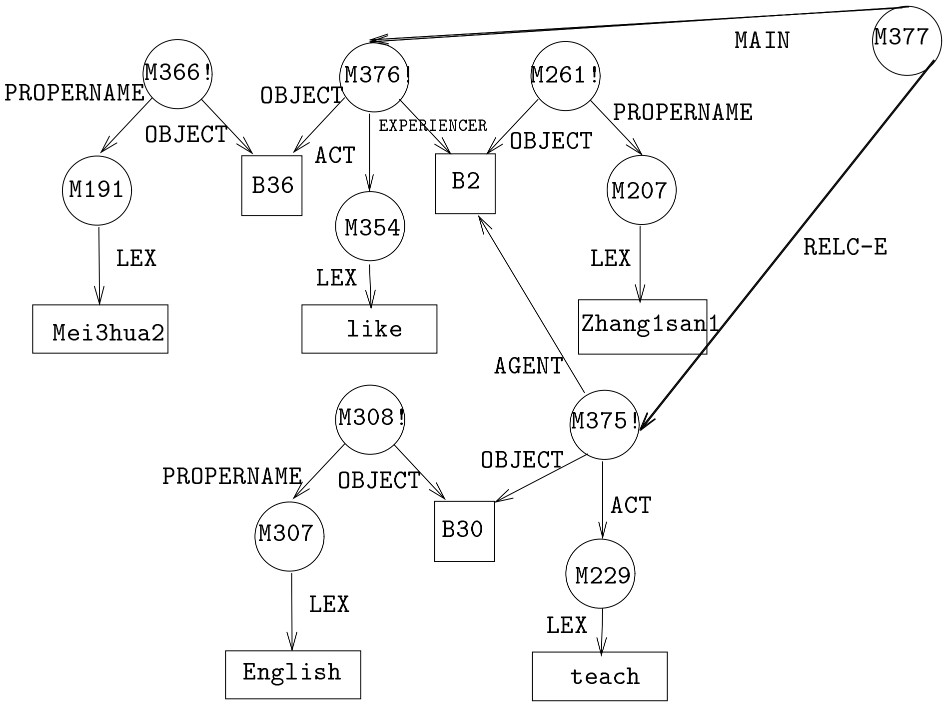

Figure 5.6: *Zhang1san1 who taught English liked Mei3hua2.*

the node `M375!` that represents the proposition of the relative clause modifying `"Zhang1san1"`. Therefore, the relative pronoun `"who"` is generated and appended to the register `SENTENCE`, which now contains `("Zhang1san1" "who")`. `M375!` is pushed onto the front of the input buffer, which becomes `(M375! like B36)`.

The generator loops `TO` the state `G`. `M375` is expanded to `(B2 teach B30)`. The node `B2` is found representing the head noun of the relative clause, thus, it is omitted; making the input buffer `(teach B30 like B36)`. The node `teach` is `VERBIZE`d to `"taught"`. `B30` surfaces to `"English"`. Now the relative clause is completed with the register `SENTENCE` containing `("Zhang1san1" "who" "taught" "English")`. The generator continues processing the rest of the input buffer, `(like B36)`. They surface to `"liked"` and `"Mei3hua2"`, which are added to the `SENTENCE` making it `("Zhang1san1" "who" "taught" "English" "like" "Mei3hua2")`.

5.6 Generation of interrogative sentences

The generation of interrogative sentences is similar to that of declarative sentences except that when the subject is not a wh-pronoun, it requires fronting the auxiliary verb and, in wh-question, fronting the wh-pronoun. In the case of auxiliary verb fronting, if no auxiliary verb is available, one is made and moved to the beginning of the sentence; then, the main verb bears the root form. In the case of wh-pronoun fronting, the auxiliary verb fronting should apply first. Let's look at three examples of interrogative sentences generation:

誰　教　英文*?*
shei2　jiao1　Ying1 wen2?
who　teach　English
Who taught English? (5.11)

```
(M278! (CLASS (M211 (LEX English))) (MEMBER B10))

(P10 (ACT (M279 (LEX teach))) (AGENT V1) (OBJECT B10))
```

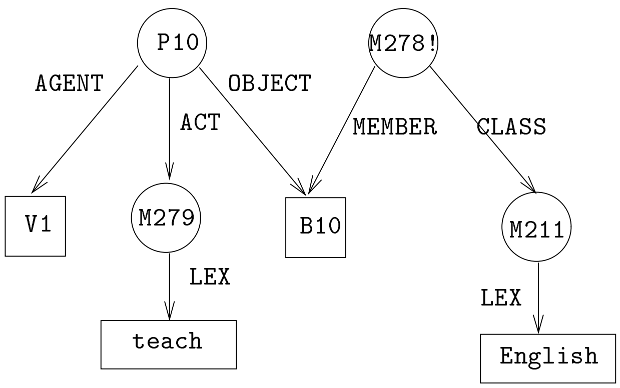

Figure 5.7: *Who taught English?*

The register `MOOD` contains `WH-QUESTION`, which means the sentence to be generated is an interrogative. The main proposition `P10` is expanded to `(teach V1 B10)`. These three nodes are ordered according to the verb `teach`'s lexical feature `SURFACE_ARGUMENTS` whose value is `AVO`, i.e. `AGENT:V1`

VERB:teach **O**BJECT:B10. Because the variable node V1, representing the human wh-pronoun *who*[5], is in subject position, neither wh-pronoun fronting nor auxiliary verb fronting are applied. V1, teach, and B10 surface as "Who", "taught", and "English" respectively. A question mark is attached to the end of SENTENCE making it ("Who" "taught" "English?").

Here is a second example:

張三	將要	吃	什麼?
Zhang1 san1	jiang1 yao4	chi1	She2 mo5?
Zhang1san1	will	eat	what

What will Zhang1san1 eat? (5.12)

```
(M261! (OBJECT B2) (PROPERNAME (M207 (LEX Zhang1san1))))

(P56 (ACT (M272 (LEX eat) (TENSE FUTURE))) (AGENT B2) (OBJECT V2))
```

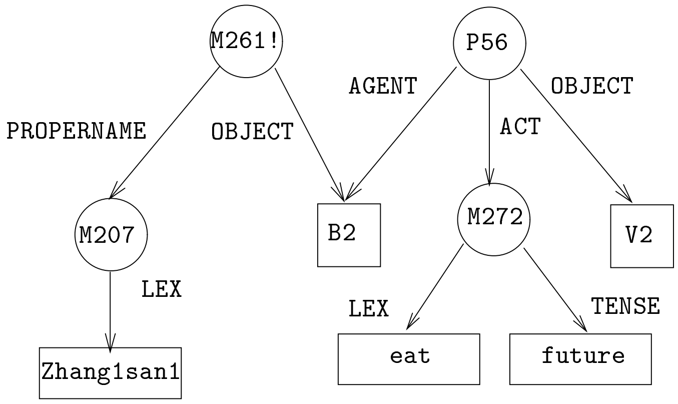

Figure 5.8: *What will Zhang1san1 eat?*

The main proposition P56 is expanded to (eat B2 V2). According to the SURFACE_ARGUMENTS of eat, these nodes are arranged into the order (B2 eat V2), which are AGENT, VERB, and OBJECT respectively. The generator goes on to surface each node. B2 surfaces to the subject "Zhang1san1". The verb eat surfaces to "will" "eat" because of the TENSE arc indicating future tense. Because

[5]Another variable node V2 represents the nonhuman wh-pronoun, *what*.

the subject, "Zhang1san1" is not a wh-pronoun, the auxiliary "will" is moved to the beginning of SENTENCE, ("will" "Zhang1san1" "eat"). For the same reason, the variable node V2 representing the nonhuman wh-pronoun, "what", is moved to the front of the SENTENCE. Then the question mark is attached to the end of SENTENCE, which now contains ("What" "will" "Zhang1san1" "eat?").

Let's look at an example of yes-no question:

李四	喜歡	張三	嗎?
Li3 si4	xi3 huan1	Zhang1 san1	ma1?
Li3si4	like	Zhang1san1	Q

Did Li3si4 like Zhang1san1? (5.13)

```
(M261! (OBJECT B2) (PROPERNAME (M207 (LEX Zhang1san1))))

(M262! (OBJECT B3) (PROPERNAME (M208 (LEX Li3si4))))

(M543! (ACT (M541 (LEX like))) (EXPERIENCER B3) (OBJECT B2))
```

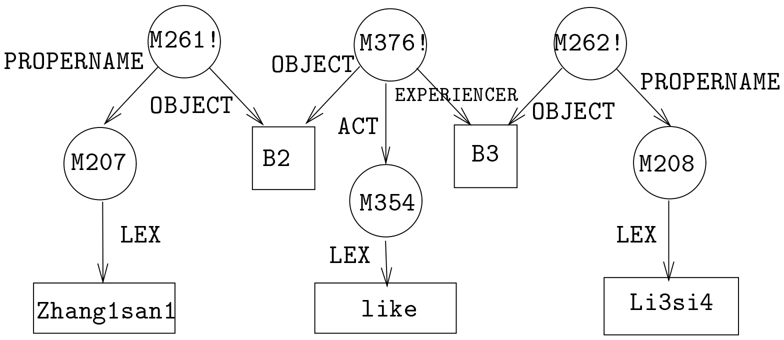

Figure 5.9: *Did Li3si4 like Zhang1san1?*

This example is an interrogative sentence because the register MOOD has the value YES-NO-Q. The main proposition M543! is expanded into three nodes which are arranged in syntactic order as (B3 like B2). Because the subject "Li3si4", represented by the node B3, is not a wh-pronoun, auxiliary verb fronting is required. Since no auxiliary verb is available and the default tense for the sentence is the past tense, the auxiliary verb `"did"` is built and moved to the front of `SENTENCE`. The node `like` is `VERBIZE`d to the root form, `" like"`. The last node `B2` surfaces to `"Zhang1san1"` then a question mark is attached to it. The final `SENTENCE` is (`"Did" "Li3si4" "like" "Zhang1san1?"`).

Although question answering is not part of the translation of the interrogatives, it is done to show `CASSIE`'s understanding of the questions. The parser `INDUCE` the answer to the interrogative using `SNIP` and stores the node representing the answer in the register `ANSWER`. After generating the interrogative, the generator prints the answer. If the register `ANSWER` is empty, it means there is no answer available, the string *"I don't know"* is printed to the screen; otherwise, the node in the register `ANSWER` surfaces in the same way as generating all other sentences. For the answer to yes-no questions, an additional string `"Yes,"` is added in front of it.

Chapter 6

Conclusion

The objective of this thesis has been to construct a Chinese-English machine translation system using SNePS as an interlingua. Most of the effort has been devoted to the design of unambiguous interlingua representations for various kinds of concepts and relations, and solving some difficult problems in parsing such as Chinese word segmentation, long-distant dependencies, and the disambiguation of Chinese sentences.

The parsing is lexicon-driven. In the lexicon, semantic case information integrated with syntactic information specifies meaning relations between predicates and their arguments. The lexical information further supplies syntactic information for sequencing predicates and their arguments. There have been two advantages with the lexicon-driven design. First, semantic case information together with the SNePS built-in inference mechanism have proved effective in resolving many kinds of ambiguities such as those occurring in the serial-verb constructions and sequences of nouns. Second, since most syntactic structures are captured in the definition of the verb, the size of the grammar is significantly reduced. However, something still needs improvement. For instance, the possible semantic case orders of verbs are encoded on an individual base; however, these orders should generalize across verbs. It would be nice to represent this generalization by classifying verbs according to the ordering of their semantic cases.

A bottom-up parsing design is implemented to handle two-way branching constructions and the relatively free word order in Chinese. The parser shifts the constituents parsed onto a stack holding off decisions on what rules applies until enough information is available. Based on adequate information, the bottom-up parser avoids endless recursion and reduces costly backtrackings. It is beyond the scope of the this project to construct a grammar which can translate most sentence patterns in Chinese. However, besides simple sentences, the parser is able to analyze interrogatives, relative clauses, and the Chinese serial-verb constructions. An annotated sample run in the appendix demonstrates that the system is able to deliver satisfactory results of translation.

There are several directions in which the present research can be extended.

- In this project, we are primarily concerned with difficulties arising from processing either monolingual components i.e, parsing Chinese or generating English. Some problems concerning contrastive aspects or the interface between Chinese and English have been given provisional solutions or simply ignored for the present. In Chinese and English, there are different ways of expressing, for example, number, definiteness, determination, and temporal references and so on. Lexical or structural equivalences are not easy to come by. For example, Chinese does not register number with the noun phrases; so, their English equivalents will default to the plural form. However, in some cases, the single form is preferred over the plural one. Systematic studies of these cross-linguistic differences are required for better quality of translation.

- In a useful natural language application to our machine-translation research, a Chinese generator can be built to allow multilingual interfaces to a data base or expert system. The annotated sample run in the appendix will show the query-answering capability of our system by responding to the Chinese interrogatives with English answers in addition to the English translations of the interrogatives. A Chinese generator would enable the user to make queries using Chinese and have the answers translated back into Chinese. There have been many CASSIE projects using an English interface. With this additional Chinese interface, those projects are becoming multilingual.

Appendix A

Knowledge base file

```
(noun (concrete abstract))
(concrete (animate inanimate color sound smell power))
(abstract (knowledge society emotion symbol time event))

(animate ((animal (property edible mobile))
          (plant (property edible))))
(inanimate ((place (property locative))
            (institution (property locative))
            (celestial (property locative))
            (object (property locative))
            (stuff (property locative))
            authority phenomenon activity))
(knowledge (language theory fact))
(society (culture politics education economy))

(animal (human nonhuman))
(plant (flower tree produce grain))
(stuff (supply machinery stationery
       (building (property locative))))
(activity (meal sports))

(human ((body (part head hand foot))
        woman man family person title profession))
```

```
(supply ((food (property edible))
         clothing money jewelry commodity cultural_material))
(machinery ((transportation (property mobile))
            part equipment tool))

(man ((father (property kinship))
      (son (property kinship))))
(woman ((mother (property kinship))
        (daughter (property kinship))))
(person (1st_pronoun 2nd_pronoun 3rd_pronoun))

(transportation (vehicle boat
                (aircraft (ability fly))))
```

Appendix B

Annotated sample run

This appendix presents an interaction running the Chinese-English translation system described in this thesis. This sample run is produced using the unix `script` command that can record the dribble of a demo session. In addition to giving the romanized representation for the Chinese input sentences and their word-for-word English gloss, annotations are provided to comment on or explain various phenomena. Some line breaks are changed so that long lines can fit the width of the pages. Annotations are preceded by semicolons '`;;;`'. SNePS Command follows the '*' prompt; and, LISP command follows the '`-->`' prompt. After the GATN parser is invoked, Chinese input sentences follows the ':' prompt.

```
>/projects/snwiz/bin/acl-sneps
Allegro CL 4.3 [SPARC; R1] (5/20/96 16:11)
Copyright (C) 1985-1996, Franz Inc., Berkeley, CA, USA.  All Rights Reserved.
;; Optimization settings: safety 1, space 0, speed 3, debug 3.
;; For a complete description of all compiler switches given the
;; current optimization settings evaluate (EXPLAIN-COMPILER-SETTINGS).

SNePS-2.3/SNeRE [PL:2 1996/02/21 20:38:33] loaded.
Type `(sneps)' or `(snepslog)' to get started.
USER(1): (sneps)


   Welcome to SNePS-2.3/SNeRE [PL:2 1996/02/21 20:38:33]

Copyright (C) 1984, 1988, 1989, 1993, 1994, 1995 by Research Foundation of
State University of New York. SNePS comes with ABSOLUTELY NO WARRANTY!
Type `(copyright)' for detailed copyright information.
Type `(demo)' for a list of example applications.

   11/26/1997 14:20:21

* (demo "MTdemo")

File MTdemo is now the source of input.

 CPU time : 0.00

* (resetnet t)

Net reset

 CPU time : 0.02

* (define agent object experiencer Benefactive locative lex member class
        superclass subclass part whole has-ability ability property propername
        main relc-a relc-o relc-e relc-b relc-l comp xcomp intent quant
        adposition arg1 arg2 kinship possessor rel modifier tense aspect)

(AGENT OBJECT EXPERIENCER BENEFACTIVE LOCATIVE LEX MEMBER CLASS
 SUPERCLASS SUBCLASS PART WHOLE HAS-ABILITY ABILITY PROPERTY PROPERNAME
 MAIN RELC-A RELC-O RELC-E RELC-B RELC-L COMP XCOMP INTENT QUANT
 ADPOSITION ARG1 ARG2 KINSHIP POSSESSOR REL MODIFIER TENSE ASPECT)

 CPU time : 0.02

* ;;; Define some useful paths for path-base inference.

;;; Path for going down the node tree to assert propositions e.g.,
;;; main clauses as well as the relative clauses. This way, propositions will
```

```
;;; only be built. They won't be asserted until we get the final valid parse;
;;; so, proposition nodes built by invalid parses or backtracking will not be
;;; asserted.
(define-path describe_assert
    (compose
     (or member (compose (or (compose property property-)
                             (compose propername propername-)
                             (compose adposition adposition-)) object))
     (kplus (or agent- object- benefactive- experiencer- locative-))
     (kstar (compose (kstar (compose
                             (or relc-a- relc-o- relc-e- relc-b- relc-l-)
                             main))
                     (kstar (or comp- intent- xcomp-))))))
DESCRIBE_ASSERT implied by the path
(COMPOSE
 (OR MEMBER
     (COMPOSE
      (OR (COMPOSE PROPERTY PROPERTY-)
          (COMPOSE PROPERNAME PROPERNAME-)
          (COMPOSE ADPOSITION ADPOSITION-))
      OBJECT))
 (KPLUS
  (OR AGENT- OBJECT- BENEFACTIVE- EXPERIENCER- LOCATIVE-))
 (KSTAR
  (COMPOSE
   (KSTAR
    (COMPOSE
     (OR RELC-A- RELC-O- RELC-E- RELC-B- RELC-L-)
     MAIN))
   (KSTAR (OR COMP- INTENT- XCOMP-)))))
DESCRIBE_ASSERT- implied by the path
(COMPOSE
 (KSTAR
  (COMPOSE (KSTAR (OR COMP INTENT XCOMP))
   (KSTAR
    (COMPOSE MAIN-
     (OR RELC-A RELC-O RELC-E RELC-B RELC-L)))))
 (KPLUS
  (OR AGENT OBJECT BENEFACTIVE EXPERIENCER LOCATIVE))
 (OR MEMBER-
     (COMPOSE OBJECT-
      (OR (COMPOSE PROPERTY PROPERTY-)
          (COMPOSE PROPERNAME PROPERNAME-)
          (COMPOSE ADPOSITION ADPOSITION-)))))

 CPU time : 0.02

* (define-path relative-clauses
    (kstar (compose
```

```
            (kplus (compose (or relc-a- relc-o- relc-e- relc-b- relc-l-) main))
            (kstar (or comp- intent- xcomp-)))))
RELATIVE-CLAUSES implied by the path
(KSTAR
 (COMPOSE
  (KPLUS
   (COMPOSE
    (OR RELC-A- RELC-O- RELC-E- RELC-B- RELC-L-)
    MAIN))
  (KSTAR
   (OR COMP- INTENT- XCOMP-))))
RELATIVE-CLAUSES- implied by the path
(KSTAR
 (COMPOSE
  (KSTAR (OR COMP INTENT XCOMP))
  (KPLUS
   (COMPOSE MAIN-
    (OR RELC-A RELC-O RELC-E RELC-B RELC-L)))))

 CPU time : 0.02

* ;;; Path for going down the node tree to describe every clause,
;;; relative clause as well as the main clause.
;;; Only propositions from the final valid parse will be described.
(define-path main-clauses
    (compose main
             (kstar (compose
                     (kstar (compose
                             (or relc-a- relc-o- relc-e- relc-b- relc-l-)
                             main))
                     (kstar (or comp- intent- xcomp-))))))
MAIN-CLAUSES implied by the path
(COMPOSE
 MAIN
 (KSTAR
  (COMPOSE
   (KSTAR
    (COMPOSE
     (OR RELC-A- RELC-O- RELC-E- RELC-B- RELC-L-)
     MAIN))
   (KSTAR (OR COMP- INTENT- XCOMP-)))))
MAIN-CLAUSES- implied by the path
(COMPOSE
 (KSTAR
  (COMPOSE
   (KSTAR (OR COMP INTENT XCOMP))
   (KSTAR
    (COMPOSE
     MAIN-
```

```
     (OR
      RELC-A RELC-O RELC-E RELC-B RELC-L)))))
 MAIN-)

 CPU time : 0.01

* ;;; Paths to allow porperties to be inherited down or up the hierarchy.
;;; Note:  In this project, objects (class membership relation) and
;;; classes of objects (the superclass relation) are distinguished.
(define-path class (compose class (kstar (compose subclass- superclass))))
CLASS implied by the path (COMPOSE CLASS
                           (KSTAR (COMPOSE SUBCLASS- SUPERCLASS)))
CLASS- implied by the path (COMPOSE
                            (KSTAR (COMPOSE SUPERCLASS- SUBCLASS))
                            CLASS-)

 CPU time : 0.00

* (define-path subclass
             (compose subclass (kstar(compose superclass- subclass))))
SUBCLASS implied by the path (COMPOSE SUBCLASS
                              (KSTAR (COMPOSE SUPERCLASS- SUBCLASS)))
SUBCLASS- implied by the path (COMPOSE
                               (KSTAR (COMPOSE SUBCLASS- SUPERCLASS))
                               SUBCLASS-)

 CPU time : 0.00

* (define-path part (compose part (kstar (compose whole- ! part))
                           (kstar (compose subclass- ! superclass))))
PART implied by the path (COMPOSE PART (KSTAR (COMPOSE WHOLE- ! PART))
                          (KSTAR (COMPOSE SUBCLASS- ! SUPERCLASS)))
PART- implied by the path (COMPOSE
                           (KSTAR (COMPOSE SUPERCLASS- ! SUBCLASS))
                           (KSTAR (COMPOSE PART- ! WHOLE)) PART-)

 CPU time : 0.00

* (define-path whole
             (compose whole (kstar (compose superclass- ! subclass))))
WHOLE implied by the path (COMPOSE WHOLE
                           (KSTAR (COMPOSE SUPERCLASS- ! SUBCLASS)))
WHOLE- implied by the path (COMPOSE
                            (KSTAR (COMPOSE SUBCLASS- ! SUPERCLASS))
                            WHOLE-)

 CPU time : 0.00

* (define-path object
```

```
    (or object
        (compose ! property property- ! object
                 (kstar (compose superclass- ! subclass)))))
OBJECT implied by the path (OR OBJECT
                               (COMPOSE ! PROPERTY PROPERTY- ! OBJECT
                                (KSTAR
                                 (COMPOSE SUPERCLASS- ! SUBCLASS))))
OBJECT- implied by the path (OR OBJECT-
                                (COMPOSE
                                 (KSTAR
                                  (COMPOSE SUBCLASS- ! SUPERCLASS))
                                 OBJECT- ! PROPERTY PROPERTY- !))

 CPU time : 0.00

* (define-path can
    (compose has-ability
             (or (kstar (compose subclass- ! superclass))
                 ;;obsevered ability
                 (compose member- ! class
                          (kstar (compose subclass- ! superclass)))
                 (kstar (compose superclass- ! subclass))
                 ;;inherited ability
                 (compose (kstar (compose superclass- ! subclass))
                          class- ! member))))
CAN implied by the path (COMPOSE HAS-ABILITY
                         (OR (KSTAR (COMPOSE SUBCLASS- ! SUPERCLASS))
                             (COMPOSE MEMBER- ! CLASS
                              (KSTAR (COMPOSE SUBCLASS- ! SUPERCLASS)))
                             (KSTAR (COMPOSE SUPERCLASS- ! SUBCLASS))
                             (COMPOSE
                              (KSTAR (COMPOSE SUPERCLASS- ! SUBCLASS))
                              CLASS- ! MEMBER)))
CAN- implied by the path (COMPOSE
                          (OR (KSTAR (COMPOSE SUPERCLASS- ! SUBCLASS))
                              (COMPOSE
                               (KSTAR (COMPOSE SUPERCLASS- ! SUBCLASS))
                               CLASS- ! MEMBER)
                              (KSTAR (COMPOSE SUBCLASS- ! SUPERCLASS))
                              (COMPOSE MEMBER- ! CLASS
                               (KSTAR
                                (COMPOSE SUBCLASS- ! SUPERCLASS))))
                          HAS-ABILITY-)

 CPU time : 0.01

* ^^

--> ;;; Define some pieces of advice to allow the GATN interpreter
```

```
;;; to read BIG5 encoded inputs.
;;; Escape punctuation chars e.g.,^|{},:,
;;; in the second byte of Chinese BIG5 encoding.
(excl:advise parser::convertline :before b5 nil
 (let ((line (car excl:arglist))) ;; Modify the argument list.
   (if (or (equal "" line) (lisp::find (schar line 0) '(#\^ #\;))) nil
     (setf (car excl:arglist)
       (with-output-to-string (s)
 (map nil #'(lambda (n) (if (lisp::find n ",:!()[]{}'`/#^|")
    (format s "~A~A" "\\" n);; put escape char
  (format s "~a" n))) line))))
   ;;; Do a scavenge gc before parsing.
   (excl:gc)))
PARSER:CONVERTLINE
--> (excl:compile-advice 'parser::convertline)
PARSER:CONVERTLINE
--> ;;; Loading the GATN grammar.
(atnin "grammar.b5")

State S processed.
State SP processed.
State ARGS processed.
State V processed.
State V/POSTVERB processed.
State V/V processed.
State V/PREVERB processed.
State V/NEGATIVE processed.
State V/END processed.
State S/END processed.
State G processed.
State G1 processed.
State NP processed.
State NP/ADJ processed.
State NP/CL processed.
State NP/QUANT processed.
State NP/DET processed.
State NP/DE processed.
State NP/END processed.
State PP processed.
State PP/NP processed.
State PP/PREP processed.
State PP/END processed.
; While compiling ROLE-ASSIGNMENT:
Warning: PARSER::GET-SENSES, :OPERATOR was defined in
         /projects/snwiz/Install/Sneps-2.3.2-acl4.3/nlint/parser/parser.lisp
         and is now being defined at the top level

 Atnin read in states: (NIL NIL NIL NIL NIL NIL NIL NIL NIL NIL NIL NIL
                        NIL NIL NIL NIL NIL NIL NIL NIL NIL NIL NIL NIL
                        NIL NIL NIL NIL PP/END PP/PREP PP/NP PP NP/END
                        NP/DE NP/DET NP/QUANT NP/CL NP/ADJ NP G1 G
                        S/END V/END V/NEGATIVE V/PREVERB V/V V/POSTVERB
                        V ARGS SP S)
```

```
--> ;;; Two lexicons, one for parsing Chinese and another for generating English.
;;; Make sure we always start with the Chinese lexicon, even on interrupted or
;;; aborted generation which doesn't switch English lexicon back to Chinese.
(and (boundp '*lexicon*ch) (setq englex:*lexicon* *lexicon*ch))
NIL
--> ;;; Load English lexicon; then, map it into the hashtable, *lexicon*eng.
;;; (Create a hashtable for English lexicon if none exists.)
(prog1
  (lexin "lexicon.english")
  (or (boundp '*lexicon*eng)(setq *lexicon*eng (make-hash-table :test 'equal)))
  (maphash #'(lambda (k v)(setf (gethash k *lexicon*eng) v)) englex:*lexicon*))
undefined- (NIL)
("Li3si4" "Zhang1san1" "Wang2wu3" "Lao3li3" "Lao3zhang1" "Lao3wang2"
 "Mei3hua2" "Taipei" "Tweety" "Dumbo" "Clyde" "she" "English"
 "Japanese" "language" "Linguistics" "television" "antenna"
 "refrigerator" "meeting" "school" "yin2hang2" "janitor" "bank" "book"
 "cow" "milk" "table" "money" "rock" "well" "job" "bird" "canary"
 "elephant" "son" "bowl" "bill" "officer" "soldier" "order" "bicycle"
 "jewel" "dinner" "eat" "break" "fly" "sleep" "believe" "work" "drink"
 "promise" "represent" "attend" "persuade" "study" "fall" "teach"
 "give" "steal" "attempt" "rob" "come" "go" "run" "buy" "adorn" "use"
 "read" "like" "dislike" "kiss" "insist" "pay" "obey" "consider"
 "change" "decide" "congratulate" "get" "suggest" "see" "ride"
 "remember" "put" "to" "in" "on" "fast" "beautiful" "big" "red" "new"
 "Chinese" "wooden" "round" "young" "smart" "happy" "one")
--> ;;; Load Chinese lexicon for parsing.
(lexin "lexicon.b5")
undefined- (NIL)
("李四" "四" "老李" "李" "張三" "三" "老張" "張" "王五" "五"
 "老王" "王" "志成" "成" "大偉" "偉" "保民" "民" "建正" "正"
 "偉健" "健" "台北" "北" "興華" "大華" "國華" "美華" "華" "甜啼"
 "啼" "丹寶" "寶" "卡來迪" "迪" "誰" "什麼" "她" "麼" "會議"
 "學校" "校" "銀行" "行" "工友" "友" "石頭" "頭" "書" "牛" "牛奶"
 "奶" "桌" "錢" "英文" "日文" "文" "語言" "言" "語言學" "學"
 "電視" "視" "天線" "線" "冰箱" "箱" "井" "事" "工作" "作" "鳥"
 "金絲雀" "雀" "大象" "象" "兒子" "子" "碗" "帳" "軍官" "官"
 "士兵" "兵" "命令" "令" "單車" "車" "珠寶" "晚飯" "飯" "吃"
 "打破" "破" "飛" "睡覺" "覺" "念" "做" "相信" "信" "習諾" "諾"
 "代表" "表" "出席" "席" "勸" "掉" "教" "給" "送" "偷" "來" "去" "跑"
 "買" "身飾" "飾" "用" "讀" "喝" "喜歡" "歡" "厭惡" "惡" "企圖"
 "圖" "搶" "親" "答應" "應" "堅持" "持" "付" "服從" "從" "考慮"
 "慮" "換" "決定" "定" "恭喜" "喜" "得到" "到" "建議" "議" "見"
 "騎" "記得" "得" "放" "在" "把" "裡" "上" "下" "快" "漂亮" "亮" "大"
 "紅色" "色" "新的" "中國的" "的" "木製" "製" "圓" "年青" "青"
 "高興" "興" "聰明" "明" "將要" "要" "一" "那" "這" "家" "隻" "個"
 "本" "位" "台" "." "?")
--> (defsnepscom isa ((who what) assert)
  "Assert subclass WHO is a superclass WHAT."
```

```
  #!((describe
      (assert subclass (build lex ~who) superclass (build lex ~what)
              :context KB))))
T
--> ;;; Read the file KB and build the knowledge base according to it.
;;; The KB is a hierarchical structure including features, properties,
;;; and other relations which can be inherited down or up the hierarchy.
;;; For efficiency reason, we create a KB context.
;;; Deductions for parsing will work in the KB context; whereas,
;;; deductions on interrogatives will work in the default context.
(with-open-file (KB "KB" :direction :input)
  (set-context nil KB)
  (set-default-context KB)
  (let (stream superclass)
    (loop (when (null (setq stream (read KB nil nil))) (return))
      (setq superclass (car stream))
      (dolist (subclass (second stream))
        (cond
         ((atom subclass) #!((isa ~subclass ~superclass)))
         (t (let* ((obj #!((find subclass-
                              ~#!((isa ~(car subclass) ~superclass)))))
                   (features (cdr subclass))
                   (property (assoc 'property features))
                   (ability (assoc 'ability features))
                   (part (assoc 'part features)))
              (if property
                  (dolist (property (cdr property))
                    (setq property (string-downcase (symbol-name property)))
                    #!((describe (assert object ~obj
                                   property (build lex ~property))))))
              (if ability
                  (dolist (ability (cdr ability))
                    (setq ability (string-downcase (symbol-name ability)))
                    #!((describe (assert has-ability ~obj
                                   ability (build lex ~ability))))))
              (if part
                  (dolist (part (cdr part))
                    (setq part (string-downcase (symbol-name part)))
                    #!((describe (assert whole ~obj
                                   part (build lex ~part))))))))))))


((ASSERTIONS NIL) (RESTRICTION NIL) (NAMED (KB DEFAULT-DEFAULTCT)))
((ASSERTIONS NIL) (RESTRICTION NIL) (NAMED (KB DEFAULT-DEFAULTCT)))
(M3! (SUBCLASS (M1 (LEX CONCRETE))) (SUPERCLASS (M2 (LEX NOUN))))
(M5! (SUBCLASS (M4 (LEX ABSTRACT))) (SUPERCLASS (M2 (LEX NOUN))))
(M7! (SUBCLASS (M6 (LEX ANIMATE))) (SUPERCLASS (M1 (LEX CONCRETE))))
(M9! (SUBCLASS (M8 (LEX INANIMATE))) (SUPERCLASS (M1 (LEX CONCRETE))))
(M11! (SUBCLASS (M10 (LEX COLOR))) (SUPERCLASS (M1 (LEX CONCRETE))))
(M13! (SUBCLASS (M12 (LEX SOUND))) (SUPERCLASS (M1 (LEX CONCRETE))))
(M15! (SUBCLASS (M14 (LEX SMELL))) (SUPERCLASS (M1 (LEX CONCRETE))))
```

```
(M17! (SUBCLASS (M16 (LEX POWER))) (SUPERCLASS (M1 (LEX CONCRETE))))
(M19! (SUBCLASS (M18 (LEX KNOWLEDGE))) (SUPERCLASS (M4 (LEX ABSTRACT))))
(M21! (SUBCLASS (M20 (LEX SOCIETY))) (SUPERCLASS (M4 (LEX ABSTRACT))))
(M23! (SUBCLASS (M22 (LEX EMOTION))) (SUPERCLASS (M4 (LEX ABSTRACT))))
(M25! (SUBCLASS (M24 (LEX SYMBOL))) (SUPERCLASS (M4 (LEX ABSTRACT))))
(M27! (SUBCLASS (M26 (LEX TIME))) (SUPERCLASS (M4 (LEX ABSTRACT))))
(M29! (SUBCLASS (M28 (LEX EVENT))) (SUPERCLASS (M4 (LEX ABSTRACT))))
(M31! (SUBCLASS (M30 (LEX ANIMAL))) (SUPERCLASS (M6 (LEX ANIMATE))))
(M33! (OBJECT (M30 (LEX ANIMAL))) (PROPERTY (M32 (LEX edible))))
(M35! (OBJECT (M30 (LEX ANIMAL))) (PROPERTY (M34 (LEX mobile))))
(M37! (SUBCLASS (M36 (LEX PLANT))) (SUPERCLASS (M6 (LEX ANIMATE))))
(M38! (OBJECT (M36 (LEX PLANT))) (PROPERTY (M32 (LEX edible))))
(M40! (SUBCLASS (M39 (LEX PLACE))) (SUPERCLASS (M8 (LEX INANIMATE))))
(M42! (OBJECT (M39 (LEX PLACE))) (PROPERTY (M41 (LEX locative))))
(M44! (SUBCLASS (M43 (LEX INSTITUTION)))(SUPERCLASS (M8 (LEX INANIMATE))))
(M45! (OBJECT (M43 (LEX INSTITUTION))) (PROPERTY (M41 (LEX locative))))
(M47! (SUBCLASS (M46 (LEX CELESTIAL)))(SUPERCLASS (M8 (LEX INANIMATE))))
(M48! (OBJECT (M46 (LEX CELESTIAL))) (PROPERTY (M41 (LEX locative))))
(M50! (SUBCLASS (M49 (LEX OBJECT))) (SUPERCLASS (M8 (LEX INANIMATE))))
(M51! (OBJECT (M49 (LEX OBJECT))) (PROPERTY (M41 (LEX locative))))
(M53! (SUBCLASS (M52 (LEX STUFF))) (SUPERCLASS (M8 (LEX INANIMATE))))
(M54! (OBJECT (M52 (LEX STUFF))) (PROPERTY (M41 (LEX locative))))
(M56! (SUBCLASS (M55 (LEX AUTHORITY)))(SUPERCLASS (M8 (LEX INANIMATE))))
(M58! (SUBCLASS (M57 (LEX PHENOMENON)))(SUPERCLASS (M8 (LEX INANIMATE))))
(M60! (SUBCLASS (M59 (LEX ACTIVITY))) (SUPERCLASS (M8 (LEX INANIMATE))))
(M62! (SUBCLASS (M61 (LEX LANGUAGE)))(SUPERCLASS (M18 (LEX KNOWLEDGE))))
(M64! (SUBCLASS (M63 (LEX THEORY))) (SUPERCLASS (M18 (LEX KNOWLEDGE))))
(M66! (SUBCLASS (M65 (LEX FACT))) (SUPERCLASS (M18 (LEX KNOWLEDGE))))
(M68! (SUBCLASS (M67 (LEX CULTURE))) (SUPERCLASS (M20 (LEX SOCIETY))))
(M70! (SUBCLASS (M69 (LEX POLITICS))) (SUPERCLASS (M20 (LEX SOCIETY))))
(M72! (SUBCLASS (M71 (LEX EDUCATION))) (SUPERCLASS (M20 (LEX SOCIETY))))
(M74! (SUBCLASS (M73 (LEX ECONOMY))) (SUPERCLASS (M20 (LEX SOCIETY))))
(M76! (SUBCLASS (M75 (LEX HUMAN))) (SUPERCLASS (M30 (LEX ANIMAL))))
(M78! (SUBCLASS (M77 (LEX NONHUMAN))) (SUPERCLASS (M30 (LEX ANIMAL))))
(M80! (SUBCLASS (M79 (LEX FLOWER))) (SUPERCLASS (M36 (LEX PLANT))))
(M82! (SUBCLASS (M81 (LEX TREE))) (SUPERCLASS (M36 (LEX PLANT))))
(M84! (SUBCLASS (M83 (LEX PRODUCE))) (SUPERCLASS (M36 (LEX PLANT))))
(M86! (SUBCLASS (M85 (LEX GRAIN))) (SUPERCLASS (M36 (LEX PLANT))))
(M88! (SUBCLASS (M87 (LEX SUPPLY))) (SUPERCLASS (M52 (LEX STUFF))))
(M90! (SUBCLASS (M89 (LEX MACHINERY))) (SUPERCLASS (M52 (LEX STUFF))))
(M92! (SUBCLASS (M91 (LEX STATIONERY))) (SUPERCLASS (M52 (LEX STUFF))))
(M94! (SUBCLASS (M93 (LEX BUILDING))) (SUPERCLASS (M52 (LEX STUFF))))
(M95! (OBJECT (M93 (LEX BUILDING))) (PROPERTY (M41 (LEX locative))))
(M97! (SUBCLASS (M96 (LEX MEAL))) (SUPERCLASS (M59 (LEX ACTIVITY))))
(M99! (SUBCLASS (M98 (LEX SPORTS))) (SUPERCLASS (M59 (LEX ACTIVITY))))
(M101! (SUBCLASS (M100 (LEX BODY))) (SUPERCLASS (M75 (LEX HUMAN))))
(M103! (PART (M102 (LEX head))) (WHOLE (M100 (LEX BODY))))
(M105! (PART (M104 (LEX hand))) (WHOLE (M100 (LEX BODY))))
(M107! (PART (M106 (LEX foot))) (WHOLE (M100 (LEX BODY))))
(M109! (SUBCLASS (M108 (LEX WOMAN))) (SUPERCLASS (M75 (LEX HUMAN))))
(M111! (SUBCLASS (M110 (LEX MAN))) (SUPERCLASS (M75 (LEX HUMAN))))
(M113! (SUBCLASS (M112 (LEX FAMILY))) (SUPERCLASS (M75 (LEX HUMAN))))
(M115! (SUBCLASS (M114 (LEX PERSON))) (SUPERCLASS (M75 (LEX HUMAN))))
(M117! (SUBCLASS (M116 (LEX TITLE))) (SUPERCLASS (M75 (LEX HUMAN))))
(M119! (SUBCLASS (M118 (LEX PROFESSION)))(SUPERCLASS (M75 (LEX HUMAN))))
(M121! (SUBCLASS (M120 (LEX FOOD))) (SUPERCLASS (M87 (LEX SUPPLY))))
```

```
(M122! (OBJECT (M120 (LEX FOOD))) (PROPERTY (M32 (LEX edible))))
(M124! (SUBCLASS (M123 (LEX CLOTHING))) (SUPERCLASS (M87 (LEX SUPPLY))))
(M126! (SUBCLASS (M125 (LEX MONEY))) (SUPERCLASS (M87 (LEX SUPPLY))))
(M128! (SUBCLASS (M127 (LEX JEWELRY))) (SUPERCLASS (M87 (LEX SUPPLY))))
(M130! (SUBCLASS (M129 (LEX COMMODITY)))(SUPERCLASS (M87 (LEX SUPPLY))))
(M132! (SUBCLASS (M131 (LEX CULTURAL_MATERIAL)))(SUPERCLASS (M87 (LEX SUPPLY))))
(M134! (SUBCLASS (M133 (LEX TRANSPORTATION)))(SUPERCLASS (M89 (LEX MACHINERY))))
(M135! (OBJECT (M133 (LEX TRANSPORTATION)))(PROPERTY (M34 (LEX mobile))))
(M137! (SUBCLASS (M136 (LEX PART))) (SUPERCLASS (M89 (LEX MACHINERY))))
(M139! (SUBCLASS (M138 (LEX EQUIPMENT)))(SUPERCLASS (M89 (LEX MACHINERY))))
(M141! (SUBCLASS (M140 (LEX TOOL))) (SUPERCLASS (M89 (LEX MACHINERY))))
(M143! (SUBCLASS (M142 (LEX FATHER))) (SUPERCLASS (M110 (LEX MAN))))
(M145! (OBJECT (M142 (LEX FATHER))) (PROPERTY (M144 (LEX kinship))))
(M147! (SUBCLASS (M146 (LEX SON))) (SUPERCLASS (M110 (LEX MAN))))
(M148! (OBJECT (M146 (LEX SON))) (PROPERTY (M144 (LEX kinship))))
(M150! (SUBCLASS (M149 (LEX MOTHER))) (SUPERCLASS (M108 (LEX WOMAN))))
(M151! (OBJECT (M149 (LEX MOTHER))) (PROPERTY (M144 (LEX kinship))))
(M153! (SUBCLASS (M152 (LEX DAUGHTER))) (SUPERCLASS (M108 (LEX WOMAN))))
(M154! (OBJECT (M152 (LEX DAUGHTER))) (PROPERTY (M144 (LEX kinship))))
(M156! (SUBCLASS (M155 (LEX 1ST_PRONOUN)))(SUPERCLASS (M114 (LEX PERSON))))
(M158! (SUBCLASS (M157 (LEX 2ND_PRONOUN)))(SUPERCLASS (M114 (LEX PERSON))))
(M160! (SUBCLASS (M159 (LEX 3RD_PRONOUN)))(SUPERCLASS (M114 (LEX PERSON))))
(M162! (SUBCLASS (M161 (LEX VEHICLE)))
 (SUPERCLASS (M133 (LEX TRANSPORTATION))))
(M164! (SUBCLASS (M163 (LEX BOAT)))
 (SUPERCLASS (M133 (LEX TRANSPORTATION))))
(M166! (SUBCLASS (M165 (LEX AIRCRAFT)))
 (SUPERCLASS (M133 (LEX TRANSPORTATION))))
(M168! (ABILITY (M167 (LEX fly))) (HAS-ABILITY (M165 (LEX AIRCRAFT))))
NIL

--> ;;; The lexicon also provides some knowledge. Read it to expand the KB.
(with-open-file (KB "lexicon.b5" :direction :input)
  (let (stream)
    (loop (when (null (setq stream (read KB nil nil))) (return))
      (if (member (rest (assoc 'ctgy (second stream))) '(n npr pronoun))
          (let* ((features (second stream))
                 (object (rest (assoc 'sense features)))
                 (superclass (rest (assoc 'superclass features)))
                 (part (rest (assoc 'part features)))
                 (kinship (rest (assoc 'kinship features)))
                 (property (rest (assoc 'property features)))
                 (ability (rest (assoc 'ability features))))
            (if kinship
                (progn #!((assert object #B2 propername (build lex ~object)))
                       (dolist (rel kinship)
                         #!((describe (assert arg1 #B1 arg2 *B2
                                              kinship (build lex ~rel)))))))
            (if superclass #!((isa ~object ~superclass)))
            (if property #!((describe (assert object (build lex ~object)
                                       property (build lex ~property)))))
            (if ability
                #!((describe
```

```
                (assert has-ability (build lex ~object)
                  ability (build lex ~(string-downcase
                                       (symbol-name ability)))))))
        (if part
            #!((describe
                (assert whole (build lex ~object)
                  part (build lex ~(mapcar
                                     #'(lambda (n)
                                         (string-downcase
                                          (format nil "~A" n))) part))))))
        )))))
```

```
(M170! (SUBCLASS (M169 (LEX Li3si4))) (SUPERCLASS (M110 (LEX MAN))))
(M172! (SUBCLASS (M171 (LEX Lao3li3))) (SUPERCLASS (M110 (LEX MAN))))
(M174! (SUBCLASS (M173 (LEX Zhang1san1))) (SUPERCLASS (M110 (LEX MAN))))
(M176! (SUBCLASS (M175 (LEX Lao3zhang1))) (SUPERCLASS (M110 (LEX MAN))))
(M178! (SUBCLASS (M177 (LEX Wang2wu3))) (SUPERCLASS (M110 (LEX MAN))))
(M180! (SUBCLASS (M179 (LEX Lao3wang2))) (SUPERCLASS (M110 (LEX MAN))))
(M182! (SUBCLASS (M181 (LEX Zhi4cheng2))) (SUPERCLASS (M110 (LEX MAN))))
(M184! (SUBCLASS (M183 (LEX Da4wei3))) (SUPERCLASS (M110 (LEX MAN))))
(M186! (SUBCLASS (M185 (LEX Bao3min2))) (SUPERCLASS (M110 (LEX MAN))))
(M188! (SUBCLASS (M187 (LEX jian4zheng4)))(SUPERCLASS (M110 (LEX MAN))))
(M190! (SUBCLASS (M189 (LEX wei3jian4))) (SUPERCLASS (M110 (LEX MAN))))
(M192! (SUBCLASS (M191 (LEX Taipei))) (SUPERCLASS (M39 (LEX PLACE))))
(M194! (SUBCLASS (M193 (LEX xing1hua2))) (SUPERCLASS (M110 (LEX MAN))))
(M196! (SUBCLASS (M195 (LEX Da4hua2))) (SUPERCLASS (M110 (LEX MAN))))
(M198! (SUBCLASS (M197 (LEX Guo2hua2))) (SUPERCLASS (M110 (LEX MAN))))
(M200! (SUBCLASS (M199 (LEX Mei3hua2))) (SUPERCLASS (M108 (LEX WOMAN))))
(M202! (SUBCLASS (M201 (LEX who)))(SUPERCLASS (M159 (LEX 3RD_PRONOUN))))
(M204! (SUBCLASS (M203 (LEX she))) (SUPERCLASS (M108 (LEX WOMAN))))
(M206! (SUBCLASS (M205 (LEX meeting)))(SUPERCLASS (M59 (LEX ACTIVITY))))
(M208! (SUBCLASS (M207 (LEX school)))(SUPERCLASS (M43 (LEX INSTITUTION))))
(M210! (SUBCLASS (M209 (LEX bank)))(SUPERCLASS (M43 (LEX INSTITUTION))))
(M212! (SUBCLASS (M211 (LEX janitor))) (SUPERCLASS (M75 (LEX HUMAN))))
(M214! (SUBCLASS (M213 (LEX rock))) (SUPERCLASS (M49 (LEX OBJECT))))
(M216! (SUBCLASS (M215 (LEX book)))
 (SUPERCLASS (M131 (LEX CULTURAL_MATERIAL))))
(M218! (SUBCLASS (M217 (LEX cow))) (SUPERCLASS (M77 (LEX NONHUMAN))))
(M220! (SUBCLASS (M219 (LEX milk))) (SUPERCLASS (M120 (LEX FOOD))))
(M222! (OBJECT (M219 (LEX milk))) (PROPERTY (M221 (LEX LIQUID))))
(M224! (SUBCLASS (M223 (LEX table))) (SUPERCLASS (M52 (LEX STUFF))))
(M226! (SUBCLASS (M225 (LEX English)))(SUPERCLASS (M18 (LEX KNOWLEDGE))))
(M228! (SUBCLASS (M227 (LEX Japanese)))(SUPERCLASS (M18 (LEX KNOWLEDGE))))
(M230! (SUBCLASS (M229 (LEX language)))(SUPERCLASS (M18 (LEX KNOWLEDGE))))
(M232! (SUBCLASS (M231 (LEX linguistics)))(SUPERCLASS (M18 (LEX KNOWLEDGE))))
(M234! (SUBCLASS (M233 (LEX television)))(SUPERCLASS (M138 (LEX EQUIPMENT))))
(M236! (PART (M235 (LEX antenna screen))) (WHOLE (M233 (LEX television))))
(M238! (SUBCLASS (M237 (LEX antenna))) (SUPERCLASS (M138 (LEX EQUIPMENT))))
(M240! (SUBCLASS (M239 (LEX refrigerator)))
 (SUPERCLASS (M138 (LEX EQUIPMENT))))
(M242! (SUBCLASS (M241 (LEX well))) (SUPERCLASS (M49 (LEX OBJECT))))
(M244! (SUBCLASS (M243 (LEX work))) (SUPERCLASS (M59 (LEX ACTIVITY))))
(M246! (SUBCLASS (M245 (LEX job))) (SUPERCLASS (M59 (LEX ACTIVITY))))
```

```
(M248! (SUBCLASS (M247 (LEX bird))) (SUPERCLASS (M30 (LEX ANIMAL))))
(M249! (ABILITY (M167 (LEX fly))) (HAS-ABILITY (M247 (LEX bird))))
(M251! (SUBCLASS (M250 (LEX canary))) (SUPERCLASS (M247 (LEX bird))))
(M253! (SUBCLASS (M252 (LEX elephant))) (SUPERCLASS (M30 (LEX ANIMAL))))
(M255! (SUBCLASS (M254 (LEX son))) (SUPERCLASS (M110 (LEX MAN))))
(M257! (SUBCLASS (M256 (LEX bowl))) (SUPERCLASS (M129 (LEX COMMODITY))))
(M259! (SUBCLASS (M258 (LEX bill))) (SUPERCLASS (M125 (LEX MONEY))))
(M261! (SUBCLASS (M260 (LEX officer)))(SUPERCLASS (M118 (LEX PROFESSION))))
(M263! (SUBCLASS (M262 (LEX soldier)))(SUPERCLASS (M118 (LEX PROFESSION))))
(M265! (SUBCLASS (M264 (LEX order))) (SUPERCLASS (M55 (LEX AUTHORITY))))
(M267! (SUBCLASS (M266 (LEX bicycle))) (SUPERCLASS (M161 (LEX VEHICLE))))
(M269! (SUBCLASS (M268 (LEX jewel))) (SUPERCLASS (M127 (LEX JEWELRY))))
(M271! (SUBCLASS (M270 (LEX dinner))) (SUPERCLASS (M96 (LEX MEAL))))
NIL

--> (parse)

 ATN parser initialization...

 Trace level = 0.

 Beginning at state 'S'.


 Input sentences in normal English orthographic convention.
 Sentences may go beyond a line by having a space followed by a <CR>
 To exit the parser, write ^end.


 : ^^

 Enter Lisp Read/Eval/Print loop. Type ^^ to continue

--> ;;; Create variable nodes for wh-questions.
;;; Set context back to the default context.
(progn (setq *infertrace* nil *parse-trees* T *all-parses* nil)
       (set-default-context DEFAULT-DEFAULTCT)
       (values ($ 'who) ($ 'what)))

((ASSERTIONS NIL) (RESTRICTION NIL) (NAMED (DEFAULT-DEFAULTCT)))
(V1)
(V2)
--> ^^

 : ;;; All the three following sentences mean "Zhang1san1 read book."
;;; Nei4  ben3  shu1  Zhang1 san1  du2  le5. (CL: classifier;
;;; that   CL   book  Zhang1san1   read LE    LE: perfective aspect)
 那本書張三讀了.

(M272! (OBJECT B1) (PROPERNAME (M173 (LEX Zhang1san1))))
(M274! (CLASS (M215 (LEX book))) (MEMBER B4))
```

```
(M278! (ACT (M277 (LEX read) (TENSE PAST))) (AGENT B1) (OBJECT B4))

 Zhang1san1 read that book.

 Time (sec.): 0.1

 : ;;;
;;; Zhang1 san1  nei4  ben3  shu1  du2  le5.
;;; Zhang1san1   that   CL   book  read LE
 張三那本書讀了.

(M279! (CLASS (M215 (LEX book))) (MEMBER B5))
(M281! (ACT (M277 (LEX read) (TENSE PAST))) (AGENT B1) (OBJECT B5))

 Zhang1san1 read that book.

 Time (sec.): 0.06

 : ;;;
;;; Zhang1 san1  du2  le5  nei4  ben3  shu1.
;;; Zhang1san1   read LE   that   CL   book
 張三讀了那本書.

(M282! (CLASS (M215 (LEX book))) (MEMBER B7))
(M284! (ACT (M277 (LEX read) (TENSE PAST))) (AGENT B1) (OBJECT B7))

 Zhang1san1 read that book.

 Time (sec.): 0.07

 : ;;; For an interrgative, besides its English translation,
;;; the system answers the query. The answer follows the 'CASSIE:' prompt.
;;; Shei2  du2   shu1?
;;;  who   read  book
 誰讀書?

(P2 (ACT (M285 (LEX read))) (AGENT V1) (OBJECT B7))

 Who read that book?

 CASSIE:  Zhang1san1 read that book.

 Time (sec.): 0.29

 : ;;; No parse due to the wrong use of pronoun.
;;; "read" requires a human agent; however, "what" is for non-human.
;;; She2 mo5  du2  shu1?
;;;   what    read book
 什麼讀書?
```

```
 Time (sec.): 0.21

 : ;;;
;;; Zhang1 san1  du2   she2 mo5?
;;; Zhang1san1   read    what
 張三讀什麼?

(P5 (ACT (M285 (LEX read))) (AGENT B1) (OBJECT V2))

 What did Zhang1san1 read?

 CASSIE:  Zhang1san1 read that book.

 CASSIE:  Zhang1san1 read that book.

 CASSIE:  Zhang1san1 read that book.

 Time (sec.): 0.3

 : ;;;
;;; Zhang1 san1  du2  le5  nei4  ben3  shu1  ma1?
;;; Zhang1san1   read LE   that  CL    book  Q    (Q= question marker)
 張三讀了那本書嗎?

 Did Zhang1san1 read that book?

 CASSIE:  Yes, Zhang1san1 read that book.

 Time (sec.): 0.22

 : ;;; Ambiguities in a sequence of nouns without conjunctions.
;;; Noun Verb  Noun    Noun
;;; ===========================
;;; John teach English Japanese
;;; ---------------------------
;;; John teach Mary    Japanese
;;;
;;; Noun    Noun Verb  Noun
;;; ===========================
;;; English John teach Mary
;;; ---------------------------
;;; Mary    John teach English
;;;
;;; Noun Verb  Noun Noun    Noun    Noun
;;; ==========================================
;;; John teach Mary Allen   Pat     Linguistics (conjoined nouns)
;;; ------------------------------------------
;;; John teach Mary English AI      Linguistics (conjoined nouns)
```

```
;;; -------------------------------------------
;;; John teach Math English AI      Linguistics (conjoined nouns)
;;; -------------------------------------------
;;; John teach Mary Jane    English Linguistics (conjoined nouns)
;;; -------------------------------------------
;;; John teach Mary brother AI      Linguistics (kinship relation)
;;; -------------------------------------------
;;; John give  Mary brother TV      antenna     (part-whole relation)
;;;
;;; Zhang1 san1  jiao1  Ying1 wen2  Ri4 wen2.
;;; Zhang1san1   teach   English    Japanese
 張三教英文日文.

(M290! (CLASS (M227 (LEX Japanese))) (MEMBER B9))
(M291! (CLASS (M225 (LEX English))) (MEMBER B10))
(M293! (ACT (M292 (LEX teach))) (AGENT B1) (OBJECT B10 B9))

 Zhang1san1 taught English and Japanese.

 Time (sec.): 0.09

 : ;;; Shouldn't reply "Zhang1san1 taught Japanese English."
;;; Zhang1 san1  jiao1  shei2  Ri4 wen2.
;;; Zhang1san1   teach   who   Japanese
 張三教誰日文?

(P8 (ACT (M292 (LEX teach))) (AGENT B1) (EXPERIENCER V1) (OBJECT B9))

 Whom did Zhang1san1 teach Japanese?

 CASSIE:  I don't know.

 Time (sec.): 0.32

 : ;;; Reduction Inference.
;;; Shei2  jiao1  Ri4 wen2?
;;; who    teach  Japanese
 誰教日文?

(P10 (ACT (M292 (LEX teach))) (AGENT V1) (OBJECT B9))

 Who taught Japanese?

 CASSIE:  Zhang1san1 taught Japanese.

 Time (sec.): 0.28

 : ;;;
;;; Zhang1 san1  jiao1  Li3 si4  Ri4 wen2.
```

```
;;; Zhang1san1   teach  Li3si4   Japanese
 張三教李四日文.

(M295! (CLASS (M227 (LEX Japanese))) (MEMBER B11))
(M296! (OBJECT B12) (PROPERNAME (M169 (LEX Li3si4))))
(M297! (ACT (M292 (LEX teach))) (AGENT B1) (EXPERIENCER B12)
 (OBJECT B11))

 Zhang1san1 taught Li3si4 Japanese.

 Time (sec.): 0.1

 : ;;; Same question. Last time, CASSIE answered "I don't know."
;;; Zhang1 san1  jiao1  shei2  Ri4 wen2.
;;; Zhang1san1   teach   who   Japanese
 張三教誰日文?

(P12 (ACT (M292 (LEX teach))) (AGENT B1) (EXPERIENCER V1) (OBJECT B11))

 Whom did Zhang1san1 teach Japanese?

 CASSIE:  Zhang1san1 taught Li3si4 Japanese.

 Time (sec.): 0.41

 : ;;;
;;; Ying1 wen2  Zhang1 san1  jiao1  Li3 si4  le5.
;;;  English    Zhang1san1   teach  Li3si4   LE
 英文張三教李四了.

(M299! (CLASS (M225 (LEX English))) (MEMBER B14))
(M301! (ACT (M300 (LEX teach) (TENSE PAST))) (AGENT B1)
 (EXPERIENCER B12) (OBJECT B14))

 Zhang1san1 taught Li3si4 English.

 Time (sec.): 0.09

 : ;;; Answer not only "English" but also
;;; "Japanese" inferred from previous inputs
;;; Zhang1 san1  jiao1  Li3 si4  she2 mo5?
;;; Zhang1san1   teach  Li3si4     what
 張三教李四什麼?

(P13 (ACT (M292 (LEX teach))) (AGENT B1) (EXPERIENCER B12) (OBJECT V2))

 What did Zhang1san1 teach Li3si4?

 CASSIE:  Zhang1san1 taught Li3si4 English.
```

```
 CASSIE:  Zhang1san1 taught Li3si4 Japanese.

 Time (sec.): 0.4

 : ;;;
;;; Shei2  jiao1  Li3 si4 Ying1 wen2?
;;;  who   teach  Li3si4  English
 誰教李四英文?

(P15 (ACT (M292 (LEX teach))) (AGENT V1) (EXPERIENCER B12) (OBJECT B14))

 Who taught Li3si4 English?

 CASSIE:  Zhang1san1 taught Li3si4 English.

 Time (sec.): 0.47

 : ;;;
;;; Li3 si4  Zhang1 san1  jiao1  Ying1 wen2.
;;; Li3si4   Zhang1san1   teach   English
 李四張三教英文.

(M303! (CLASS (M225 (LEX English))) (MEMBER B15))
(M304! (ACT (M292 (LEX teach))) (AGENT B1 B12) (OBJECT B15))

 Li3si4 and Zhang1san1 taught English.

 Time (sec.): 0.06

 : ;;;
;;; Li3 si4  jiao1  she2 mo5?
;;; Li3si4   teach    what
 李四教什麼?

(P16 (ACT (M292 (LEX teach))) (AGENT B12) (OBJECT V2))

 What did Li3si4 teach?

 CASSIE:  Li3si4 taught English.

 Time (sec.): 0.23

 : ;;; Four NPs juxtaposed without conjunctions or markers among them.
;;; Should be able to assign each of them a proper role.
;;; Zhang1 san1  jiao1  Li3 si4  Wang2 wu3 Guo2 hua2  Yu3 yang2 xue2.
;;; Zhang1san1   teach  Li3si4   Wang2wu3  Guo2hua2    Linguistics
 張三教李四王五國華語言學.
```

```
(M306! (CLASS (M231 (LEX linguistics))) (MEMBER B16))
(M307! (OBJECT B17) (PROPERNAME (M197 (LEX Guo2hua2))))
(M308! (OBJECT B18) (PROPERNAME (M177 (LEX Wang2wu3))))
(M309! (ACT (M292 (LEX teach))) (AGENT B1) (EXPERIENCER B12 B17 B18)
 (OBJECT B16))

 Zhang1san1 taught Li3si4, Wang2wu3 and Guo2hua2 linguistics.

 Time (sec.): 0.24

 : ;;; Should NOT answer:
;;; "Zhang1san1 taught Li3si4 Wang2wu3, Guo2hua2 and Linguistics."
;;; Zhang1 san1  jiao1  Li3 si4  she2 mo5?
;;; Zhang1san1   teach  Li3si4     what
 張三教李四什麼?

 What did Zhang1san1 teach Li3si4?

 CASSIE:  Zhang1san1 taught Li3si4 linguistics.

 CASSIE:  Zhang1san1 taught Li3si4 English.

 CASSIE:  Zhang1san1 taught Li3si4 Japanese.

 Time (sec.): 0.48

 : ;;; Transpose the NPs and leave out one NP from the previous input string.
;;; Showing true understanding. Not just surface string matching.
;;; Zhang1 san1  jiao1  Li3 si4  Guo2 hua2  she2 mo5?
;;; Zhang1san1   teach  Li3si4   Guo2hua2     what
 張三教國華李四什麼?

(P17 (ACT (M292 (LEX teach))) (AGENT B1) (EXPERIENCER B12 B17)
 (OBJECT V2))

 What did Zhang1san1 teach Guo2hua2 and Li3si4?

 CASSIE:  Zhang1san1 taught Guo2hua2 and Li3si4 linguistics.

 Time (sec.): 0.41

 : ;;; Chinese adjective is a state verb. No copula "be".
;;; Lao3 li3  gao1 xing4.
;;; Lao3li3     happy
 老李高興.

(M312! (OBJECT B19) (PROPERNAME (M171 (LEX Lao3li3))))
(M314! (OBJECT B19) (PROPERTY (M313 (LEX happy))))
```

```
 Lao3li3 is happy.

 Time (sec.): 0.04

 : ;;;
;;; Shei2  gao1 xing4?
;;;  who     happy
 誰高興?

(P18 (OBJECT V11) (PROPERTY (M313 (LEX happy))))

 Who is happy?

 CASSIE:  Lao3li3 is happy.

 Time (sec.): 0.14

 : ;;;                preposition  Locative postposition
;;; Shi2 tou2  diao4  dao4  zhe4 ge5 jing3  li3.
;;;   stone    fall    to   det  CL  well   in
 石頭掉到這個井裡.

(M315! (CLASS (M241 (LEX well))) (MEMBER B20))
(M319! (ADPOSITION (M318 (LEX into))) (OBJECT B20))
(M320! (CLASS (M213 (LEX rock))) (MEMBER B21))
(M323! (ACT (M322 (LEX fall))) (LOCATIVE B20) (OBJECT B21))

 Rocks fell into the well.

 Time (sec.): 0.42

 : ;;;
;;;                           prep           postposition
;;; Li3 si4  ba3  shu1   fan4  zai4  zhuo1    xia4.
;;; Li3si4   BA   book   put    at   table    under
 李四把書放在那張桌下.

(M324! (CLASS (M223 (LEX table))) (MEMBER B22))
(M327! (ADPOSITION (M326 (LEX under))) (OBJECT B22))
(M329! (CLASS (M215 (LEX book))) (MEMBER B24))
(M333! (ACT (M332 (LEX put))) (AGENT B12) (LOCATIVE B22) (OBJECT B24))

 Li3si4 put books under that table.

 Time (sec.): 0.46

 : ;;;
;;; Li3 si4  lai2  dao4  Tai2 bei3.
;;; Li3si4   come   to    Taipei
```

```
 李四來到台北.

(M334! (OBJECT B25) (PROPERNAME (M191 (LEX Taipei))))
(M336! (ADPOSITION (M335 (LEX to))) (OBJECT B25))
(M339! (ACT (M338 (LEX come))) (AGENT B12) (LOCATIVE B25))

 Li3si4 came to Taipei.

 Time (sec.): 0.44

 : ;;; One relative clause embedded in another.
;;; Both the NPs for agent and object contain relative clauses.
;;; The agent and experiencer roles for the verb, study, are coreferential.
;;; Zai4 yi1 ge5 yin2 hang2  zuo4  shi4  de5  Da4 hua2  xi3 huan1  nian4
;;;  in  one CL    bank       do   thing DE   Da4hua2     like     study
;;; yu3 yan2 xue2 de5  Guo2 hua2  mai3  de  shu1.
;;; Linguistics   DE   Guo2hua2   buy   DE  book
 在一家銀行做事的大華喜歡念語言學的國華買的書.

(M341! (CLASS (M231 (LEX linguistics))) (MEMBER B27))
(M343! (ACT (M342 (LEX study))) (AGENT B17) (EXPERIENCER B17)
 (OBJECT B27))
(M348! (OBJECT B28) (PROPERNAME (M195 (LEX Da4hua2))))
(M349! (CLASS (M243 (LEX work))) (MEMBER B29))
(M355! (CLASS (M209 (LEX bank))) (MEMBER B32))
(M358! (ADPOSITION (M357 (LEX in))) (OBJECT B32))
(M359! (ACT (M243)) (AGENT B28) (LOCATIVE B32) (OBJECT B29))
(M340! (CLASS (M215 (LEX book))) (MEMBER B26))
(M362
 (MAIN (M361! (ACT (M360 (LEX like))) (EXPERIENCER B28) (OBJECT B26)))
 (RELC-E (M359!))
 (RELC-O
  (M345! (ACT (M344 (LEX buy))) (AGENT B17) (BENEFACTIVE B17)
   (OBJECT B26))))
(M347 (MAIN (M345!))
 (RELC-B
  (M343! (ACT (M342 (LEX study))) (AGENT B17) (EXPERIENCER B17)
   (OBJECT B27))))

 Da4hua2 who worked in a bank liked the books
  which Guo2hua2 who studied linguistics bought.

 Time (sec.): 0.47

 : ;;; Relative clause combined with serial-verb construction may lead to
;;; the garden-path problem.
;;;
;;; Da4 hua2  xi3 huan1  nian4  yu3 yan2 xue2  de5  Guo2 hua2  mai3  de  shu1.
;;; Da4hua2     like     study   Linguistics   DE   Guo2hua2   buy   DE  book
```

```
;;; "Da4hua2 liked the books which Guo2hua2 who studied Linguistics bought."
;;;
;;; Da4 hua2  xi3 huan1  nian4  yu3 yan2 xue2  de5  Guo2 hua2.
;;; Da4hua2     like     study   Linguistics   DE   Guo2hua2
;;; "Da4hua2 liked Guo2hua2 who studied Linguistics."
;;;
;;; Da4 hua2  xi3 huan1  nian4  yu3 yan2 xue2.
;;; Da4hua2     like     study   Linguistic
;;; "Da4hua2 liked studying Linguistics."
;;; Take a segment of the previous input to test if CASSIE gets confused.
;;; Zai4 yi1 ge5 yin2 hang2 zuo4 shi4 de5 Da4 hua2
;;;  in  one CL     bank    do   thing DE Da4hua2
;;; xi3 huan1 nian4 yu3 yan2 xue2 ma1?
;;; like    study Linguistics   Q
 在一家銀行做事的大華喜歡念語言學嗎?

(M359! (ACT (M243 (LEX work))) (AGENT B28) (LOCATIVE B32) (OBJECT B29))
(M366 (ACT (M360 (LEX like))) (EXPERIENCER B28)
 (XCOMP
  (M363 (ACT (M342 (LEX study))) (AGENT B28) (EXPERIENCER B28)
   (OBJECT B27))))

 Did Da4hua2 who worked in a bank like to study linguistics?

 CASSIE:  I don't know.

 Time (sec.): 0.88

 : ;;; "Who did Da4hua2 who worked in a bank like?" Should answer "I don't know."
;;; Zai4  yi1  ge5  yin2 hang2  zuo4  shi4  de5  Da4 hua2  xi3 huan1  shei2?
;;;  in   one  CL     bank       do   thing DE   Da4hua2     like      who
 在一家銀行做事的大華喜歡誰?

(P28 (ACT (M360 (LEX like))) (EXPERIENCER B28) (OBJECT V1))

 Whom did Da4hua2 who worked in a bank like?

 CASSIE:  I don't know.

 Time (sec.): 1.04

 : ;;;
;;; Zai4 yi1 ge5 yin2 hang2 zuo4 shi4  de5 Da4 hua2 xi3 huan1 she2 mo5?
;;; in   one CL    bank     do   thing DE  Da4hua2    like       what
 在一家銀行做事的大華喜歡什麼?

(P34 (ACT (M360 (LEX like))) (EXPERIENCER B28) (OBJECT V2))

 What did Da4hua2 who worked in a bank like?
```

```
CASSIE:  Da4hua2 who worked in a bank liked the books
         which Guo2hua2 who studied linguistics bought.

Time (sec.): 1.45

 : ;;;
;;; Nian4 yu3 yan2 xue2 de5 Guo2 hua2 xi3 huan1 nian4 yu3 yan2 xue2 ma1?
;;; study Linguistics   DE  Guo2hua2    like    study Linguistics   Q
念語言學的國華喜歡念語言學嗎?

(M343! (ACT (M342 (LEX study))) (AGENT B17) (EXPERIENCER B17) (OBJECT B27))
(M343! (ACT (M342 (LEX study))) (AGENT B17) (EXPERIENCER B17) (OBJECT B27))
(M370 (ACT (M360 (LEX like))) (EXPERIENCER B17) (XCOMP (M343!)))

Did Guo2hua2 who studied linguistics like studying linguistics?

CASSIE:  I don't know.

Time (sec.): 0.49

 : ;;;
;;; Zai4 yi1 ge5 yin2 hang2 zuo4 shi4  de5 Da4 hua2 xi3 huan1 Mei3 hua2.
;;; in   one CL    bank     do   thing DE  Da4hua2    like    Mei3hua2
在一家銀行做事的大華喜歡美華.

(M373! (CLASS (M243 (LEX work))) (MEMBER B34))
(M378! (CLASS (M209 (LEX bank))) (MEMBER B37))
(M380! (ADPOSITION (M357 (LEX in))) (OBJECT B37))
(M381! (ACT (M243)) (AGENT B28) (LOCATIVE B37) (OBJECT B34))
(M372! (OBJECT B33) (PROPERNAME (M199 (LEX Mei3hua2))))
(M383
 (MAIN (M382! (ACT (M360 (LEX like))) (EXPERIENCER B28) (OBJECT B33)))
 (RELC-E (M381!)))

Da4hua2 who worked in a bank liked Mei3hua2.

Time (sec.): 0.3

 : ;;; The answer shouldn't include "books".
;;; The inference returns "books and Mei3hua2" but "books" is filtered out.
在一家銀行做事的大華喜歡誰?

Whom did Da4hua2 who worked in a bank like?

CASSIE:  Da4hua2 who worked in a bank liked Mei3hua2.

Time (sec.): 2.08
```

```
 : ;;; Showing adjective ordering in English generation.
;;; Two DEs: the DE as relative clause marker and the DE as adjective marker.
;;; Jiao1 Li3 si4 Ying1 wen2 de5 Lao3 zhang1 xi3 huan1 Wang2 wu3 mai3 de5
;;; teach Li3si4  English    DE  Lao3zhang1  like      Wang2wu3  buy  DE
;;;
;;; xin1  de  hong2 se4  da4  Zhang1  mu4 zhi4  yuan2  zuo1.
;;; new   DE     red     big    CL     wooden   round  table
 教李四英文的老張喜歡王五買的新的紅色大張木製圓桌.

(M396! (OBJECT B39) (PROPERNAME (M175 (LEX Lao3zhang1))))
(M397! (CLASS (M225 (LEX English))) (MEMBER B40))
(M398! (ACT (M292 (LEX teach))) (AGENT B39) (EXPERIENCER B12)
 (OBJECT B40))
(M384! (CLASS (M223 (LEX table))) (MEMBER B38))
(M386! (OBJECT B38) (PROPERTY (M385 (LEX round))))
(M388! (OBJECT B38) (PROPERTY (M387 (LEX wooden))))
(M390! (OBJECT B38) (PROPERTY (M389 (LEX big))))
(M392! (OBJECT B38) (PROPERTY (M391 (LEX red))))
(M394! (OBJECT B38) (PROPERTY (M393 (LEX new))))
(M400
 (MAIN (M399! (ACT (M360 (LEX like))) (EXPERIENCER B39) (OBJECT B38)))
 (RELC-E (M398!))
 (RELC-O
  (M395! (ACT (M344 (LEX buy))) (AGENT B18) (BENEFACTIVE B18)
    (OBJECT B38))))

 Lao3zhang1 who taught Li3si4 English liked
 the big new round red wooden tables which Wang2wu3 bought.

 Time (sec.): 0.4

 : ^^

 Enter Lisp Read/Eval/Print loop. Type ^^ to continue

--> ;;; The node type can determine the mood of a sentence.
;;; Surface an unasserted molecule node, we get a Yes-No question
;;; because CASSIE has doubts about it.
;;; "Did Da4hua2 who worked in a bank like to study Linguistics?"
(SURFACE
 (full-describe
  (find act (build lex like)
        experiencer (find (object- propername lex) Da4hua2) = D
        xcomp (find act (build lex study) agent (* 'D) experiencer (* 'D))))
 = unasserted)

(M366 (ACT (M360 (LEX like))) (EXPERIENCER B28)
 (XCOMP
  (M363 (ACT (M342 (LEX study))) (AGENT B28) (EXPERIENCER B28)
```

```
   (OBJECT B27))))
 Did Da4hua2 who worked in a bank like to study linguistics?

--> ;;; To assert an unasserted molecule node is to make her believe it.
;;; So now we will get a declarative sentence.
(SURFACE (describe (! (* 'unasserted))))

(M366! (ACT (M360 (LEX like))) (EXPERIENCER B28)
 (XCOMP
  (M363 (ACT (M342 (LEX study))) (AGENT B28) (EXPERIENCER B28)
   (OBJECT B27))))
 Da4hua2 who worked in a bank liked to study linguistics.

--> ;;; To surface a pattern node, we get a wh-question.
;;; The choice of wh-words is determined by
;;; the role the variable node plays in a semantic network.
;;; "WHO" is used when agent is a variable node.
#!((surface
    (full-describe
     (build act (build lex teach) Agent ~(* 'who)
            experiencer (find (object- propername lex) Li3si4)
            object ~(car (find (member- class lex) English)) = E))))

(P43 (ACT (M292 (LEX teach))) (AGENT V1) (EXPERIENCER B12) (OBJECT B10))
 Who taught Li3si4 English?
NIL
NIL
--> ;;; "WHAT" is used when the object role is a variable node and
;;; the selectional restriction of object role is non-human.
;;; "What did Da4hua2 who worked in a bank like?"
#!((surface
    (full-describe
     ((build act (build lex like) experiencer ~(* 'D) object ~(* 'what))))))

(P34 (ACT (M360 (LEX like))) (EXPERIENCER B28) (OBJECT V2))
 What did Da4hua2 who worked in a bank like?
NIL
NIL
--> ;;; "WHOM" is used when the syntactical object is a variable node and
;;; its role requires the human feature.
#!((surface
    (full-describe (build act (build lex teach)
                          Agent (find (object- propername lex) Zhang1san1)
                          experiencer ~(* 'who) object ~(* 'E)))))

(P44 (ACT (M292 (LEX teach))) (AGENT B1) (EXPERIENCER V1) (OBJECT B10))
 Whom did Zhang1san1 teach English?
NIL
NIL
```

```
--> ^^

 : ;;; Passive voice
;;; Wan3 da3po4 le5.
;;; bowl break  LE
 碗打破了.

(M401! (CLASS (M256 (LEX bowl))) (MEMBER B41))
(M403! (ACT (M402 (LEX break) (TENSE PAST))) (AGENT B42) (OBJECT B41))

 Bowls were broken.

 Time (sec.): 0.09

 : ^^

 Enter Lisp Read/Eval/Print loop. Type ^^ to continue

--> ;;; The voice of a sentence can be affected by the node structure.
#!((surface (describe ~(car (find act (build lex read tense past) agent B1))
                   = active)))

(M278! (ACT (M277 (LEX read) (TENSE PAST))) (AGENT B1) (OBJECT B4))
 Zhang1san1 read that book.
NIL
NIL
--> ;;; By taking away the node for the agent role
;;; an active sentence will turned into a passive sentence.
(erase (describe (find (propername lex) Zhang1san1)))

(M272! (OBJECT B1) (PROPERNAME (M173 (LEX Zhang1san1))))
(M272 OBJECT (B1) PROPERNAME (M173))
node dismantled.

--> #!((surface (describe ~(* 'active))))

(M278! (ACT (M277 (LEX read) (TENSE PAST))) (AGENT B1) (OBJECT B4))
 That book was read.
NIL
NIL
--> ^^

 : ;;; Parsing sentential complement
;;; Zhang1 san1 xiang1 xin4 Li3 si4 xi3 huan1 Lao3 wang2 mai3 de shu1.
;;; Zhang1san1    believe   Li3si4  like      Lao3wang2  buy  DE book
 張三相信李四喜歡老王買的書.

(M404! (CLASS (M215 (LEX book))) (MEMBER B43))
(M405! (OBJECT B44) (PROPERNAME (M179 (LEX Lao3wang2))))
```

```
(M409! (OBJECT B45) (PROPERNAME (M173 (LEX Zhang1san1))))
(M408
 (MAIN (M407 (ACT (M360 (LEX like))) (EXPERIENCER B12) (OBJECT B43)))
 (RELC-O
  (M406! (ACT (M344 (LEX buy))) (AGENT B44) (BENEFACTIVE B44)
   (OBJECT B43))))
(M411! (ACT (M410 (LEX believe))) (AGENT B45) (COMP (M407)))

 Zhang1san1 believed that Li3si4 liked the books which Lao3wang2 bought.

 Time (sec.): 0.3

 : ;;; Sentential complement is not asserted.
;;; Li3 si4  xi3 huan1  Lao3 wang2  mai3  de  shu1  ma1?
;;; Li3si4     like     Lao3wang2   buy   DE  book  Q
 李四喜歡老王買的書嗎?

(M406! (ACT (M344 (LEX buy))) (AGENT B44) (BENEFACTIVE B44)
 (OBJECT B43))
(M407 (ACT (M360 (LEX like))) (EXPERIENCER B12) (OBJECT B43))

 Did Li3si4 like the books which Lao3wang2 bought?

 CASSIE:  I don't know.

 Time (sec.): 0.57

 : ;;; Relative clauses inside the sentential complement are asserted though.
;;; Lao3 wang2  mai3  shu1  ma1?
;;; Lao3wang2   buy   book  Q
 老王買書嗎?

 Did Lao3wang2 buy books?

 CASSIE:  Yes, Lao3wang2 bought books.

 Time (sec.): 0.53

 : ;;; The whole sentence is also asserted.
;;; Zhang1 san1 xiang1 xin4 Li3 si4 xi3 huan1 Lao3 wang2 mai3 de shu1 ma1?
;;; Zhang1san1    believe   Li3si4  like      Lao3wang2  buy  DE book Q
 張三相信李四喜歡老王買的書嗎?

 Did Zhang1san1 believe that Li3si4 liked the books which Lao3wang2 bought?

 CASSIE:  Yes, Zhang1san1 believed that
          Li3si4 liked the books which Lao3wang2 bought.

 Time (sec.): 0.76
```

```
 : ;;; Handle two-way branching
;;; Mixed deep right-branching (Scomp and Xcomp) and
;;; left-branching (relative clause).
;;; Zhang1 san1 xiang1 xin4 Mei3 hua2 xiang1 xin4
;;; Zhang1san1    believe   Mei3hua2    believe
;;; Wang2 wu3 quan4   Li3 si4 mai3 Lao3 wang2 mai3 de shu1.
;;; Wang2wu3  persude Li3si4  buy  Lao3wang2  buy  DE book
 張三相信美華相信王五勸李四買老王買的書.

(M412! (CLASS (M215 (LEX book))) (MEMBER B46))
(M413! (ACT (M344 (LEX buy))) (AGENT B44) (BENEFACTIVE B44) (OBJECT B46))
(M420
 (MAIN
  (M418 (ACT (M417 (LEX persuade))) (AGENT B18) (OBJECT B12)
   (XCOMP
    (M414 (ACT (M344 (LEX buy))) (AGENT B12) (BENEFACTIVE B12)
     (OBJECT B46)))))
 (RELC-O
  (M413! (ACT (M344)) (AGENT B44) (BENEFACTIVE B44) (OBJECT B46))))
(M422! (ACT (M410 (LEX believe))) (AGENT B45)
 (COMP (M421 (ACT (M410)) (AGENT B33) (COMP (M418)))))

 Zhang1san1 believed that Mei3hua2 believed that
 Wang2wu3 persuaded Li3si4 to buy the books which Lao3wang2 bought.

 Time (sec.): 0.46

 : ;;; Serial verb construction:
;;; A sentence containing two or more VPs or clauses.
;;; No syntactical markers to indicate the relationship among them.
;;; Noun Verb     Noun   Verb Noun
;;; ==============================
;;; John eat      dinner read book (Two independent events)
;;; John believe  Mary   read book (Sentential complement)
;;; John persuade Mary   read book (Object control)
;;; John promise  Mary   read book (Subject control)
;;; John buy      book   give Mary (Intention)
;;;
;;; Zhang1 san1  chi1  wan3 fan4  du2  shu1  shui4 jiao4.
;;; Zhang1san1   eat    dinner    read book    sleep
 張三吃晚飯讀書睡覺.

(M424! (CLASS (M270 (LEX dinner))) (MEMBER B48))
(M426! (ACT (M425 (LEX eat))) (AGENT B45) (OBJECT B48))
(M423! (CLASS (M215 (LEX book))) (MEMBER B47))
(M427! (ACT (M285 (LEX read))) (AGENT B45) (OBJECT B47))
(M429! (ACT (M428 (LEX sleep))) (AGENT B45))
```

```
 Zhang1san1 ate dinners, read books and slept.

 Time (sec.): 0.32

 : ;;; Object control:
;;; The controller of Xcomp's subject is the object of the matrix verb.
;;; The Xcomp is a non-finite clause without an overt subject.
;;; The agent and experiencer of the verb 'study' are coreferential.
;;; Zhang1 san1  quan4    Li3 si4  nian4  yu3 yan2 xue2.
;;; Zhang1san1   pursuade Li3si4   study  Linguistics
;;;
;;; f-structure in Lexical Functional Grammar (LFG)
;;; SUBJ  PRED 'Zhang1san1'
;;;       PERS  3
;;;       NUM  SING
;;; PRED  'PERSUADE<(SUBJ)(OBJ)(XCOMP)>'
;;; OBJ   PRED 'Li3si4'
;;;       PERS  3
;;;       NUM  SING
;;; XCOMP SUBJ [^OBJ]
;;;       PRED 'STUDY<(SUBJ)(OBJ)>'
;;;       OBJ  PRED 'Linguistics'
;;;            PERS  3
;;;            NUM   SING
 張三勸李四念語言學.

(M430! (CLASS (M231 (LEX linguistics))) (MEMBER B50))
(M433! (ACT (M417 (LEX persuade))) (AGENT B45) (OBJECT B12)
 (XCOMP
  (M431 (ACT (M342 (LEX study))) (AGENT B12) (EXPERIENCER B12)
   (OBJECT B50))))

 Zhang1san1 persuaded Li3si4 to study linguistics.

 Time (sec.): 0.21

 : ;;;
;;; Shei2  jiang1 yao4  nian4  yu3 yan2 xue2?
;;;  who       will     study   Linguistics
 誰將要念語言學?

(P49 (ACT (M364 (LEX study) (TENSE FUTURE))) (AGENT V1)
 (EXPERIENCER V1) (OBJECT B50))

 Who will study linguistics?

 CASSIE:  Li3si4 will study linguistics.

 Time (sec.): 0.97
```

```
 : ;;; Subject control:
;;; The subject of the matrix verb controls the subject of the Xcomp.
;;; In LFG, the Chinese verb  許諾 "promise"
;;; subcategorizes for a subject, an object and an Xcomp.
;;; Zhang1 san1  xu3 nuo4 Li3 si4  nian4  yu3 yan2 xue2.
;;; Zhang1san1   promise  Li3si4   study  Linguistics
 張三許諾李四念語言學.

(M435! (CLASS (M231 (LEX linguistics))) (MEMBER B51))
(M439! (ACT (M438 (LEX promise))) (AGENT B45) (OBJECT B12)
 (XCOMP
  (M436 (ACT (M342 (LEX study))) (AGENT B45) (EXPERIENCER B45)
   (OBJECT B51))))

 Zhang1san1 promised Li3si4 to study linguistics.

 Time (sec.): 0.23

 : ;;; Both comp and Xcomp (infinitive clauses) are not asserted.
;;; Li3 si4  qi4 tu2  qiang3 yi1 jia1  yin2 hang2.
;;; Li3si4   attempt  rob    one  CL     bank
 李四企圖搶一家銀行.

(M440! (CLASS (M209 (LEX bank))) (MEMBER B52))
(M447! (ACT (M446 (LEX attempt))) (AGENT B12)
 (XCOMP (M443 (ACT (M442 (LEX rob))) (AGENT B12) (OBJECT B52))))

 Li3si4 attempted to rob a bank.

 Time (sec.): 0.23

 : ;;;
;;; Zhang1 san1 tou1  Li3 si4  qian2  yong4.
;;; Zhang1san1  steal Li3si4   money   use
 張三偷李四錢用.

(M449! (CLASS (M448 (LEX money))) (MEMBER B53))
(M455! (ACT (M454 (LEX steal))) (AGENT B45) (BENEFACTIVE B12)
 (INTENT (M451 (ACT (M450 (LEX use))) (AGENT B45) (OBJECT B53)))
 (OBJECT B53))

 Zhang1san1 stole money from Li3si4 to use.

 Time (sec.): 0.27

 : ;;;
;;; Shei2 jiang1 yao4 yong4 qian2.
;;;  who     will      use  money
```

```
 誰將要用錢?

(P51 (ACT (M452 (LEX use) (TENSE FUTURE))) (AGENT V1) (OBJECT B53))

 Who will use money?

 CASSIE:  Zhang1san1 will use money.

 Time (sec.): 1.09

 : ;;;
;;; Zhang1 san1 gei3 Li3 si4 qian2 yong4.
;;; Zhang1san1  give Li3si4  money  use
 張三給李四錢用.

(M457! (CLASS (M448 (LEX money))) (MEMBER B54))
(M461! (ACT (M460 (LEX give))) (AGENT B45) (BENEFACTIVE B12)
 (INTENT (M458 (ACT (M450 (LEX use))) (AGENT B12) (OBJECT B54)))
 (OBJECT B54))

 Zhang1san1 gave money to Li3si4 to use.

 Time (sec.): 0.25

 : ;;;
;;; Shei2 jiang1 yao4 yong4 qian2.
;;;  who     will      use  money
 誰將要用錢?

(P53 (ACT (M452 (LEX use) (TENSE FUTURE))) (AGENT V1) (OBJECT B54))

 Who will use money?

 CASSIE:  Li3si4 will use money.

 Time (sec.): 1.2

 : ;;;
;;; Zhang1 san1  gei3  Li3 si4  yi1  ben3  shu1  du2.
;;; Zhang1san1   give  Li3si4   one   CL   book  read
 張三給李四一本書讀.

(M463! (CLASS (M215 (LEX book))) (MEMBER B55))
(M468! (ACT (M460 (LEX give))) (AGENT B45) (BENEFACTIVE B12)
 (INTENT (M465 (ACT (M285 (LEX read))) (AGENT B12) (OBJECT B55)))
 (OBJECT B55))

 Zhang1san1 gave a book to Li3si4 to read.
```

```
 Time (sec.): 0.29

 : ;;;
;;; Wang2 wu3  mai3  shu1  song4 Li3 si4.
;;; Wnag1wu3   buy   book  give  Li3si4
 王五買書送李四.

(M469! (CLASS (M215 (LEX book))) (MEMBER B56))
(M473! (ACT (M344 (LEX buy))) (AGENT B18) (BENEFACTIVE B18)
 (INTENT
  (M470 (ACT (M460 (LEX give))) (AGENT B18) (BENEFACTIVE B12)
   (OBJECT B56)))
 (OBJECT B56))

 Wang2wu3 bought books to give Li3si4.

 Time (sec.): 0.24

 : ;;;
;;; Shei2  song4  Li3 si4  shu1?
;;;  who   give   Li3si4   book
 誰送李四書?

(P55 (ACT (M460 (LEX give))) (AGENT V1) (BENEFACTIVE B12) (OBJECT B56))

 Who gave books to Li3si4?

 CASSIE:  Wang2wu3 gave books to Li3si4.

 Time (sec.): 2.05

 : ;;;
;;; Zhang1 san1  qu4  mai3  shu1.
;;; Zhang1san1   go   buy   book
 張三去買書.

(M474! (CLASS (M215 (LEX book))) (MEMBER B57))
(M478! (ACT (M477 (LEX go))) (AGENT B45)
 (INTENT
  (M475 (ACT (M344 (LEX buy))) (AGENT B45) (BENEFACTIVE B45)
   (OBJECT B57))))

 Zhang1san1 went to buy books.

 Time (sec.): 0.23

 : ;;; A series of three verb phrases in a row; deep right-branching
;;; Li3 si4  mai3  shu1  gei3  Zhang1 san1  du2.
;;; Li3si4   buy   book  give  Zhang1san1  read
```

```
 李四買書給張三讀.

(M479! (CLASS (M215 (LEX book))) (MEMBER B58))
(M484! (ACT (M344 (LEX buy))) (AGENT B12) (BENEFACTIVE B12)
 (INTENT
  (M482 (ACT (M460 (LEX give))) (AGENT B12) (BENEFACTIVE B45)
   (INTENT (M480 (ACT (M285 (LEX read))) (AGENT B45) (OBJECT B58)))
   (OBJECT B58)))
 (OBJECT B58))

 Li3si4 bought books to give Zhang1san1 to read.

 Time (sec.): 0.34

 : ;;;
;;; Zhang1 san1 du2   she2 mo5?
;;; Zhang1san1  read    what
 張三讀什麼?

(P56 (ACT (M285 (LEX read))) (AGENT B45) (OBJECT V2))

 What did Zhang1san1 read?

 CASSIE:  Zhang1san1 read books.

 CASSIE:  Zhang1san1 read books.

 Time (sec.): 1.06

 : ;;;
;;; Li3 si4  gei3  shei2  shu1
;;; Li3si4   give   who   book
 李四給誰書?

(P58 (ACT (M460 (LEX give))) (AGENT B12) (BENEFACTIVE V1) (OBJECT B58))

 Whom did Li3si4 give books to?

 CASSIE:  Li3si4 gave books to Zhang1san1.

 Time (sec.): 2.7

 : ;;; The agent and benefactive roles of the verb 'buy' are coreferential.
;;; Benefactive role does not surface.
;;; Wan2 wu3  mai3  shu1  du2.
;;; Wan2wu3   buy   book  read
 王五買書讀.

(M486! (CLASS (M215 (LEX book))) (MEMBER B59))
```

```
(M489! (ACT (M344 (LEX buy))) (AGENT B18) (BENEFACTIVE B18)
 (INTENT (M487 (ACT (M285 (LEX read))) (AGENT B18) (OBJECT B59)))
 (OBJECT B59))

 Wang2wu3 bought books to read.

 Time (sec.): 0.25

 : ;;; Agent and Benefactive of the verb 'adorn' are also coreferential.
;;; Benefactive role surfaces in English but does not surface in Chinese.
;;; Mei3 hua2 shen1 shi4 zhu1 bao3.
;;; Mei3hua2    adorn     jewels
 美華身飾珠寶.

(M490! (CLASS (M268 (LEX jewel))) (MEMBER B61))
(M492! (ACT (M491 (LEX adorn))) (AGENT B33) (BENEFACTIVE B33)
 (OBJECT B61))

 Mei3hua2 adorned herself with jewels.

 Time (sec.): 0.18

 : ;;;
;;; Li3 si4  yan4 wu4  Zhang1san1 tou1  qian2.
;;; Li3si4   dislike   Zhang1san1 steal momey
 李四厭惡張三偷錢.

(M493! (CLASS (M448 (LEX money))) (MEMBER B62))
(M498! (ACT (M497 (LEX dislike))) (EXPERIENCER B12) (OBJECT B45)
 (XCOMP (M494 (ACT (M454 (LEX steal))) (AGENT B45) (OBJECT B62))))

 Li3si4 disliked Zhang1san1's stealing money.

 Time (sec.): 0.27

 : ;;;
;;; Li3 si4  jian1 chi2  yao4  fu4  zhang4.
;;; Li3si4     insist    want  pay   bill
 李四堅持要付帳.

(M499! (CLASS (M258 (LEX bill))) (MEMBER B63))
(M505! (ACT (M504 (LEX insist))) (AGENT B12)
 (XCOMP (M501 (ACT (M500 (LEX pay))) (AGENT B12) (OBJECT B63))))

 Li3si4 insisted on paying bills.

 Time (sec.): 0.3

 : ;;;
```

```
;;; Jun1 guan1  jian1 chi2  yao4  shi4 bing1  fu2 cong2  ming4 ling4.
;;;   officer     insist    want   soldier      obey        order
 軍官堅持要士兵服從命令.

(M506! (CLASS (M264 (LEX order))) (MEMBER B65))
(M507! (CLASS (M262 (LEX soldier))) (MEMBER B66))
(M508! (CLASS (M260 (LEX officer))) (MEMBER B67))
(M513! (ACT (M504 (LEX insist))) (AGENT B67) (OBJECT B66)
 (XCOMP (M510 (ACT (M509 (LEX obey))) (AGENT B66) (OBJECT B65))))

 Officers insisted on soldier's obeying orders.

 Time (sec.): 0.39

 : ;;;
;;; Li3 si4  kao3 lu4  huan4  gong1 zuo4.
;;; Li3si4   consider  change     job
 李四考慮換工作.

(M514! (CLASS (M245 (LEX job))) (MEMBER B68))
(M520! (ACT (M519 (LEX consider))) (EXPERIENCER B12)
 (XCOMP (M516 (ACT (M515 (LEX change))) (AGENT B12) (OBJECT B68))))

 Li3si4 considered changing jobs.

 Time (sec.): 0.29

 : ;;;
;;; Li3 si4 jue2 ding4 huan4  gong1 zuo4.
;;; Li3si4    decide   change     job
 李四決定換工作.

(M521! (CLASS (M245 (LEX job))) (MEMBER B69))
(M525! (ACT (M524 (LEX decide))) (EXPERIENCER B12)
 (XCOMP (M522 (ACT (M515 (LEX change))) (AGENT B12) (OBJECT B69))))

 Li3si4 decided to change jobs.

 Time (sec.): 0.3

 : ;;;
;;; Li3 si4  gong1 xi3    Zhang1 san1 de2 dao4 gong1 zuo4.
;;; Li3si4   congratulate Zhang1san1    get        job
 李四恭喜張三得到工作.

(M526! (CLASS (M245 (LEX job))) (MEMBER B70))
(M532! (ACT (M531 (LEX congratulate))) (AGENT B12) (BENEFACTIVE B45)
 (XCOMP (M528 (ACT (M527 (LEX get))) (BENEFACTIVE B45) (OBJECT B70))))
```

```
 Li3si4 congratulated Zhang1san1 on getting jobs.

 Time (sec.): 0.31

 : ;;;
;;; Li3 si4  jian4 yi4  Zhang1 san1 nian4  yu3 yan2 xue2.
;;; Li3si3    suggest   Zhang1san1  study   Linguistics
 李四建議張三念語言學.

(M533! (CLASS (M231 (LEX linguistics))) (MEMBER B71))
(M537! (ACT (M536 (LEX suggest))) (AGENT B12) (BENEFACTIVE B45)
 (XCOMP
  (M534 (ACT (M342 (LEX study))) (AGENT B45) (EXPERIENCER B45)
   (OBJECT B71))))

 Li3si4 suggested that Zhang1san1 study linguistics.

 Time (sec.): 0.3

 : ;;;
;;; Li3si4 jian4 Zhang1 san1 qi2  yi1 tai2 dan1 che1.
;;; Li3si4  see  Zhang1san1  ride one  CL   bicycle
 李四見張三騎一台單車.

(M538! (CLASS (M266 (LEX bicycle))) (MEMBER B72))
(M545! (ACT (M544 (LEX see))) (EXPERIENCER B12) (OBJECT B45)
 (XCOMP (M541 (ACT (M540 (LEX ride))) (AGENT B45) (OBJECT B72))))

 Li3si4 saw Zhang1san1 riding a bicycle.

 Time (sec.): 0.41

 : ;;;
;;; Li3 si4 ji4 de2  jian4 guo4 Zhang1 san1.
;;; Li3si4  remember  see  ASP  Zhang1san1
 李四記得見過張三.

(M548! (ACT (M547 (LEX remember))) (EXPERIENCER B12)
 (XCOMP (M546! (ACT (M544 (LEX see))) (EXPERIENCER B12) (OBJECT B45))))

 Li3si4 remembered seeing Zhang1san1.

 Time (sec.): 0.28

 : ;;;
;;; Li3 si4 jian4 Zhang1 san1  ma1?
;;; Li3si4   see  Zhang1san1   Q
 李四見張三嗎?
```

```
(M546! (ACT (M544 (LEX see))) (EXPERIENCER B12) (OBJECT B45))

 Did Li3si4 see Zhang1san1?

 CASSIE:  Yes, Li3si4 saw Zhang1san1.

 Time (sec.): 0.16

 : ;;;
;;; Li3 si4  da1ying4  Zhang1 san1  ji4 de2  qu4  mai3  shu1.
;;; Li3si4   promise   Zhang1san1   remember go   buy   book
 李四答應張三記得去買書.

(M549! (CLASS (M215 (LEX book))) (MEMBER B73))
(M554! (ACT (M438 (LEX promise))) (AGENT B12) (OBJECT B45)
 (XCOMP
  (M551 (ACT (M547 (LEX remember))) (EXPERIENCER B12)
   (XCOMP
    (M550 (ACT (M344 (LEX buy))) (AGENT B12) (BENEFACTIVE B12)
     (OBJECT B73))))))

 Li3si4 promised Zhang1san1 to remember to buy books.

 Time (sec.): 0.39

 : ;;;
;;; Shei2 mai3 shu1?
;;;  who  buy  book
 誰買書?

(P60 (ACT (M344 (LEX buy))) (AGENT V1) (BENEFACTIVE V1) (OBJECT B73))

 Who bought books?

 CASSIE:  I don't know.

 Time (sec.): 1.9

 : ^^

 Enter Lisp Read/Eval/Print loop. Type ^^ to continue

--> (setq *all-parses* t)
T
--> ^^

 : ;;; The following sentences are to demonstrate the parser's ability to
;;;  distinguish three kinds of possessive relations:
;;; 1. kinship 2. object possession 3. part-whole.
```

```
;;; and three kinds of DEs:
;;; 1. possessive marker 2. relative cluase marker 3. adjectival marker.
;;;
;;; There are two possible readings for this exmaple:
;;; 1 Li3si4 liked the sons who kissed Da4wei3.
;;; 2 Li3si4 liked to kiss Da4wei3's sons.
;;; Li3 si4  xi3 huan1  qin1  Da4 wei3  de5  er2 zi2.
;;; Li3si4     like     kiss  Da4wei3   DE     son
 李四喜歡親大偉的兒子.

(M555! (CLASS (M254 (LEX son))) (MEMBER B74))
(M556! (OBJECT B75) (PROPERNAME (M183 (LEX Da4wei3))))
(M560
 (MAIN (M559! (ACT (M360 (LEX like))) (EXPERIENCER B12) (OBJECT B74)))
 (RELC-O (M558! (ACT (M557 (LEX kiss))) (AGENT B74) (OBJECT B75))))

 Li3si4 liked the sons who kissed Da4wei3.

Try next parse? y

(M565! (ARG1 B74) (ARG2 B75) (KINSHIP (M254 (LEX son))))
(M564! (ACT (M360 (LEX like))) (EXPERIENCER B12)
 (XCOMP (M561 (ACT (M557 (LEX kiss))) (AGENT B12) (OBJECT B74))))

 Li3si4 liked to kiss Da4wei3's sons.

Try next parse? n

 Time (sec.): 0.85

 : ;;; Object possession:
;;; Syntactically, it could be read as:
;;; "Li3si4 liked the book which kissed Da4wei3."
;;; The parser rules out this reading by checking semantic roles.
;;; Correct reading: "Li3si4 liked to kiss Da4wei3's books."
;;; This sentence further illustrates why Chinese language processing
;;; should be knowledge-based or semantic-driven.
;;; Li3 si4  xi3 huan1  qin1  Da4 Wei3  de5  shu1.
;;; Li3si4     like     kiss   Da4wei3   DE   book
 李四喜歡親大偉的書.

(M571! (OBJECT B77) (POSSESSOR B75) (REL (M215 (LEX book))))
(M566! (CLASS (M215 (LEX book))) (MEMBER B77))
(M569! (ACT (M360 (LEX like))) (EXPERIENCER B12)
 (XCOMP (M567 (ACT (M557 (LEX kiss))) (AGENT B12) (OBJECT B77))))

 Li3si4 liked to kiss Da4wei3's books.

Try next parse? y
```

```
 Time (sec.): 1.2

 : ;;; Kinship relations mixed with object possessive relations.
;;; The parser discards the wrong reading:
;;; "Li3si4 liked the book which bought Bao3min2's son."
;;; Correct reading: "Li3si4 liked to buy Bao3miin2's son's books."
;;; Li3 si4 xi3 huan1 mai3 Bao3 min2 de5 er2 zi2 de5 shu1.
;;; Li3si4    like    buy  Bao3min2  DE    son   DE  book
 李四喜歡買保民的兒子的書.

(M580! (OBJECT B80) (POSSESSOR B74) (REL (M215 (LEX book))))
(M581! (ARG1 B74) (ARG2 B82) (KINSHIP (M254 (LEX son))))
(M573! (CLASS (M215 (LEX book))) (MEMBER B80))
(M579! (ACT (M360 (LEX like))) (EXPERIENCER B12)
 (XCOMP
  (M577 (ACT (M344 (LEX buy))) (AGENT B12) (BENEFACTIVE B12)
   (OBJECT B80))))

 Li3si4 liked to buy Bao3min2's son's books.

Try next parse? y

 Time (sec.): 2.07

 : ;;; There is ambiguity, when the relative marker and
;;; the possessive DEs mix together.
;;; Ambiguity over what the head of the relative clause is.
;;; Two parsers: 1. Jian4zheng4's sons who bought books.
;;;              2. son of Jian4zheng4 who bought books.
;;; Note: instead of being "Jian4zheng4 who bought books's son", we have
;;;                         "son of Jian4zheng4 who bought books."
;;; Li3 si4 xi3 huan1 mai3 shu1 de jian4 zheng4 de er2 zi2.
;;; Li3si4    like    buy   book DE Jian4zhang4  DE   son
 李四喜歡買書的建正的兒子.

(M586! (OBJECT B88) (PROPERNAME (M187 (LEX jian4zheng4))))
(M587! (CLASS (M215 (LEX book))) (MEMBER B89))
(M588! (ACT (M344 (LEX buy))) (AGENT B88) (BENEFACTIVE B88)
 (OBJECT B89))
(M597! (ARG1 B87) (ARG2 B88) (KINSHIP (M254 (LEX son))))
(M585! (CLASS (M254 (LEX son))) (MEMBER B87))
(M596! (ACT (M360 (LEX like))) (EXPERIENCER B12) (OBJECT B87))

 Li3si4 liked the sons of jian4zheng4 who bought books.

Try next parse? y

(M597! (ARG1 B87) (ARG2 B88) (KINSHIP (M254 (LEX son))))
```

```
(M598! (CLASS (M215 (LEX book))) (MEMBER B93))
(M600 (MAIN (M596!))
 (RELC-O
  (M599! (ACT (M344 (LEX buy))) (AGENT B87) (BENEFACTIVE B87)
   (OBJECT B93))))

 Li3si4 liked jian4zheng4's sons who bought books.

Try next parse? n

 Time (sec.): 12.98

 : ;;; The parser is able to distinguish among three kinds of DE:
;;; Adjectival, possessive and relative cluase marker all in one sentence.
;;; Two possible readings: son of zhang1san1 who studied chinese language.
;;;                        zhang1san1's son who studied chinese language.
;;; Li3 si4 xi3 huan1 nian4 Zhon1 guo2 de yu3 yan2 de xing1 hua2 de er2 zi3.
;;; Li3si4    like    study Chinese       language DE Xing1hua2  DE son
 李四喜歡念中國的語言的興華的兒子.

(M602! (OBJECT B95) (PROPERNAME (M193 (LEX xing1hua2))))
(M603! (CLASS (M229 (LEX language))) (MEMBER B96))
(M605! (OBJECT B96) (PROPERTY (M604 (LEX Chinese))))
(M606! (ACT (M342 (LEX study))) (AGENT B95) (EXPERIENCER B95)
 (OBJECT B96))
(M606! (ACT (M342 (LEX study))) (AGENT B95) (EXPERIENCER B95)
 (OBJECT B96))
(M627! (ARG1 B94) (ARG2 B95) (KINSHIP (M254 (LEX son))))
(M601! (CLASS (M254 (LEX son))) (MEMBER B94))
(M626! (ACT (M360 (LEX like))) (EXPERIENCER B12) (OBJECT B94))

 Li3si4 liked the sons of xing1hua2 who studied Chinese language.

Try next parse? y

(M627! (ARG1 B94) (ARG2 B95) (KINSHIP (M254 (LEX son))))

 Li3si4 liked the sons of xing1hua2 who studied Chinese language.

Try next parse? y

(M627! (ARG1 B94) (ARG2 B95) (KINSHIP (M254 (LEX son))))
(M614! (OBJECT B100) (PROPERTY (M604 (LEX Chinese))))
(M619! (OBJECT B104) (PROPERTY (M604)))
(M625! (OBJECT B108) (PROPERTY (M604)))
(M631! (CLASS (M229 (LEX language))) (MEMBER B112))
(M632! (OBJECT B112) (PROPERTY (M604)))
(M634 (MAIN (M626!))
 (RELC-O
```

```
  (M633! (ACT (M342 (LEX study))) (AGENT B94) (EXPERIENCER B94)
   (OBJECT B112))))

 Li3si4 liked xing1hua2's sons who studied Chinese language.

Try next parse? n

 Time (sec.): 6.27

 : ^^

 Enter Lisp Read/Eval/Print loop. Type ^^ to continue

--> (setq *all-parses* nil)
NIL
--> ^^

 : ;;; Kinship relationship in the dative construction.
;;; Li3 si4 gei3 wei3 jian4 de5 er2 zi2 yi1 ben3 shu1.
;;; Li3si4  give Wei3jian4  DE    son   one  CL  book
 李四給偉健的兒子一本書.

(M641! (ARG1 B116) (ARG2 B115) (KINSHIP (M254 (LEX son))))
(M635! (CLASS (M215 (LEX book))) (MEMBER B113))
(M639! (CLASS (M254 (LEX son))) (MEMBER B116))
(M640! (ACT (M460 (LEX give))) (AGENT B12) (BENEFACTIVE B116)
 (OBJECT B113))

 Li3si4 gave a book to wei3jian4's sons.

 Time (sec.): 0.75

 : ;;; possessive relations
;;; Li3 si4 xi3 huan1 Zhang1 san1 de shu1.
;;; Li3si4    like    Zhang1san1  DE book
 李四喜歡張三的書.

(M644! (OBJECT B117) (POSSESSOR B45) (REL (M215 (LEX book))))
(M642! (CLASS (M215 (LEX book))) (MEMBER B117))
(M643! (ACT (M360 (LEX like))) (EXPERIENCER B12) (OBJECT B117))

 Li3si4 liked Zhang1san1's books.

 Time (sec.): 0.91

 : ;;; kinship and possessive relation; three levels of left-branching
;;; Li3si4 xi3 huan1 wei3 jian4 de5 er2 zi2 de5 shu1.
;;; Li3si4   like    Wei3jian4  DE    son    DE  book
 李四喜歡偉健的兒子的書.
```

```
(M652! (OBJECT B118) (POSSESSOR B116) (REL (M215 (LEX book))))
(M641! (ARG1 B116) (ARG2 B115) (KINSHIP (M254 (LEX son))))
(M645! (CLASS (M215 (LEX book))) (MEMBER B118))
(M651! (ACT (M360 (LEX like))) (EXPERIENCER B12) (OBJECT B118))

 Li3si4 liked wei3jian4's son's books.

 Time (sec.): 2.53

 : ;;; Ambiguity arsing from a sequence of nouns
;;; Two nouns side by side can be two different nouns or
;;; they can form one noun compound.
;;; Li3 si4 gei3 Zhi4 cheng2 Wang2 wu3 yi1 ben3 shu1.
;;; Li3si4  give Zhi4cheng2  Wang2wu3  one  CL  book
 李四給志成王五一本書.

(M653! (CLASS (M215 (LEX book))) (MEMBER B121))
(M655! (OBJECT B122) (PROPERNAME (M181 (LEX Zhi4cheng2))))
(M656! (ACT (M460 (LEX give))) (AGENT B12) (BENEFACTIVE B122 B18)
 (OBJECT B121))

 Li3si4 gave a book to Zhi4cheng2 and Wang2wu3.

 Time (sec.): 0.51

 : ^^

 Enter Lisp Read/Eval/Print loop. Type ^^ to continue

--> (setq *all-parses* t)
T
--> ^^

 : ;;; The structure is similar to the previous one;
;;; but, in this one, the second noun is a kinship term.
;;; Li3 si4 gei3  Zhi4 cheng2  er2 zi2 yi1 ben3 shu1.
;;; Li3si4  give  Zhi4cheng2     son   one  CL  book
 李四給志成兒子一本書.

(M657! (CLASS (M215 (LEX book))) (MEMBER B123))
(M660! (CLASS (M254 (LEX son))) (MEMBER B125))
(M661! (ACT (M460 (LEX give))) (AGENT B12) (BENEFACTIVE B122 B125)
 (OBJECT B123))

 Li3si4 gave a book to sons and Zhi4cheng2.

Try next parse? y
```

```
 Time (sec.): 0.75

 : ;;; The interlingual becomes a part of the knowledge base.
;;; Expand knowledge base while translating text.
;;; Zhi4 cheng2 you3 er2 zi2.
;;; Zhi4cheng2  have   son
 志成有兒子.

(M666! (ARG1 B128) (ARG2 B122) (KINSHIP (M254 (LEX son))))

 Zhi4cheng2 has sons.

Try next parse? n

 Time (sec.): 0.15

 : ;;; The compound noun is turned into the kinship relationship.
;;; Therefore, there are two parses now.
;;; Li3 si4 gei3 Zhi4 cheng2  er2 zi2 yi1 ben3 shu1.
;;; Li3si4  give Zhi4cheng2     son   one  CL  book
 李四給志成兒子一本書.

(M667! (CLASS (M215 (LEX book))) (MEMBER B129))
(M670! (CLASS (M254 (LEX son))) (MEMBER B131))
(M671! (ACT (M460 (LEX give))) (AGENT B12) (BENEFACTIVE B122 B131)
 (OBJECT B129))

 Li3si4 gave a book to sons and Zhi4cheng2.

Try next parse? y

(M665! (CLASS (M254 (LEX son))) (MEMBER B128))
(M672! (ACT (M460 (LEX give))) (AGENT B12) (BENEFACTIVE B128)
 (OBJECT B129))

 Li3si4 gave a book to Zhi4cheng2's sons.

Try next parse? n

 Time (sec.): 0.91

 : ^^

 Enter Lisp Read/Eval/Print loop. Type ^^ to continue

--> (setq *all-parses* nil)
NIL
--> ^^
```

```
 : ;;; A serial of four nouns without syntactical markers to link them.
;;; The parser is able to assign a proper role to each of them.
;;; Li3 si4 gei3 Wang2 wu3 Zhang1 san1 dian4 shi4 bing1 xiang1.
;;; Li3si4  give Wang2wu3  Zhang1san1  television refrigerator
 李四給王五張三電視冰箱.

(M673! (CLASS (M239 (LEX refrigerator))) (MEMBER B132))
(M674! (CLASS (M233 (LEX television))) (MEMBER B133))
(M675! (ACT (M460 (LEX give))) (AGENT B12) (BENEFACTIVE B18 B45)
 (OBJECT B132 B133))

 Li3si4 gave televisions and refrigerators to Wang2wu3 and Zhang1san1.

 Time (sec.): 0.47

 : ;;; Nouns that can form compound should be semantically close.
;;; Both "Wang2wu3" and "Zhang1san1" from previous sentences are human, so
;;; they can form compound, same for "television" and "refrigerator".
;;; School (inanimate) and janitor (animate) are in different categories,
;;; so they can't form compound nouns; therefore, there is no parse.
;;; Zhang1 san1 gei3 nei4 wei4 xue2 xiao4 gong1 you3 dian4 shi4 tian1 xian4
;;; Zhang1san1  give Det   CL    school    janitor   television   antenna
 李四給那位學校工友電視天線.

 Time (sec.): 1.0

 : ;;; Part-whole relation.
;;; Expand knowledge base with natural language interface.
;;; Xue2 xiao4  you3  gong1 you3.
;;;   school    have   janitor
 學校有工友.

(M722! (PART (M211 (LEX janitor))) (WHOLE (M207 (LEX school))))

 Schools have janitors.

 Time (sec.): 0.17

 : ;;; The fact that the antenna is a television part
;;; is specified in the lexicon and is built as a semantic network
;;; at the initialization stage.
;;; The knowledge can be used for the inference now.
;;; Zhang1 san1 gei3 nei4 wei4 xue2 xiao4 gong1 you3 dian4 shi4 tian1 xian4
;;; Zhang1san1  give Det   CL    school    janitor   television   antenna
 李四給那位學校工友電視天線.

(M733! (PART B173) (WHOLE (M207 (LEX school))))
(M734! (PART B168) (WHOLE (M233 (LEX television))))
(M723! (CLASS (M237 (LEX antenna))) (MEMBER B168))
```

```
(M730! (CLASS (M211 (LEX janitor))) (MEMBER B173))
(M732! (ACT (M460 (LEX give))) (AGENT B12) (BENEFACTIVE B173)
 (OBJECT B168))

 Li3si4 gave television's antennas to that school's janitor.

 Time (sec.): 1.21

 : ;;; A series of nouns appear as the head of a relative clause
;;; Zhang1 san1 xi3 huan1 Li3 si4 gei3 Wang2 wu3 de5 dian4 shi4 bin1 xian1.
;;; Zhang1san1    like    Li3si4  give Wang2wu3  DE  television refrigerator
 張三喜歡李四給王五的電視冰箱.

(M735! (CLASS (M239 (LEX refrigerator))) (MEMBER B174))
(M736! (CLASS (M233 (LEX television))) (MEMBER B175))
(M739
 (MAIN
  (M738! (ACT (M360 (LEX like))) (EXPERIENCER B45) (OBJECT B174 B175)))
 (RELC-O
  (M737! (ACT (M460 (LEX give))) (AGENT B12) (BENEFACTIVE B18)
   (OBJECT B174 B175))))

 Zhang1san1 liked the televisions and the refrigerators
 which Li3si4 gave Wang2wu3.

 Time (sec.): 0.68

 : ;;;
;;; Jin1 si1 que4 fei1.
;;;    cannary    fly
 金絲雀飛.

(M740! (CLASS (M250 (LEX canary))) (MEMBER B176))
(M742! (ACT (M167 (LEX fly))) (AGENT B176))

 Canaries flew.

 Time (sec.): 0.34

 : ;;; That birds can fly and canaries are birds is defined in the lexicon.
;;; 'Canary' inherits the ability from its superclass, bird.
;;; Therefore, the parser infers that a canary can fly.
;;; Jin1 si1 que4 neng2 fei1 ma1?
;;;    canary     can  fly  Q
 金絲雀能飛嗎?

(M743 (ABILITY (M167 (LEX fly))) (HAS-ABILITY (M250 (LEX canary))))

 Can a canary fly?
```

```
 CASSIE:  Yes, a canary can fly.

 Time (sec.): 0.37

 : ;;;
;;; Tweety neng2 fei1 ma1?
;;; Tweety can   fly  Q
 甜啼能飛嗎?

(M746! (OBJECT B177) (PROPERNAME (M745 (LEX Tweety))))
(M747 (ABILITY (M167 (LEX fly))) (HAS-ABILITY B177))

 Can Tweety fly?

 CASSIE:  I don't know.

 Time (sec.): 0.22

 : ;;;
;;; Tweety shi4 yi1 zhi1 jin1 si1 que4.
;;; Tweety is   one CL   cannary
 甜啼是一隻金絲雀.

(M750! (CLASS (M250 (LEX canary))) (MEMBER B177))

 Tweety is a canary.

 Time (sec.): 0.22

 : ;;; Tweety, a member of the canary, inherits ability from its class.
;;; Tweety neng2 fei1 ma1?
;;; Tweety  can  fly  Q
 甜啼能飛嗎?

 Can Tweety fly?

 CASSIE:  Yes, Tweety can fly.

 Time (sec.): 0.15

 : ;;;
;;; Da4 xiang4 neng2 fei1 ma1?
;;; elephant   can   fly  Q
 大象能飛嗎?

(M752 (ABILITY (M167 (LEX fly))) (HAS-ABILITY (M252 (LEX elephant))))

 Can an elephant fly?
```

```
 CASSIE:  I don't know.

 Time (sec.): 0.3

 : ;;;
;;; Dumbo shi4 yi1 zhi1 Da4 xiang4.
;;; Dumbo  is  one  CL   elephant
 丹寶是一隻大象.

(M757! (OBJECT B180) (PROPERNAME (M756 (LEX Dumbo))))
(M758! (CLASS (M252 (LEX elephant))) (MEMBER B180))

 Dumbo is an elephant.

 Time (sec.): 0.35

 : ;;; member-class relationship
;;; Clyde shi4 yi1 zhi1 Da4 xiang4.
;;; Clyde  is  one  CL   elephant
 卡來迪是一隻大象.

(M762! (OBJECT B182) (PROPERNAME (M761 (LEX Clyde))))
(M763! (CLASS (M252 (LEX elephant))) (MEMBER B182))

 Clyde is an elephant.

 Time (sec.): 0.36

 : ;;; Add more knowledge to the knowledge base.
;;; Dumbo neng2 fei1.
;;; Dumbo  can  fly
 丹寶能飛.

(M764 (ABILITY (M167 (LEX fly))) (HAS-ABILITY B180))

 Dumbo can fly.

 Time (sec.): 0.09

 : ;;; Observed ability can go up the inheritance hierarchy.
;;; Da4 xiang4 neng2 fei1 ma1?
;;;  elephant   can  fly  Q
 大象能飛嗎?

 Can an elephant fly?

 CASSIE:  Yes, an elephant can fly.
```

```
 Time (sec.): 0.49

 : ;;; Observed ability can only go up but not up then down the hierarchy.
;;; Clyde neng2 fei1 ma1?
;;; Clyde  can  fly  Q
 卡來迪能飛嗎?

(M766 (ABILITY (M167 (LEX fly))) (HAS-ABILITY B182))

 Can Clyde fly?

 CASSIE:  I don't know.

 Time (sec.): 0.13

 : ;;;
;;; Dumbo neng2 fei1 ma1?
;;; Dumbo  can  fly  Q
 丹寶能飛嗎?

 Can Dumbo fly?

 CASSIE:  Yes, Dumbo can fly.

 Time (sec.): 0.15

 : ^end


 ATN Parser exits...


--> ^^

 CPU time : 85.42

*

End of MTdemo demonstration.
```

Bibliography

[Allen, 1987] James Allen. *Natural Language Understanding*. Benjamin/Cummings, Menlo Park, CA, 1987.

[Chomsky, 1963] Norm Chomsky. Formal properties of grammars. In *Handbook of Math and Psych.*, volume 2. John Wiley and Sons, New York, NY, 1963.

[Cook, 1989] Walter A. Cook. *Case Grammar Theory*. Georgetown University Press, Washington, D. C., 1989.

[Greenberg, 1963] Joseph H. Greenberg. *Some Universals of Grammar with Particular Reference to the Order of Meaningful Elements*. MIT Press, Cambridge, MA, 1963.

[Kaplan and Bresnan, 1982] R. Kaplan and J. Bresnan. Lexical-functional grammar: A formal system for grammatical representation. In Joan Bresnan, editor, *The Mental Representation of Grammatical Relations*, pages 282–390. MIT Press, Cambridge, MA, 1982.

[Li and Thompson, 1981] C. Li and S. Thompson. *Mandarin Chinese: A Functional Reference Grammar*. University of California Press, Berkeley, CA, 1981.

[Martins and Shapiro, 1988] J. P. Martins and S. C. Shapiro. A model for belief revision. *Artificial Intelligence*, 35(1):25–79, 1988.

[Rapaport, 1988] W. J. Rapaport. Syntactic semantics: Foundations of computational natural-language understanding. In J. Fetzer, editor, *Aspects of Artificial Intelligence*, pages 81–131. Kluwer Academic Publishers, Dordrecht, Holland, 1988. Reprinted in E. Dietrich, editor, *Thinking Computers & Virtual Persons: Essays on the Intentionality of Machines,* pages 225–273. Academic Press, San Diego, 1994.

[Rapaport, 1991] William J. Rapaport. Predication, fiction, and artificial intelligence. *Topoi*, 10:79–111, 1991.

[Shapiro and Group, 1994] S. C. Shapiro and The SNePS Implementation Group. *SNePS 2.1 User's Manual*. Department of Computer Science, SUNY at Buffalo, 1994.

[Shapiro and Rapaport, 1987] S. C. Shapiro and W. J. Rapaport. SNePS considered as a fully intensional propositional semantic network. In N. Cercone and G. McCalla, editors, *The Knowledge Frontier*, pages 263–315. Springer–Verlag, New York, 1987.

[Shapiro and Rapaport, 1991] Stuart C. Shapiro and William J. Rapaport. Models and minds: Knowledge representation for natural-language competence. In Robert Cummins and John Pollock, editors, *Philosophy and AI: Essays at the Interface*, pages 215–259. MIT Press, Cambridge, MA, 1991.

[Shapiro and Rapaport, 1992] Stuart C. Shapiro and William J. Rapaport. The SNePS family. *Computers & Mathematics with Applications*, 23(2–5):243–275, January–March 1992.

[Shapiro, 1978] S. C. Shapiro. Path-based and node-based inference in semantic networks. In D. Waltz, editor, *Tinlap–2: Theoretical Issues in Natural Languages Processing*, pages 219–225, New York, 1978. ACM.

[Shapiro, 1979] S. C. Shapiro. The SNePS semantic network processing system. In N. V. Findler, editor, *Associative Networks: The Representation and Use of Knowledge by Computers*, pages 179–203. Academic Press, New York, 1979.

[Shapiro, 1982] S. C. Shapiro. Generalized augmented transition network grammars for generation from semantic networks. *The American Journal of Computational Linguistics*, 8(1):12–25, 1982.

[Shapiro, 1989] S. C. Shapiro. The CASSIE projects: An approach to natural language competence. In *Proceedings of the 4th Portugese Conference on Artificial Intelligence*, pages 362–380, Lisbon, Portugal, 1989. Springer-Verlag.

[Tang, 1989] T.-C Tang. *Studies on Chinese Morphology and Syntax:2*. Student Book Co., Ltd, Taipei, Taiwan, 1989.