2 COMPUTATIONAL MODELLING IN ARTIFICIAL INTELLIGENCE

Walter Daelemans and Koenraad de Smedt

Chapter prepared for:
De Smedt, K. (1996). CompJutional models of incremental grammatical encoding. In A. Dijkstra & K. de Smedt (Eds.) (1996). *Computational psycholinguistics: AI and connectionist models of human language processing* (pp. 24-48). London: Taylor & Francis, 1996.

2.1   Introduction

Artificial Intelligence (AI) is a branch of computer science in which methods and techniques are developed that permit intelligent computer systems to be built. These systems allow the simulation of different aspects of human and animal cognition, including perception, action, communication, problem solving, and learning. Whether these systems are really intelligent is a controversial issue which will not concern us here (see for example Copeland, 1993, for an overview). What is more important is that AI stimulated researchers in cognitive psychology to be more explicit in their theories about mental processes. By the very fact that AI enabled machines to somehow carry out linguistic and other cognitive tasks, researchers could no longer ignore the possibilities of computer models as a precise reflection of ideas about how humans carry out such tasks. Put briefly, AI instigated the use of a computational vocabulary and provided programming methods and tools for building working models of language processing and other aspects of cognition.

AI approaches to natural language are often referred to as *computational linguistics* (CL) or *natural language processing* (NLP) although not all computational linguists see their work as belonging to AI. For some computational linguists, the linguistic aspect is the most important: the goal of the implementation of a natural language processing program is not so much a simulation of human language processing, but rather as a test of linguistic theory with respect to its principles, consistency, correctness, and linguistic coverage. For other computational linguists, the computer science aspect is more important: natural language processing presents complex information processing problems and therefore the challenge is to arrive at efficient solutions to these problems by means of powerful mathematical and computer processing techniques.

From our point of view, the problem with some AI models of language processing is therefore that they do not seek to account for the characteristics of psycholinguistic processes in humans, but rather are meant to be in line with linguistic theory and with the state of the art in computer science. Moreover, AI researchers often also pursued the construction of working systems that can be applied to solving practical problems, for example spelling and grammar checking, machine translation, human-computer communication, etc. These systems, the products of *language technology* or *linguistic engineering,* are designed only for special practical purposes. They are based on designs that disguise fundamental problems in modelling aspects of language processing, and are therefore of no real psychological interest (Garnham, 1994).

But even if many AI models of language processing are not particularly suited or intended as psycholinguistic models, the large body of AI research is not to be dismissed as irrelevant for computational psycholinguistics for two reasons. First, even if AI researchers have somewhat different goals than psycholinguists have, they also have to face the same fundamental problems, so there is no reason why AI solutions should always be wrong or irrelevant from a psycholinguistic point of view. On the contrary, to a certain extent there has been some cross-fertilization between AI and psychology. Second, the rich repertoire of modelling techniques provided by AI has proven practical and useful in building

computational models of language processing that are of psychological interest, as witnessed in several chapters in Parts II and III of this volume. The aim of the current chapter is to support the readers' understanding of the models presented in those chapters by introducing the essentials of the AI methods that underlie those models.

While AI has assimilated many different computational methods including the connectionist approaches discussed in Chapter 3, the current chapter will be restricted to traditional or mainstream AI, which is based on symbol manipulation. The next section will explain this basis. Then the chapter will briefly introduce some of the architectures, modelling principles, and formalisms offered by AI. It concludes with a discussion of the shortcomings of AI and a brief guide to the literature.

2.2   Symbol manipulation

AI considers linguistic and other intelligent tasks as problems requiring the acquisition, representation and use of knowledge. Consequently, the use of knowledge representation and manipulation is central in AI. Much of the research in AI has led to theories of problem solving and knowledge representation in general, while language processing tasks such as the production and interpretation of utterances are often seen as particular instances of more general problem classes. The main research questions include the following. Which knowledge sources are necessary; which strategies should be used in processing; and how can all of this be represented and stored so as to be executed by a computer? The answers of traditional AI are based on symbolic representations of knowledge, and on processing as symbol manipulation.

The fact that computers can be seen as general symbol manipulators and not just number crunchers is due to Allen Newell, for which he received the 1975 Turing Award. With Herbert Simon, he formulated the *Physical Symbol System Hypothesis* (PSSH), a central hypothesis in both AI and Cognitive Science (Newell, 1980). According to the PSSH, mental processes are no more than the operations of a physical system that is capable of manipulating symbols. Not only human brains, but also computers have these capacities. The PSSH projects the mental onto the physical as follows. Mental concepts (including linguistic concepts, such as phonemes and words) are represented by physical symbols. The term *physical* means that the symbols should be implemented in some sense in physical reality, for example as electric states in a computer memory. Relations between concepts are represented by structures (groupings) of physical symbols. Mental functions are represented by physical processes notated as programs that manipulate physical symbol structures.

Programs are themselves represented as symbol structures, so that they can be manipulated by other programs or even by themselves. Because of this recursion, learning can be explained within this framework: the mind can change itself in useful ways by manipulating its own mental structures and programs through learning. According to the PSSH, the manipulation of symbol structures is both necessary and sufficient for cognition. A consequence of the PSSH is that cognition is independent from its physical realization. In other words, the hypothesis holds regardless of whether physical symbols are located in the human brain as networks of neurons, or implemented in a computer as pointers to memory

locations on a silicon chip. More detailed, layered views on computation and cognition based on symbol manipulation have been put forward in the literature (Newell, 1982; Pylyshyn, 1989; Steels, 1990; Marr, 1982).

2.3   Architectures

AI approaches complex tasks by decomposing them into subtasks so as to make the modelling of the task more manageable. Figure 1.1 in Chapter 1 provides an overview of comprehension and production of natural language and their possible decomposition into modules representing subtasks. In how far these modules are autonomous, and in how far they interact with one another, are topics of much debate. This debate goes on not only in AI, but also in connectionism, where there has recently been a move towards more modular architectures, e.g. CALM (see Chapter 3). One extreme is a strictly *sequential* architecture, where the different modules are accessed in sequence: output of one component is input to the next component. The other extreme is to have no modules at all, but to have an *integrated* system where knowledge at all levels acts together.

Clearly, other kinds of architectures exhibiting some limited form of interaction between the different modules have been proposed to address certain processing issues. In the area of sentence comprehension (see Chapter 8), for instance, a strictly sequential architecture is problematic. Possible syntactic structures can run into the hundreds or thousands for normal sentences, due to the combinatory explosion of several individual lexical and structural ambiguities. However, these often be resolved as a natural side-effect of solving semantic ambiguities. Therefore, a sequential 'syntax first' strategy, in which all possible syntactic parses are computed first and are then input to the semantic component, is impractical. Instead, a tighter interaction has been pursued by CL researchers as well as psycholinguists.

One possible interaction consists of a form or turn taking between modules in an *interleaving* architecture. This is realized, for example, in the interplay between syntax and semantics in the UNIFICATION SPACE (see Chapter 8). Another architecture consists of a direct *feedback* from a module to the previous one. This is for example the case in the sentence generator POPEL, where a module responsible for conceptualizing interacts with one for formulating (see Chapter 11). In *blackboard* architectures, modules do not communicate directly, but via a common channel called the blackboard. In *object-oriented* designs, objects representing parts of modules communicate with other objects via *message passing*. Parallelism can be exploited by letting a module start processing before the output of the previous one is complete. A *parallel* architecture has been proposed for some models of incremental sentence production discussed in Chapter 11.

Entirely different kinds of interactions are present in *connectionist* architectures. *Localist* connectionist systems enable direct interaction between symbols through weighted links, within and across modules. *Distributed* connectionist systems allow many symbols to be represented by different patterns of activity in a group of cells, thus making even the representation of symbols a matter of interaction. Chapter 3 provides a more in depth discussion of connectionism.

2.4   Knowledge representation formalisms

Within each module, the AI approach aims at putting together the necessary knowledge and methods needed for that module to accomplish its task. This information needs to be captured in a formalism for the representation and manipulation of knowledge. At this point, it is useful to reflect on the role of formalisms in AI in general and in linguistic models in particular. A formalism is a description language with a syntax and (ideally) unambiguous semantics that provides a bridge between the computer program and the theory.

A formalism consists of two parts. A *data organization* part consists of a language for describing domain entities, properties and relations involved. In a language processing model, this could contain knowledge about words and grammar rules, among other things. A second part, the *inference* part, determines methods for how the data can be used to carry out a task. Specific methods include *logical resolution, if-then rule application, inheritance*, etc., which will be discussed below.

In a trivial sense, most implemented representation formalisms are equivalent, because they can express any computation. In general, it will therefore always be possible to translate the knowledge represented in one formalism into another formalism. However, representation formalisms differ in the kinds of abstractions they promote, and thus differ in the ease of use for particular problems. AI has therefore been oriented toward the development of formalisms that are expressive and offer powerful abstraction mechanisms, for example, to classify information in a minimally redundant way. In addition, practical considerations of economy, readability, uniformity, and ease of maintenance (see for example Daelemans, De Smedt, and Gazdar, 1992) may be criteria for choosing one formalism over another.
However, the preference of certain representations over other ones should be theoretically motivated. Conflicts may occur. For example, a model of the lexicon that is linguistically felicitous removes as much redundancy as possible, but this economy is not necessarily psychologically felicitous (Stemberger & MacWhinney, 1986). The criteria of psychological realism on the one hand and linguistic abstraction on the other hand can thus be in conflict because they are relevant to different enterprises.

2.5   Some modelling principles

Having considered knowledge representation and some criteria for knowledge representation formalisms, we will now briefly discuss some principles and distinctions that are common in symbolic representations of knowledge.

2.5.1   Procedural/declarative representations
Let us consider the way in which the data organization and the inference methods are represented and operationalized. Many AI models use a *declarative* architecture, where the data organization part contains factual knowledge descriptions represented for example as lists of words and grammar rules. Problem solving methods are then implemented separately as operations performing inferences on this knowledge. For example, a model for syntactic

sentence parsing may consist of a general parsing method and a separately specified set of grammar rules for a specific language (see Chapter 8).

In a *procedural* architecture, the data organization and a specific problem solving method are integrated in the form of procedures to carry out a specific task (see Figure 2.1). Consequently, reflection on data organizations and inference methods is hardly possible. This is no problem for models of cognitive tasks that are considered automatic and unconscious activities. In Chapter 11, the IPG model is given as an example of a procedural architecture for natural language production. In this model, syntactic constituents and grammatical functions are viewed as active procedures, and the syntactic tree structure is not a data structure but a hierarchy of procedures calling other procedures.

Figure 2.1   Declarative and procedural architectures.
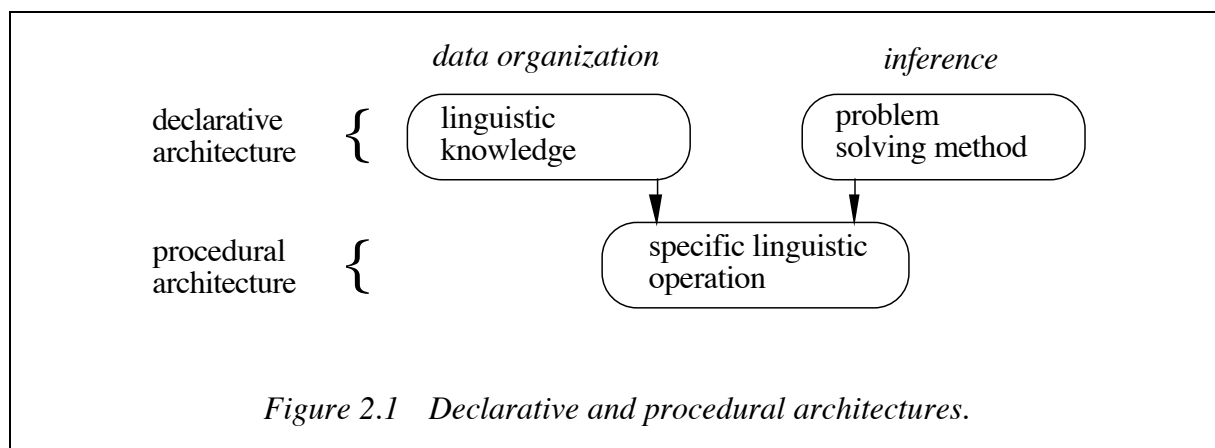
However, a procedural architecture is not always at an advantage. When linguistic knowledge and problem solving methods are tied together, they are harder to manipulate, so that the learning and development of procedures present serious problems. In particular, extensions to a procedural language processing system may lead to discrepancies between existing procedures and new situations, which would require additional procedures to solve the conflict, and so on. Most AI systems for natural language processing use a declarative architecture.

2.5.2   Type/token

In their data organizations, symbolic models usually distinguish between types and tokens. *Tokens* represent objects which can be identified in the real world and are unique, such as the first word you have spoken today. If this word happens to be *socks* then that occurrence of the word is unique and different from any other occurrence of the word *socks*. All occurrences are linguistic tokens of the same *type*, which is the word *socks* as it exists in the English vocabulary. The difference may be important for representational reasons. For example, in one sentence, *socks* may be the subject, whereas in another it may be the direct object. In one context, it may refer to clothing, whereas in another it may be the name of a cat. Should we want to represent both sentences at once in memory, then we can avoid conflicts by representing both tokens as separate units. Object-oriented models explicitly represent tokens as separate *instances* of types. Nevertheless, some models need to link repetitions of the same items more strongly and therefore use and reuse linguistic types rather than tokens. For

example, the spreading activation model for phonological encoding by Dell (1986) discussed in Chapter 13 uses units for each type of phoneme in each positional slot in the syllable. Repeated occurrences of these units are modelled by repeated activation of the same unit through time. The TRACE model discussed in Chapter 5 uses type units to handle the occurrences of a phoneme within a certain time window.

2.5.3   Search

Problem solving in AI is generally based on the representation of possible situations in the search for solutions. These situations are called *states* and the set of all states, together with operators to change from one state to another, is called the *state space*. Some of the states are begin states and some are goal states. Search methods are algorithms allowing a state space to be searched for a path leading from a begin state to a goal state. When the state space involves choices between possible paths leading from one state to the next, the state space takes the shape of a tree. Chapter 7 discusses the use of a tree shaped state space for segmenting complex words. This *L-tree* or *trie* is a lexicon structure where each node is labeled with a particular grapheme or phoneme. Words are looked up by matching each successive character in an input string with a node in the L-tree.

Methods searching a tree can be *blind*, which means that possible states are tried in a systematic but uninformed way. One kind of systematic search is *depth first*, where at each choice point between several possible next states, only the first alternative is considered, following this path as long as possible, and backtracking to the next possibility when necessary. Chapter 8 discusses a model for sentence comprehension based on ATNs with backtracking. Another kind or search is *breadth first*, where all alternatives may be considered in parallel. Examples from Chapter 8 are the Sausage Machine and race-based parsing, two models of sentence comprehension with limited forms of parallel search. These two search methods are illustrated in Figure 2.2

Figure 2.2   Depth-first search (left) and breadth-first search (right).

Instead of being blind, search methods can make use of *heuristics,* i.e. knowledge about the domain to traverse the state space in a more efficient way (Pearl, 1980). An example of such a heuristic in sentence comprehension is *right association:* the preference to attach a new constituent to the rightmost node of a syntactic tree (see Chapter 8). One problem with heuristics is that their use can lead to a *local minimum*, which means that they may lead to a path that seems the most promising in a local context but turn out to be bad in a global context (as illustrated by *garden path* sentences). More recent and sophisticated forms of

search include probabilistic methods that help to escape from local minima. In the UNIFICATION SPACE, discussed in Chapter 8, the state space is searched with a probabilistic optimization technique.

2.5.4　Planning

Searching a solution for a problem is often viewed as *planning* in AI work, and planning techniques from everyday life are applied in the process. These techniques include, for instance, the division of plans in separate smaller steps, each achieving a particular subgoal (*hierarchical* planning), the analysis of appropriate means to achieve certain ends (*means-ends* analysis), and on-the fly modification of plans to achieve secondary goals when appropriate (*opportunistic* planning). Chapter 10 takes a strong plan-oriented approach to discourse production. The TEXT model discussed in that chapter makes use of *schemata*, which are prepackaged plans for the generation of structured discourse. In order to generate discourse defining an object, for example, TEXT has the choice between several schemata, including a schema which describes parts of the object and one which lists defining characteristics.

Whereas schemata are hierarchically structured plans, more recent text planners take a more flexible approach, e.g., by assembling plans *incrementally*, in the order in which the plan elements appear in the final text, and by choosing plan elements depending on the current context (see Chapter 10). Also, the modelling of spontaneous speech, for example, requires incremental planning. This is necessary to account for the fact that people sometimes start speaking before they have delineated the complete content of their utterance. Models of incremental planning in grammatical encoding are discussed in Chapter 11.

2.6　AI paradigms and formalisms for knowledge representation

There are two opposing trends in the development of formalisms for linguistic knowledge representation. Some of the research is aimed at using and refining existing general purpose AI paradigms for the representation of all knowledge, including linguistic knowledge. A full inventory is beyond the scope of the current chapter (see our short bibliographic guide below). In the following sections we will group general purpose formalisms into three main paradigms: frame-based formalisms, rule-based formalisms, and logic. Other research is aimed at the development of special purpose formalisms for language. Among these, only a limited number of grammar formalisms will be discussed. The choice of formalisms and examples is mostly determined by the models which will be further discussed in later chapters of this book, while only a few other formalisms that do not return in the remainder of this book are added because they are important.

2.6.1 Structured knowledge representation

Semantic or associative networks

Frame-based formalisms grew out of semantic (or associative) networks that were introduced in the sixties as models for human semantic memory (Quillian, 1968; Collins & Quillian, 1970; Collins & Loftus, 1975; Brachman, 1979). The data organization of a semantic network consists of labeled *nodes* representing concepts and labeled *links* representing relations between concepts. Nodes can be used to represent both types and tokens. The *is-a* link hierarchically relates types to types or types to tokens, *part-of* links relate concepts and their parts, and in general any relation can be the label of a link. Figure 2.3 shows the common graphical notation of an example network representing some of the knowledge underlying the utterance *Lucy weeded the vegetable garden*. Note that *Lucy* is a named individual, and *vg012* is a token whereas *garden,* for instance, is a type.

Figure 2.3   Part of a semantic network.
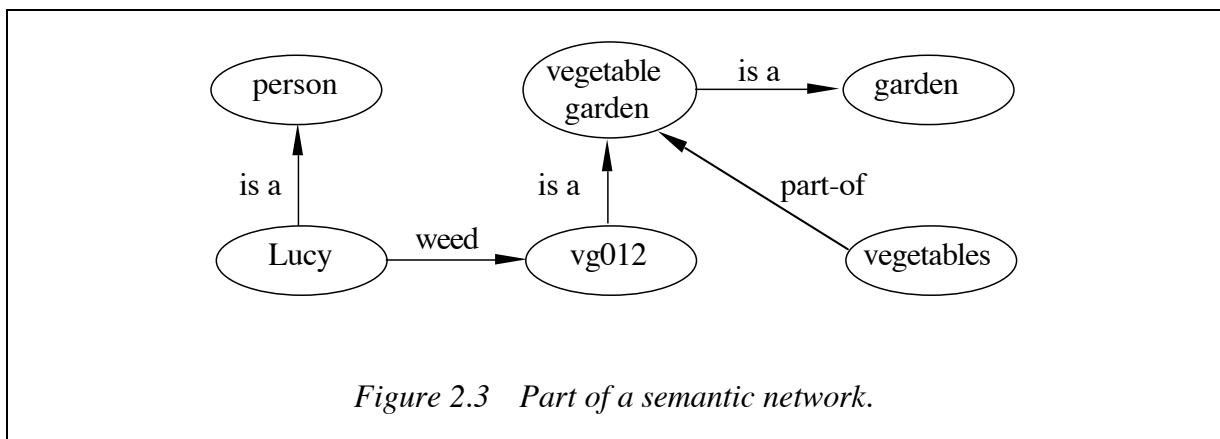
In a semantic network, each link between two nodes represents a separate proposition, e.g. the fact that Lucy is a person is a proposition separate from the fact that Lucy weeds the garden and the fact that the garden contains vegetables. But suppose that we want to represent relations between more than two nodes. A solution is to allow nodes to represent situations or actions. Each such node has outgoing links representing *thematic roles* (or *cases*) for participants in the situation or action. An example is depicted in Figure 2.4. Semantic networks and variants thereof have been used in several psycholinguistic models. Chapter 9 discusses how networks can be used to represent the meaning of a text, e.g. in Kintsch's construction-integration model. Chapter 12 discusses Roelofs' model for non-decompositional lexical semantics, based on a network with *is-a* and other links.
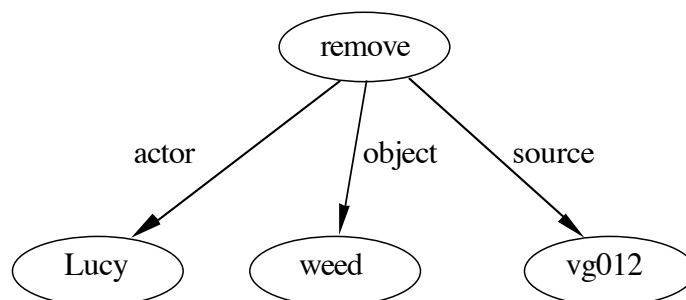
Figure 2.4   A semantic network representing an action with roles.

Frames

In order to incorporate more structure into semantic networks, *frames* were introduced by Minsky (1975) as a representation formalism. A frame is basically a encapsulated fragment of a semantic network with conceptually related nodes and links that can be addressed as a unity. For example, one can define a frame for the *remove* action and the associated roles *actor*, *object* and *source*.

The term *frame* is not only used for structures representing domain knowledge, but also for structures representing linguistic knowledge. In Dells' (1986) model for phonological encoding, discussed in Chapter 14, for example, a *syllable* frame may consist of an *onset*, a *nucleus*, and a *coda*. In the syllable *best*, these roles are filled by *b*, *e*, and *st*, respectively. Several modifications have been proposed to make frames so powerful that they can be used as high-level programming languages: (1) *constraints* on the items allowed to fill each slot in the list; (2) *recursive* frames, the slots of which may contain items that are themselves frames, (3) *procedural attachment,* in which procedures are attached to roles of the frame to compute the fillers of these roles, and (4) *inheritance*, which makes inferences along *is-a* links; this is dealt with in more detail in the next section.

Inheritance

Given the enormous number of linguistic facts that are brought to bear in language processing, it is clearly important to represent this knowledge in an efficient and general way. *Inheritance* is a powerful technique to represent generalizations over descriptions in a way similar to *is-a* relations in semantic networks, and to use these generalizations to make inferences. Daelemans et al. (1992) motivate the use of inheritance for linguistic knowledge as follows. Imagine that we are setting out on the job of building a lexicon for English. We begin by encoding everything we know about the verb *love,* including its syntactic category and how it is conjugated. Next, we turn our attention to the verb *hate*. Although these words are antonyms, they nevertheless have a lot in common: for example, they are both transitive verbs and have past participle ending on *-ed*. To save time and space, both can be categorized as transitive verbs and the common information about these kinds of words can be encoded in just one place called, say, *transitive verb*. The verb *love* then inherits information from

transitive verb. Similarly, when we hit upon intransitive verbs, we collect all information about this kind of verbs in a place *intransitive verbs*. The next step is to extract from both classes of verbs the common information which is then stored in an even more abstract category *verb*. This is shown in Figure 2.5.

Figure 2.5   Inheritance hierarchy of some English verbs.
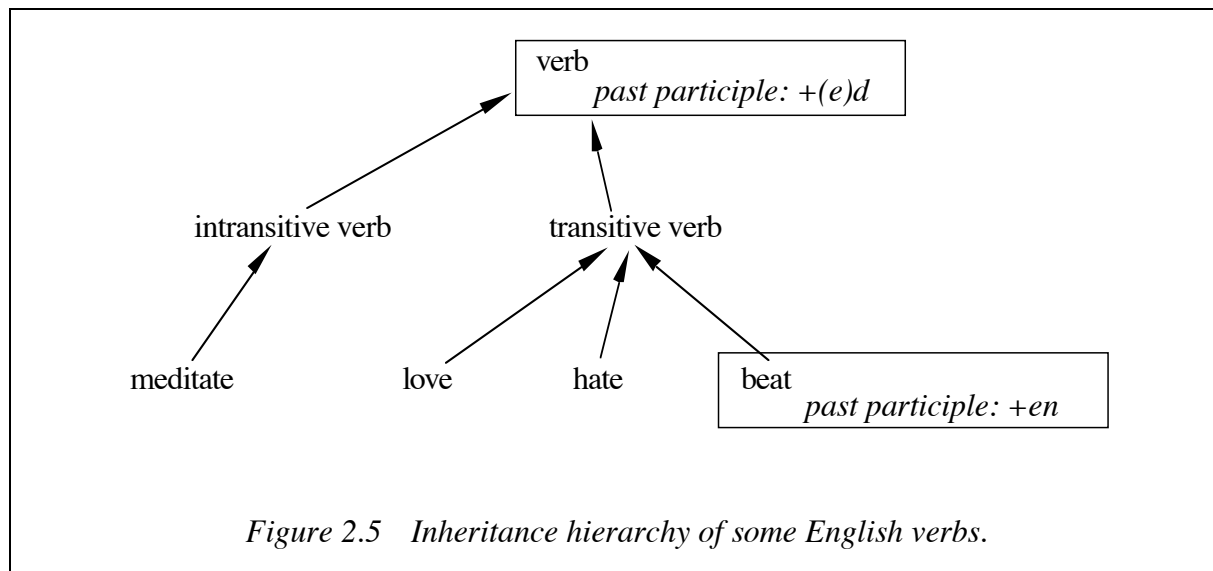
Suppose we discover the verb *beat,* which is transitive but not regular: it has a past participle ending on *-en*. If we let *beat* inherit from *transitive verb,* we still need to specify the exceptional information, but then we get an inconsistency with the inherited information. The obvious solution is to let the exceptional information override inherited information. This mechanism is called *default* (or *non-monotonic*) inheritance. Default inheritance is incorporated in semantic networks and frame-based representation languages used in AI. The sentence production model IPF, discussed in Chapter 11, is implemented in a frame-based language using default inheritance. Default inheritance is also used in specific linguistic formalisms, for example DATR (Evans & Gazdar, 1989), a formalism for lexical representation.

Marker Passing and Spreading Activation
One of the inference mechanisms available in semantic networks is *marker passing* , a process with which *intersection search* can be implemented. This type of search starts from two nodes which pass a marker to those nodes which are linked to them, a process which is repeated for each of those nodes. Whenever a marker originating from one of the original nodes encounters a marker originating from the other node, a path is found representing a association between the two concepts represented by the nodes. This way, semantic associations can be modeled, but marker passing can also be used in networks representing other types of linguistic knowledge. A similar inference mechanism in networks is *spreading activation* (explained in Chapter 3), where instead of discrete symbolic markers, a continuous (numerical) activation level is propagated along the links of a network. Chapter 12 discusses a model of lexical retrieval where a semantic network with labeled links is combined with spreading activation.

Conceptual dependency structures and conceptual graphs

Many of the early symbolic AI research on natural language understanding used semantic network or frame-based formalisms to represent its theoretical insights. Schank and his students developed CONCEPTUAL DEPENDENCY THEORY for the description of the meaning of sentences and texts (Schank, 1975; 1980). This theory was based on semantic networks, but defined only a limited number of node types and link types (conceptual primitives) that were deemed necessary and sufficient as a language of thought to represent meaning unambiguously. Any implicit information in the text (information that can be inferred by the reader) was to be made explicit in the conceptual dependency representation.

This goal gave rise to the development of a large number of data structures and inference mechanisms (often without a well-defined semantics). Data structures included *causal chains* (a chain of states enabling or motivating actions which in turn result in, or initiate, other states), *scripts* and *scenarios* (prepackaged sequences of causal chains), and MOPS (Memory Organization Packages) (Schank & Abelson, 1977; Schank, 1982). These data structures enabled directed and efficient inference mechanisms, based on following up causal connections and associations between representations at the same and at different levels of abstraction. Work by Sowa (e.g. Sowa, 1984; 1991) on CONCEPTUAL GRAPHS also follows this approach. Chapter 9 discusses some models using *scripts, scenarios* and MOPS as data structures for discourse comprehension. One problem is that these models tend to focus on the data structure, and are vague on the inference part.

The work by Schank and his students also made clear that two sources of knowledge are indispensable for developing useful symbolic natural language understanding systems: (1) knowledge about the intentions, plans and goals of different *agents* in narratives or dialogue, and (2) knowledge about preceding discourse (discourse representation). In work by Allen and Perrault (1980) and others, AI planning formalisms are combined with speech act theory to model the recognition of intention, an approach which gave rise to research on speech act planning, topic structure modeling, and user modeling. This AI work has influenced psycholinguistic models of discourse comprehension (see Chapter 9) and discourse production (see Chapter 10).

2.6.2 Production systems

Production systems are rule-based systems developed during the seventies as models for human problem solving (Newell & Simon, 1972). They are common in models for many areas of knowledge. In this kind of formalism, knowledge is expressed as rules taking the form of condition-action pairs: *if X then do Y*. For example, in a model for language production, one of the rules for producing questions might be the following:

> **if** *the intention is to query the truth of P,*
> **then** *produce a sentence about P where the finite verb of the main clause is moved up front*.

Rules of this type, often called *production* rules, can only produce actual behavior with the help of an *interpreter,* a mechanism which applies the rules to reach a given goal. In addition to the rule-base (which acts as a kind of long-term memory), a production rule system also has a short-term memory (working memory) which registers the current state of the

computation, as well as current input and output states. The control structure of a production system interpreter consists of a cyclical process, where each cycle consists of three phases:

1. *Identification*. This phase determines for which rules the condition sides are currently satisfied in working memory.

2. *Selection*. It will often happen that more than one rule's condition side will be satisfied. Since in general it is not desirable for all applicable rules to fire, one or more rules are selected on the basis of a particular conflict resolution strategy.

3. *Execution*. The action part of the chosen rule is executed. Although actions can take many forms, the most typical ones involve the addition to or removal from working memory of certain facts.

This interpreter's mode of operation is called *forward chaining* or data-driven: rules are identified when states in working memory match their conditions; the execution of the rules may in their turn activate other rules, until a goal is achieved. But it is also possible to run an interpreter in a *backward chaining* or goal-driven mode: in that case, rules are identified when their actions match the current goals; their execution may add elements of their conditions as new goals when they are not present in working memory, and so on, until rules are found whose conditions match the current states in working memory. It is evident that both modes represent different kinds of search.

Rule-based architectures have been further developed toward more sophisticated cognitive architectures, for example, ACT* (Anderson, 1983) and SOAR (Laird, Newell & Rosenbloom, 1987; Rosenbloom, Laird & Newell, 1993). The ACT* system has a semantic network (see above) as part of its long term memory.

Production systems have been used in a few psycholinguistic models, but no models based on them are discussed in the remainder of this book. Anderson, Kline and Lewis (1977) describe a production system model of language processing. In PROZIN (Kolk, 1987), agrammatism effects are simulated by manipulating the processing speed of the production system interpreter and the decay rate of facts in working memory. Lewis (1993) describes a computer model of human sentence comprehension implemented in SOAR.

2.6.3   Logic

Logic has often been used as a formal foundation for knowledge representation in AI. For this reason it is mentioned here, even if no psycholinguistic models discussed in this book are directly based on logic. The formal properties of logic formalisms are relatively well understood and make them ideally suited as a language to which other formalisms can be translated in order to evaluate and compare them.

Data organization in predicate logic consists of a set of unambiguous constants (representing entities in the domain), a set of unambiguous predicates (representing relations between entities in the domain), a set of functions (mapping between sets), variables, quantifiers, and logical connectives. Inference in predicate logic is achieved by applying deductive inference rules, e.g. by means of *resolution* . For an introduction to the logical approach to knowledge representation, see e.g. Ramsay (1988).

A practical computer language based on a limited version of predicate logic is PROLOG (Clocksin & Mellish, 1984). Below is a small program that expresses the fact that Socrates and Plato are human, and the rule that if *x* is human, then *x* is mortal:

```
human(socrates).
human(plato).
mortal(X) :- human(X).
```

The *interpreter* of PROLOG uses these facts and rules to derive other facts. For example, the following dialog is possible, where we ask whether Socrates and Descartes are mortal, and who are all mortal beings the system knows. The system infers, for instance, that Socrates and Plato are mortal. Notice that PROLOG gives a negative answer for everything that does not occur in the knowledge base.

```
|?-mortal(socrates).
yes
|?-mortal(descartes).
no
|?-mortal(X).
X=socrates;
X=plato;
no
```

Predicate logic has some severe limitations as a tool for representing linguistic knowledge which is incomplete, inconsistent, dynamically changing, or relating to time, action and beliefs. For all these problems, special purpose logics are being designed. An example is default logic, which handles exceptional information without having to modify existing general knowledge (see the section on *inheritance* above).

2.7   Grammar formalisms

Grammar formalisms constitute a special type of formalism for natural language processing, even though they are not unrelated to the knowledge representation paradigms and formalisms discussed earlier. They often use a different terminology, due to the different background of the developers, which is linguistics, logic, and theoretical computer science rather than AI, and use special notations for linguistic strings and structures. Most grammar formalisms were developed as part of the efforts to build systems for natural language understanding, which up to now received more attention in AI than natural language generation.

2.7.1   Phrase structure grammars and automata

The representation of grammatical knowledge as *phrase structure rules* is common for syntactic parsing in sentence comprehension (see Chapter 8), and to some extent, the recognition of complex words (see Chapter 7). The use of these rules is somewhat similar to production rules, but they operate on strings of linguistic items. Phrase structure rules basically specify how an initial symbol can be recursively expanded into a sequence of other symbols.

For example, the first rule in the rule set below specifies that a sentence (S) can be expanded into a noun phrase (NP) followed by a verb phrase (VP), or, inversely, that a noun phrase and a verb phrase can be reduced to a sentence. The selection mechanism chooses among various applicable rules. Often, a symbol can be expanded into different ways, for example in the following rule set describing how an NP can be rewritten as either an article followed by a noun, or an article followed by an adjective, followed by a noun.

```
S -> NP VP
NP -> PRONOUN
NP -> ART N
NP -> ART ADJ N
VP -> COPULA NP
PRONOUN -> she
COPULA -> is
ART -> the
ART -> a
ADJ -> nice
ADJ -> smart
N -> doctor
```

A *deterministic* system will choose only one rule, whereas a *non-deterministic* system may search through a space of possibilities, using e.g. a parallel or backtracking search (see Section 2.5.3). Chapter 9 argues for determinism, as embodied e.g. in PARSIFAL. When a rule is chosen, the left hand side of the rule is replaced with the right hand side. Successive expansions develop the structure until a solution is reached in the form of a sequence of words. The expansion history of a particular case can be represented as a syntactic tree structure, for example the one in Figure 2.6. Clearly, different grammars give rise to different tree structures.

Figure 2.6   A syntactic structure for She is a nice doctor.
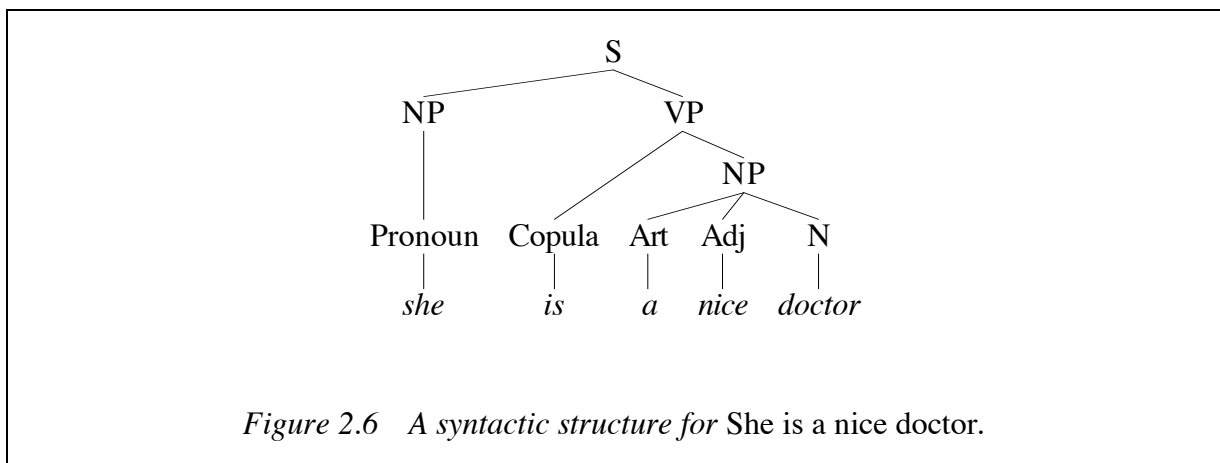
Phrase structure rules may operate in both directions, top-down, where the left hand sides of rules are rewritten as their right hand sides, or bottom-up, where the right hand sides are rewritten as the left hand sides. Depending on the form the left-hand side and the right-hand side of the phrase structure rules can take, different types of grammars can be formally defined: regular, context-free, context-sensitive, or unrestricted (see e.g. Hopcroft & Ullman, 1979; Wall, 1972). Much research in CL is based on context-free grammars.

Languages can be described by grammars, but they can also be characterized by abstract computing devices called *automata* or *transition networks*. An automaton is an idealized machine which receives an input *tape* on which it performs operations according to given instructions. Automata have internal *states* and during computation they can make transitions from one state to another. If the automaton reaches a state designated as a final state, this may signify that the input has either been accepted or rejected. Thus, an automaton is effectively a recognizer of sentences (or other linguistic units) corresponding to a given grammar. There are four main classes of automata, corresponding to the different grammar types (see e.g. Hopcroft & Ullman, 1979; Wall, 1972).

One class of automata are *finite state automata*. Figure 2.7 shows an simple automaton that recognizes several kinds of sequences including those consisting of *art adj adj n*, e.g. *a nice smart doctor*. The process starts in an initial state NP1 and accepts words, which allow it to perform transitions to other states. If after the last word, a state NP3 is reached, which is designated as a final state, the process of recognition is successful. Automata can be used to generate sentences as well.

Figure 2.7   A transition network for simple NPs.
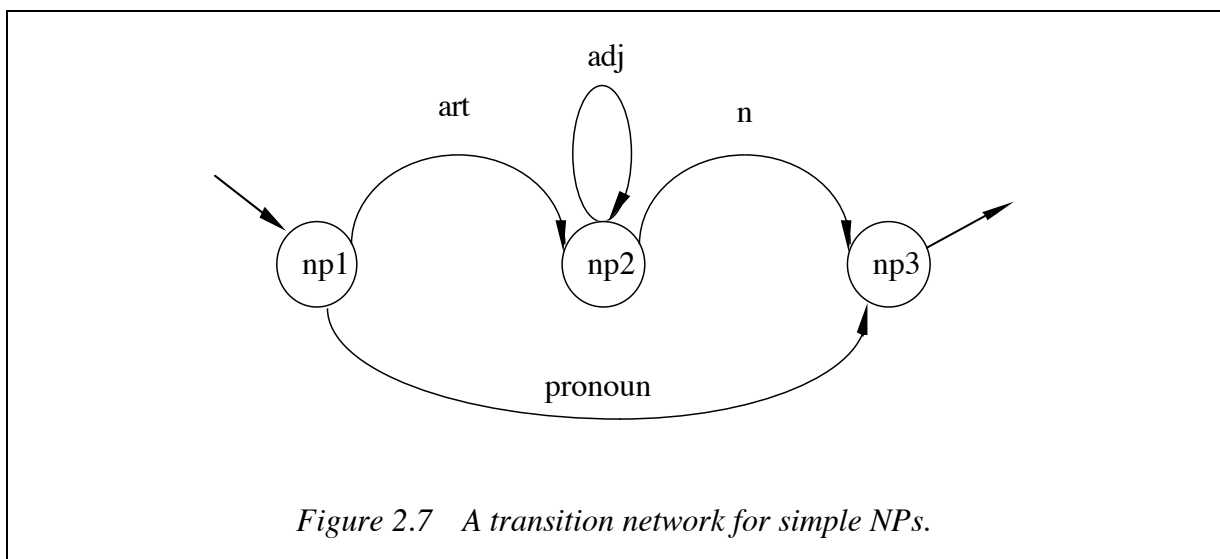
Some variants of automata have been defined and applied to parsing. In Recursive Transition Networks (RTNs), not only a word can be accepted on a transition, but also a network can be called recursively in order to recognize a string of words. Chapter 9 discusses Augmented Transition Networks (ATNs), which are not only recursive, but in which transitions can be coupled to tests and to operations on memory registers. This memory can be used to store information, e.g. to build syntactic structures. Another variant of automata are *transducers*, which operate on several input and output tapes at the same time. Chapter 7 discusses models based on *two-tape finite state transducers* for morphological analysis.

Clearly, parsing with a transition network can be seen as a kind of search. One way in which a search can often be made more effective is by storing the results of partial computations, so that when the search fails, one does not need to start from scratch. A *chart* parser is a device where the results of partial parses are stored in a working structure called a *chart* (e.g. Winograd, 1983). Figure 2.8 shows an example of a chart made with the grammar

given above. Chapter 7 discusses a model of morphological analysis based on charts. In that model, the items between nodes are morphemes rather than words.

Figure 2.8   A chart containing a record of constituents found.
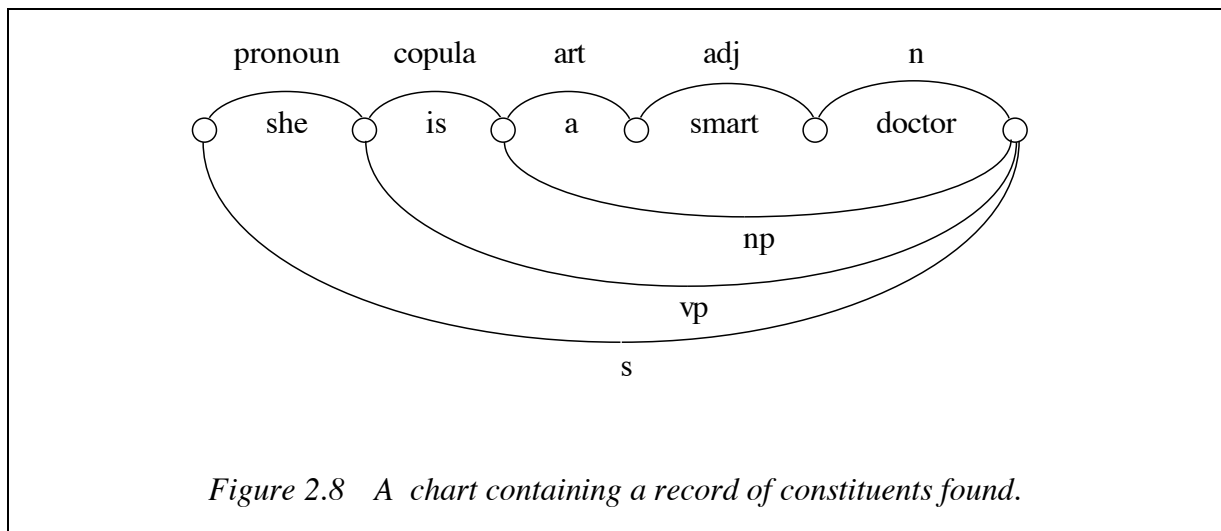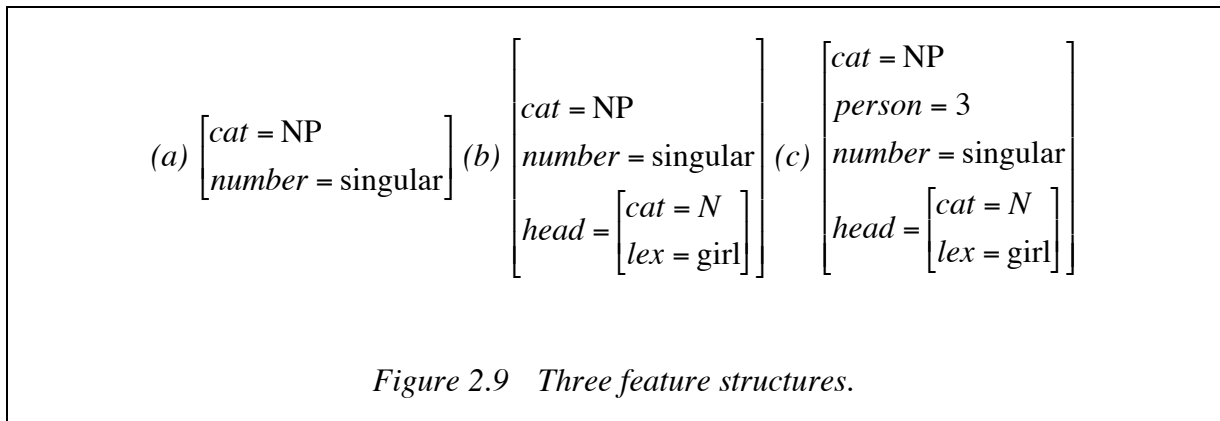
2.7.2   More expressive formalisms

During the last decade, many grammar formalisms have been devised that have in some way or other departed from simple phrase structure rules (see e.g. Sells, 1985; Shieber, 1986). They have been applied primarily to syntax, although they are also used for modeling other levels of processing. For example, categories can be made more abstract by the use of wildcards, for example, XP may stand for NP as well as for PP. Another way to increase abstractness is separating information about left to right order from information about hierarchical relations (in terms of immediate dominance). In the following set of rules, for example, the first two rules representing immediate dominance are supplemented by a third rule which is an abstract ordering rule:

```
VP -> V, NP
VP -> V, NP, PP
V < XP.
```

Another way in which the rules are made more expressive is by adding tags to categories in order to represent syntactic features, for example, for number and person. In this way, information about agreement, for example between the subject and the finite verb, is factored out. In many formalisms, feature structures (or feature graphs, or feature matrices) are the common way to represent linguistic information. A feature structure is a recursively defined structure consisting of a set of features and their values. A frequently used notation writes feature-value pairs as *feature = value,* and puts square brackets around the whole feature structure. Figure 2.9 gives three examples of feature structures in this notation: (a) describes singular NPs, (b) describes third person NPs with as their heads the noun *girl*, and (c) describes third person singular NPs with as their heads the noun *girl*.

$$(a) \begin{bmatrix} cat = \text{NP} \\ number = \text{singular} \end{bmatrix} (b) \begin{bmatrix} cat = \text{NP} \\ number = \text{singular} \\ head = \begin{bmatrix} cat = \text{N} \\ lex = \text{girl} \end{bmatrix} \end{bmatrix} (c) \begin{bmatrix} cat = \text{NP} \\ person = 3 \\ number = \text{singular} \\ head = \begin{bmatrix} cat = \text{N} \\ lex = \text{girl} \end{bmatrix} \end{bmatrix}$$

Figure 2.9   Three feature structures.

Most of the new grammar formalisms based on feature structures are unification-based. Unification is prescribed as the sole information-combining operation. It can intuitively be described as a combination of two feature structures into a new one (see Shieber, 1986, for a more formal description). For example, feature structure (c) is the unification of (a) and (b). Besides the unification operation, a unification-based grammar consists of only declarative knowledge expressed in feature structures.

Kay (1979; 1984) proposed a formalism called Functional Grammar, later called Functional Unification Grammar (FUG), which works along the lines sketched above. A unification grammar can be described as a disjunction of feature graphs describing all possible sentence forms in a language. The processing of a sentence with such a grammar is a search for the unification of an initial description of the sentence with one of the alternatives in the grammar.

Several influential theories of language use linguistic descriptions which are feature structures, including LEXICAL FUNCTIONAL GRAMMAR (LFG; Kaplan & Bresnan, 1982), GENERALIZED PHRASE STRUCTURE GRAMMAR (GPSG; Gazdar, Klein, Pullum & Sag, 1985), and HEAD-DRIVEN PHRASE STRUCTURE GRAMMAR (HPSG; Pollard & Sag, 1987). We briefly mention some other grammar formalisms, which have been created as extensions of older formalisms by the unification operation: UNIFICATION CATEGORIAL GRAMMAR (UCG) was derived (Calder, Klein & Zeevat, 1988), and FEATURE STRUCTURES BASED TREE ADJOINING GRAMMAR (FTAG; Vijay-Shanker & Joshi, 1988).

Logic grammars view language processing as resolution in the logic sense. Because resolution can be seen as unification, there is a strong link between logic grammars and other unification based formalisms. In DEFINITE CLAUSE GRAMMAR (DCG; Pereira & Warren, 1980), for example, grammar rules are expressed in a way quite similar to phrase structure rules, but are translated into PROLOG clauses. The PROLOG resolution mechanism then executes the program. The rules of DCG have the general form of phrase structure rules (rewrite rules). Below is a small grammar for sentences of a simple form.

```
s-->np, vp.
np-->pronoun.
np-->determiner, noun.
vp-->copula, np.
pronoun-->[she].
copula-->[is].
determiner-->[a].
determiner-->[the].
noun-->[doctor].
```

The program can be used to recognize or generate sentences that conform to the grammar. Below is an example of two queries to recognize sentences.

```
|?-s([she,is,a,doctor],[])
yes
|?-s([is,she,a,doctor],[])
no
```

SEGMENT GRAMMAR (SG; Kempen, 1987; De Smedt & Kempen, 1991) is a unification-based formalism especially proposed for incremental syntactic processing. This formalism views a grammar as a collection of syntactic segments. Each segment represents a single hierarchical (immediate dominance) relation between two categories. The relation between a sentence and a noun phrase that is its subject, for example, is represented as the segment S-subject-NP (see Chapters 8 and 11 for details and examples). The essence of sentence processing in SEGMENT GRAMMAR consists of using such segments as building blocks in the construction of a syntactic structure for a sentence, joining them by unification. For example, a path S-subject-NP-head-N can be formed by unifying the NP node in an S-subject-NP and that in an NP-head-N segment. Chapter 11 discusses the IPF model for grammatical encoding, which is based on the construction of syntactic structures out of segments. Chapter 8 explains a variant of SEGMENT GRAMMAR where unification is turned into a probabilistic operation dependent not only on the feature composition of the nodes to be unified, but also on the activation levels of these nodes (see Chapter 3 for activation based paradigms).

2.8   Advantages and disadvantages of symbolic systems

Symbolic approaches that are based on logic, frame-based systems, production systems, grammar formalisms, or on a combination of these representation techniques, are able to successfully perform complex natural language processing tasks. Thanks to the definition of formal operations – procedures that operate on the form of structures, irrespective of their content – symbolic systems achieve a high level of abstraction. New symbols and structures can be created dynamically during execution of a program. Moreover, symbolic structures can easily be defined recursively and can thus represent a potentially infinite number of actual structures.

However, symbolic systems have a few drawbacks. First, when symbols are represented as single pointers to memory locations, a symbolic system is vulnerable when the properties of even a single symbol change. Second, symbolic systems are rigid and complex. Each exception requires additional rules and more processing. This is particularly problematic as

the system is scaled up, even though the problem of scaling up can be somewhat alleviated by the use of powerful mechanisms such as default inheritance. It is this sheer complexity that makes the system vulnerable in the case of ill-formed or incomplete input and in the case of unforeseen interactions between rules. When a symbolic system goes wrong, it usually does not degrade gracefully, but breaks down completely. Third, the data and methods must generally be hand-coded by the system designer, because their complexity makes it hard to acquire them automatically. Machine learning of natural language from data like corpora or machine-readable dictionaries is therefore becoming an increasingly important topic, as it may alleviate these knowledge acquisition and robustness problems.

The PSSH goes a long way toward providing a framework for the study of *knowledge-based intelligence,* that is, intelligence based on the construction and manipulation of models. However, this is less the case for *behavior-based intelligence,* that is, intelligent behavior based on direct associations between sensory input and motor output without intermediate models. It is an open research question whether language processing is an instance of behavior-based or knowledge-based intelligence, or both. It also remains to be seen whether language is indeed a task much like other cognitive tasks, for example playing chess, solving algebra problems, or recognizing visual objects, or whether it requires a mode of processing that is unique.

In this chapter we have touched upon a few topics in traditional AI, but it must be stressed that AI is always incorporating new ideas from computer science and other disciplines such as neurology and biology. Recently, AI has seen the influence of radically new computing paradigms, including genetic algorithms, complex dynamic systems, and several kinds of brain style computing which are usually grouped under the term *connectionism.* The adoption of the new computing paradigms into mainstream AI has recently been stimulated by the availability of massively parallel hardware. Chapter 3 of this volume will introduce connectionism and will show how connectionist approaches are designed to overcome the robustness and acquisition problems of traditional AI systems.

2.9   Epilogue: finding your way in AI and CL

A thorough introduction to even a small subset of the formalisms, techniques and theories developed in symbolic AI and CL would require several times the size of this chapter. However, there are several good textbooks and reference works that can be used to get a deeper knowledge about the concepts introduced in this chapter. Two recent textbooks on AI are Winston (1992) and Luger and Stubblefield (1993), an older one is Charniak and McDermott (1985). They include chapters on CL formalisms. The Encyclopaedia of AI (Shapiro, 1992) and the Handbook of AI (Barr, Feigenbaum & Cohen, 1986-1989) provide introductions to all subfields and most concepts in AI and CL, and contain numerous references to the AI literature. There are anthologies of articles on AI (Webber & Nilsson, 1981), knowledge representation (Brachman & Levesque, 1985), and natural language processing (Grosz, Sparck Jones & Webber, 1986). The textbook by Winograd (1983) is a classic introduction to syntactic processing.

Programming is an essential skill for anyone who wants to develop AI models. Languages like LISP and PROLOG are especially suited to implement the formalisms discussed in this chapter. Winston and Horn (1988) and Norvig (1992) are excellent textbooks for learning how to program AI formalisms in LISP. Bratko (1986) and Flach (1994) do the same for PROLOG. Excellent textbooks especially devoted to CL are Gazdar and Mellish (1989a; 1989b), which introduces the most important CL formalisms with their implementation in Lisp or Prolog, and Allen (1994). Pereira and Shieber (1987) is a classic introduction to implementation of CL formalisms in PROLOG.

2.10   References

Allen, J. (1994). *Natural Language Understanding* (2nd ed.). Reading, MA: Addison-Wesley.

Allen, J. F., & Perrault, C. R. (1980). Analyzing intention in utterances. *Artificial Intelligence, 15*, 143–178.

Anderson, J. R. (1983). *The architecture of cognition*. Cambridge, MA: Harvard University Press.

Anderson, J. R., Kline, P., & Lewis, C. (1977). A production system model of language processing. In M. A. Just & P. A. Carpenter (Eds.), *Cognitive processes in comprehension* (pp. 271–311). Hillsdale, NJ: Lawrence Erlbaum.

Barr, A., Feigenbaum, E. A., & Cohen. P. R. (1986–1989). *The handbook of artificial intelligence* (Vols. 1–2: ed. by A. Barr & E. A. Feigenbaum; Vol. 3: ed. by P. R. Cohen & E. A. Feigenbaum; Vol. 4: ed. by A. Barr, P. R. Cohen & E. A. Feigenbaum). Reading, MA: Addison-Wesley.

Brachman, R. J. (1979). On the epistemological status of semantic networks. In N. V. Findler (Ed.), *Associative networks: Representation and use of knowledge by computers* (pp. 3–50). New York: Academic Press.

Brachman, R. J., & Levesque, H. J. (Eds.). (1985). *Readings in knowledge representation*. Los Altos: Morgan Kaufmann.

Bratko, I. (1986). *PROLOG programming for artificial intelligence*. Reading, MA: Addison-Wesley.

Calder, J., Klein, E., & Zeevat, H. (1988). Unification categorial grammar. In *Proceedings of the 12th International Conference on Computational Linguistics, Budapest* (pp. 83–86). Morristown, NJ: Association for Computational Linguistics.

Charniak, E., & McDermott, D. (1985). *Introduction to artificial intelligence*. Reading, MA: Addison-Wesley.

Clocksin, W. F., & Mellish, C. S. (1984). *Programming in PROLOG* (2nd ed.). Berlin: Springer.

Collins, A. M., & Loftus, E. F. (1975). A spreading-activation theory of semantic processing. *Psychological Review, 16*, 399–412.

Collins, A. M., & Quillian, M. R. (1970). Facilitating retrieval from semantic memory. *Acta Psychologica, 33*, 304–314.

Copeland, J. (1993). *Artificial Intelligence: A philosophical introduction*. Oxford: Blackwell.

Daelemans, W., De Smedt, K., & Gazdar, G. (1992). Inheritance in natural language processing. *Computational Linguistics, 18,* 205–218.

Dell, G. (1986). A spreading activation theory of retrieval in sentence production, *Psychological Review, 93,* 283–321.

De Smedt, K., & Kempen, G. (1991). Segment grammar: A formalism for incremental sentence generation. In C. L. Paris, W. R. Swartout & W. C. Mann (Eds.), *Natural language generation in artificial intelligence and computational linguistics* (pp. 329–349). Dordrecht: Nijhoff (Kluwer).

Evans, R., & Gazdar, G. (1989). Inference in DATR. In *Proceedings of the 4th Conference of the European Chapter of the ACL, Manchester* (pp. 66–71). Morristown, NJ: Association for Computational Linguistics.

Flach, P.A. (1994). *Simply logical: Intelligent reasoning by example*. New York: Wiley.

Garnham, A. (1994). Future directions. In M. A. Gernsbacher (Ed.), *Handbook of psycholinguistics* (pp. 1123–1144). San Diego: Academic Press.

Gazdar, G., Klein, E., Pullum, G., & Sag, I. (1985). *Generalized phrase structure grammar*. Oxford: Basil Blackwell.

Gazdar, G., & Mellish, C. (1989a). *Natural language processing in LISP: An introduction to computational linguistics*. Reading, MA: Addison-Wesley.

Gazdar, G., & Mellish, C. (1989b). *Natural language processing in PROLOG: an introduction to computational linguistics*. Reading, MA: Addison-Wesley.

Grosz, B. J., Sparck Jones, K., & Webber, B. L. (Eds.). (1986). *Readings in natural language processing*. Los Altos, CA: Morgan Kaufmann.

Hopcroft, J. E., & Ullman, J. D. (1979). *Introduction to automata theory, languages, and computation*. Reading, MA: Addison-Wesley.

Kaplan, R., & Bresnan, J. (1982). Lexical-functional grammar: A formal system for grammatical representation. In J. Bresnan (Ed.), *The mental representation of grammatical relations* (pp. 173–381). Cambridge, MA: MIT Press.

Kay, M. (1979). Functional grammar. In *Proceeding of the 5th Annual Meeting of the Berkeley Linguistic Society* (pp. 142–158).

Kay, M. (1984). Functional unification grammar: A formalism for machine translation. In *Proceedings of COLING84, Stanford* (pp. 75–78). Morristown, NJ: Association for Computational Linguistics.

Kempen, G. (1987). A framework for incremental syntactic tree formation. In *Proceedings of the 10th International Joint Conference on Artificial Intelligence, Milan* (pp. 655–660). Los Altos: Morgan Kaufmann.

Kolk, H. (1987). A theory of grammatical impairment in aphasia. In G. Kempen (Ed.), *Natural language generation: New results in artificial intelligence, psychology and linguistics* (pp. 377–391). Dordrecht: Nijhoff (Kluwer Academic Publishers).

Laird, J. E., Newell, A., & Rosenbloom, P. S. (1987). SOAR: An architecture for general intelligence. *Artificial Intelligence, 33,* 1–64.

Lewis, R. L. (1993). An architecturally-based theory of sentence comprehension. In *Proceedings of the 15th Annual Conference of the Cognitive Science Society* (pp. 108–113).

Luger, G. F., & Stubblefield, W. A. (1993). *Artificial intelligence: Structures and strategies for complex problem solving* (2nd ed.). Redwood City, CA: Benjamin Cummings.

Marr, D. (1982). *Vision*. New York: Freeman.

Minsky, M. (1975). A framework for representing knowledge. In: Winston, P. (Ed.) *The psychology of computer vision* (pp. 211–277). New York: McGraw-Hill.

Newell, A. (1980). Physical symbol systems. *Cognitive Science, 4,* 135–183.

Newell, A. (1982). The knowledge level. *Artificial Intelligence, 18,* 87–127.

Newell, A., & Simon, H. A. (1972). *Human problem solving*. Englewood Cliffs, NJ: Prentice Hall.

Norvig, P. (1992). *Paradigms of artificial intelligence programming: Case studies in COMMON LISP*. San-Mateo, CA: Morgan Kaufmann.

Pearl (1980). *Heuristics: Intelligent strategies for computer problem solving*. Reading, MA: Addison-Wesley.

Pereira, F., & Shieber, S. (1987). *PROLOG and natural-language analysis* (CSLI Lecture notes 10). Stanford, CA: Center for the Study of Language and Information.

Pereira, F. and D. Warren. (1980). Definite clause grammars for language analysis. *Artificial Intelligence, 13,* 231–278.

Pollard, C., & I. Sag. (1987). *Information-based syntax and semantics* (CSLI Lecture Notes 13). Stanford, CA: Center for the Study of Language and Information.

Pylyshyn, Z. (1989). Computing in cognitive science. In M.I. Posner (Ed.), *Foundations of cognitive science* (pp. 49–92). Cambridge, MA: MIT Press.

Quillian, M. R. (1968). Semantic memory. In M. Minsky (Ed.), *Semantic information processing* (p. 227–270). Cambridge, MA: MIT Press.

Ramsay, A. (1988). *Formal methods in artificial intelligence*. Cambridge: Cambridge University Press.

Rosenbloom, P. S., Laird, J. E., & Newell, A. (1993). *The SOAR papers: Research on integrated intelligence*. Cambridge, MA: MIT Press.

Schank, R. (1975). *Conceptual information processing*. Amsterdam: North-Holland.

Schank, R. (1980). Language and memory. *Cognitive Science, 4,* 243–284.

Schank, R. (1982). *Dynamic memory: A theory of reminding and learning in computers and people*. Cambridge: Cambridge University Press.

Schank, R., & Abelson, R. (1977). *Scripts, plans, goals and understanding: An inquiry into human knowledge structures*. Hillsdale, NJ: Erlbaum.

Sells, P. (1985). *Lectures on contemporary syntactic theories* (CSLI Lecture notes Nr. 3). Stanford, CA: Center for the Study of Language and Information.

Shapiro, S. (1992). *Encyclopedia of artificial intelligence* (2nd ed.). New York: Wiley.

Shieber, S. M. (1986). *An introduction to unification-based approaches to grammar (CSLI Lecture Notes 4)* Stanford, CA: Center for the Study of Language and Information.

Sowa, J. F. (1984). *Conceptual structures: Information processing in mind and machine*. Reading, MA: Addison-Wesley.

Sowa, J. F. (1991). Toward the expressive power of natural language. In J. F. Sowa (Ed.), *Principles of Semantic Networks* (pp. 157–190). Los Altos, CA: Morgan Kaufmann.

Steels, L. (1990). Components of Expertise. *AI Magazine,* 11(2), 29–49.

Stemberger, J., & MacWhinney, B. (1986). Frequency and the lexical storage of regularly inflected words. *Memory and Cognition, 14,* 17–26.

Vijay-Shanker, K., & Joshi, A. K. (1988). Feature structures based Tree Adjoining Grammars. In *Proceedings of the 12th International Conference on Computational Linguistics, Budapest* (pp. 83–86). Morristown, NJ: Association for Computational Linguistics.

Wall, R. (1972). *Introduction to mathematical linguistics*. Englewood Cliffs, NJ: Prentice Hall.

Webber, B. L., & Nilsson, N. J. (Eds.). (1981). *Readings in artificial intelligence*. Los Altos, CA: Morgan Kaufmann.

Winograd, T. (1983). *Language as a cognitive process*. Reading, MA: Addison-Wesley.

Winston, P. H. (1992). *Artificial intelligence* (3rd ed.). Reading, MA: Addison-Wesley.

Winston, P. H., & Horn, B. K. (1988). *Lisp* (3rd ed.). Reading, MA: Addison-Wesley.