19 NATURAL LANGUAGE GENERATION

hello, world
Kernighan & Ritchie, *The C Programming Language*

... you, MR KEITH V LINDEN, will be a millionaire January 31!
From a junk mailing

In one sense, language generation is the oldest subfield of language processing. When computers were able to understand only the most unnatural of command languages, they were spitting out natural texts. For example, the oldest and most famous C program, the "hello, world" program, is a generation program. It produces useful, literate English in context. Unfortunately, whatever subtle or sublime communicative force this text holds is produced not by the program itself but by the author of that program. This approach to generation, called **canned text**, is easy to implement, but is unable to adapt to new situations without the intervention of a programmer. CANNED TEXT

Language generation is also the most pervasive subfield of language processing. Who of us has not received a form letter with our name carefully inserted in just the right places, along with eloquent appeals for one thing or another. This sort of program is easy to implement as well, but I doubt if many are fooled into thinking that such a letter is hand-written English. The inflexibility of the mechanism is readily apparent when our names are mangled, as mine is in the junk

TEMPLATE FILLING

mailing shown above, or when other obvious mistakes are made.[1] This approach, called **template filling**, is more flexible than canned text and has been used in a variety of applications, but is still limited. For example, Weizenbaum's use of templates in ELIZA worked well in some situations, but produced nonsense in others.[2]

The success of simple generation mechanisms indicates that, to a first approximation, language generation is easier than language understanding. A language understanding system cannot generally control the complexity of the language structures it receives as input, while a generation system can limit the complexity of the structure of its output. Because of this, work in language processing initially focussed on language understanding, assuming that any generation that needed to be done could easily be handled with canned text or template filling mechanisms. Unfortunately, these simple mechanisms are not flexible enough to handle applications with any realistic variation in the information being expressed and in the context of its expression. Even the generation used in the limited domain of the "hello, world" program could use more flexibility. It might be more appropriate for the program to produce:

(1) Congratulations, you've just compiled and run a simple C program which means that your environment is configured properly.

This text is more complex than the original and we can see a number of potential variations. If the readers are experienced systems engineers, then we might choose not to congratulate them on compiling a program. Doing so might insult them. There are number of other ways of referring to the program in question, including "the program", "your first C program", "it" and "that lame excuse for code", each of which might or might not be appropriate in a given situation. Note also that we didn't need to conjoin "compiled and run" and we could have expressed the conclusion ("your environment is configured prop-

[1] A recent example of this arose when parents of a college-bound student received an acceptance letter that read "Congratulations on 987-65-4321's admission ... as a parent you will be a partner with the university in encouraging 987-65-4321 to succeed." (from the Associated Press, March 26, 1999).

[2] It's not difficult to "trick" ELIZA into producing nonsense, as shown in the following dialog with Eliza 4.3: HUMAN: "Let's talk just about you — not me. Can you think?" ELIZA: "You like to think I — not you — don't you" ... HUMAN: "I think you are an extremely stupid program." ELIZA: "Perhaps you would like to be an extremely stupid program" (Copeland, 1993, page 40).

erly") and then the evidence ("you've compiled and run a simple C program"). These variations are not necessarily synonymous. Language has evolved a variety of ways of expressing things, all of which are useful for expressing certain things in certain contexts. Therefore, to be useful in more complex environments, a generation system must be capable of: (1) producing an appropriate range of forms, and (2) choosing among those forms based on the intended meaning and the context. In this chapter, we will study the basic language generation techniques used to solve these problems, ignoring canned text and template-based mechanisms.

19.1 Introduction to Language Generation

Language understanding is somewhat like counting from one to infinity; language generation is like counting from infinity to one.
Yorick Wilks, quoted in (Dale, Eugenio, & Scott, 1998a, page 352)

Generation **from what?!**
attributed to Christopher Longuet-Higgins

NATURAL LANGUAGE GENERATION

Natural Language Generation (NLG) is the process of constructing natural language outputs from non-linguistic inputs. The goal of this process can be viewed as the inverse of that of **natural language understanding** (NLU) in that NLG maps from meaning to text, while NLU maps from text to meaning. In doing this mapping, generation visits many of the same linguistic issues discussed in the previous chapters, but the inverse orientation distinguishes its methods from those of NLU in two important ways.

NATURAL LANGUAGE UNDERSTANDING

First, the nature of the input to the generation process varies widely from one application to the next. Although the linguistic input to NLU systems may vary from one text type to another, all text is governed by relatively common grammatical rules. This is not the case for the input to generation systems. Each generation system addresses a different application with a different input specification. One system may be explaining a complex set of numeric tables while another may be documenting the structure of an object-oriented software engineering model. As a result, generation systems must extract the information necessary to drive the generation process.

Second, while both NLU and NLG must be able to represent a range of lexical and grammatical forms required for the application domain, their use of these representations is different. NLU has been characterized as a process of *hypothesis management* in which the linguistic input is sequentially scanned as the system considers alternative interpretations. Its dominant concerns include ambiguity, under-specification, and ill-formed input. These concerns are not generally addressed in generation research because they don't arise. The non-linguistic representations input to an NLG system tend to be relatively unambiguous, well-specified, and well-formed. In contrast, the dominant concern of NLG is *choice*. Generation systems must make the following choices:

- **Content selection** — The system must choose the appropriate content to express from a potentially over-specified input. The selection process is based on a specific communicative goal. For example, we noted that some of the content included in example 1 might not be appropriate for all readers. If the goal was to indicate that the environment is set up, and the reader was a systems engineer, then we'd probably express only the second clause.
- **Lexical selection** — The system must choose the lexical item most appropriate for expressing particular concepts. In example 1, for instance, it must choose between the word "configured" and other potential forms including "set up".
- **Sentence structure**
  - **Aggregation** — The system must apportion the selected content into phrase, clause, and sentence-sized chunks. Example 1 combined the actions of compiling and running into a single phrase.
  - **Referring expressions** — The system must determine how to refer to the objects being discussed. As we saw, the decision on how to refer to the program in example 1 was not trivial.

- **Discourse structure** — NLG systems frequently deal with multi-sentence discourse, which must have a coherent, discernible structure. Example 1 included two propositions in which it was clear that one was giving evidence for the other.

These issues of choice, taken together with the problem of actually putting linear sequences of words on paper, form the core of the field of NLG. Though it is a relatively young field, it has begun to develop a body of work directed at this core. This chapter will introduce this work. It will begin by presenting a simple architecture for NLG systems and will then proceed to discuss the techniques commonly used in the components of that architecture.

19.2 An Architecture for Generation

The nature of the architecture appropriate for accomplishing the tasks listed in the previous section has occasioned much debate. Practical considerations, however, have frequently led to the architecture shown in figure 19.1. This architecture contains two pipelined components:

- **Discourse Planner** – This component starts with a communicative goal and makes all the choices discussed in the previous section. It selects the content from the knowledgebase and then structures that content appropriately. The resulting discourse plan will specify all the choices made for the entire communication, potentially spanning multiple sentences and including other annotations (including hypertext, figures, etc.). DISCOURSE PLANNER
- **Surface Realizer** — This component receives the fully specified discourse plan and generates individual sentences as constrained by its lexical and grammatical resources. These resources define the realizer's potential range of output. If the plan specifies multiple-sentence output, the surface realizer is called multiple times. SURFACE REALIZER

This is by no means the only architecture that has been proposed for NLG systems. Other potential mechanisms include AI-style planning and blackboard architectures. Neither is this architecture without its problems. The simple pipeline, for example, doesn't allow decisions made in the planner to be reconsidered during surface realization. Furthermore, the precise boundary between planning and realization

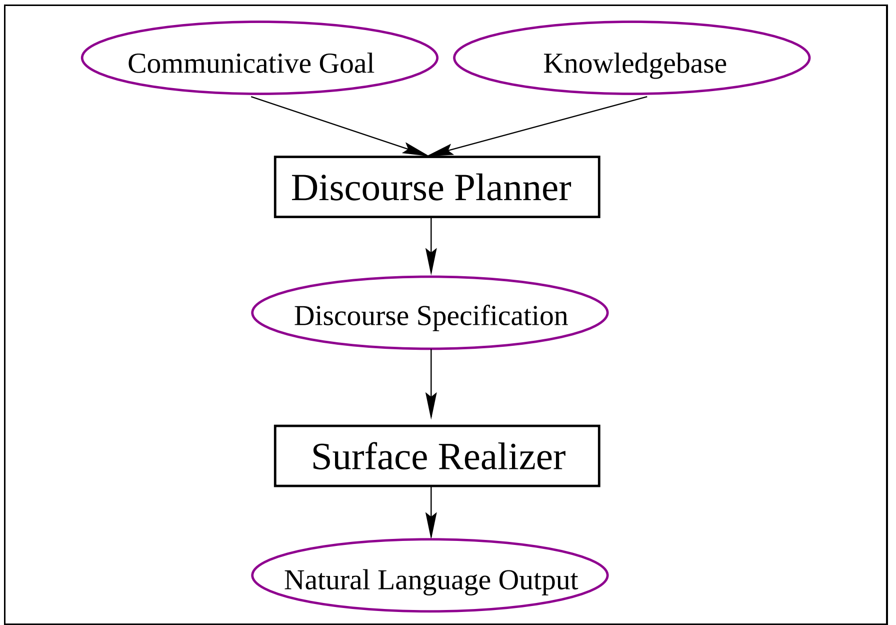


Figure 19.1 A reference architecture for NLG systems

overlap is not altogether clear. Nevertheless, we will use it to help organize this chapter. We'll start by discussing the surface realizer, the most developed of the two components, and then proceed to the discourse planner.

19.3 Surface Realization

The surface realization component produces ordered sequences of words as constrained by the contents of a lexicon and grammar. It takes as input sentence-sized chunks of the discourse specification. This section will introduce two of the most influential approaches used for this task: Systemic Grammar and Functional Unification Grammar. Both of these approaches will be used to generate the following example:

(2) The system will save the document.

There is no general consensus as to the level at which the input to the surface realizer should be specified. Some approaches specify only the propositional content, so in the case of example 2, the discourse plan would specify a saving action done by a system entity to a document entity. Other approaches go so far as to include the specification of the lexical items, in this case “save”, “system”, and “document”.

As we will see, systems using the two approaches discussed in this section take input at different levels. One thing they have in common, however, is that they take input that is functionally specified rather than syntactically specified. This fact, which is typical of generation systems, has tended to preclude the use of the syntactic formalisms discussed earlier in this book. Generation systems start with meaning and context, so it is most natural to specify the intended output in terms of **function** rather than of **form**. Example 2, for instance, could be stated in either active or passive form. Discourse planners tend not to work with these syntactic terms. They are more likely to keep track of the focus or topic of the discourse, and thus it is more natural to specify this distinction in terms of focus or topic. So in the example, if the document is the current topic of the discourse, it would be marked as the theme which could trigger the use of the passive. As we will see, both of the approaches discussed here categorize grammar in functional terms.

FUNCTION

FORM

Systemic Grammar

Systemic grammar was developed by Halliday as part of **Systemic-Functional linguistics**, a branch of linguistics that views language as a resource for expressing meaning in context. Systemic grammars represent sentences as collections of functions and maintain rules for mapping these functions onto explicit grammatical forms. This approach is well-suited to generation and has thus been widely influential in NLG. This section will start with an example of systemic sentence analysis. It will then discuss a simple systemic grammar and apply it to the running example.

SYSTEMIC-FUNCTIONAL LINGUISTICS

Systemic sentence analyses organize the functions being expressed in multiple “layers”, as shown in this analysis of example 2:

	The system	***will***	***save***	***the document***
Mood	subject	finite	predicator	object
Transitivity	actor	process		goal
Theme	theme	rheme		

Here, the mood layer indicates a simple declarative structure with subject, finite, predicator and object. The transitivity layer indicates that the "system" is the actor, or doer, of the process of "saving", and that the goal, or object acted upon, of the process is the "document". The theme layer indicates that the "system" is the theme, or focus of attention, of the sentence.

Notice that the three layers deal with different sets of functions. Mood deals with the subject, object, predicator (verb) and finite (auxiliary) of the sentence. Transitivity deals with the case roles of "actor" and "goal".[3] These indicate the roles that "the system" and "the document" play in the process of saving. Theme deals with the thematic and non-thematic elements of the sentence. This distinguishes the topic of the conversation (the theme) from other backgrounded elements (the rheme).[4] These three sets of functions, called **meta-functions**, represent three fundamental concerns in generation:

META-FUNCTIONS

INTERPERSONAL META-FUNCTION

- The **interpersonal meta-function** groups those functions that establish and maintain the interaction between the writer and the reader. It is represented here by the mood layer, which determines whether the writer is commanding, telling, or asking.

IDEATIONAL META-FUNCTION

- The **ideational meta-function** is concerned with what is commonly called the "propositional content" of the expression. Here, the transitivity layer determines the nature of the process being expressed and the variety of case roles that must be expressed. Note that this meta-function covers much of what is commonly termed "semantics".

TEXTUAL META-FUNCTION

- The **textual meta-function** is concerned with the way in which the expression fits into the current discourse. This includes issues of thematization and reference. In our example, the theme layer

[3] Case roles were developed by Fillmore (1968) and are discussed in Chapter 16.

[4] The concepts of theme and rheme were developed by the Prague school of linguistics (Firbas, 1966).

represents this in that it explicitly marks “the system” as the theme of the sentence.

This explicit concern for interpersonal and textual issues as well as traditional semantics is another feature of systemic linguistics that is attractive for NLG. Many of the choices that generation systems must make depend on the context of communication, which is formalized by the interpersonal and textual metafunctions.

A systemic grammar is capable of building a sentence structure such as the one just shown. The grammar is represented using a directed, acyclic, and/or graph called a **system network**. Figure 19.2 illustrates a simple system network. Here, the large curly brace indicates “and” (or parallel) systems, while the straight vertical lines represent “or” (or disjoint) systems. Although the system network formalism doesn’t require the use of systemic theory, we will loosely base this sample grammar on systemic categorizations. Thus, every clause (represented as the highest level feature on the far left) will simultaneously have a set of features for mood, transitivity and theme, but will either be indicative or imperative but not both. With respect to this grammar, example 2 is an indicative, declarative clause expressing an active material process with an unmarked theme.

SYSTEM NETWORK

A systemic grammar uses **realization statements** to map from the features specified in the grammar (e.g., Indicative, Declarative) to syntactic form. Each feature in the network can have a set of realization statements specifying constraints on the final form of the expression. These are shown in figure 19.2 as a set of italicized statements below each feature. Realization statements allow the grammar to constrain the structure of the expression as the system network is traversed. They are specified using a simple set of operators shown here:

REALIZATION STATEMENTS

$+X$ Insert the function X. For example, the grammar in figure 19.2 specifies that all clauses will have a predicator.

X/Y Conflate the functions X and Y. This allows the grammar to build a layered function structure by assigned different functions to the same portion of the expression. For example, active clauses conflate the actor with the subject, while passive clauses conflate the goal with the subject.

$X\hat{}Y$ Order function X somewhere before function Y. For example, indicative sentences place the subject before the predicator.

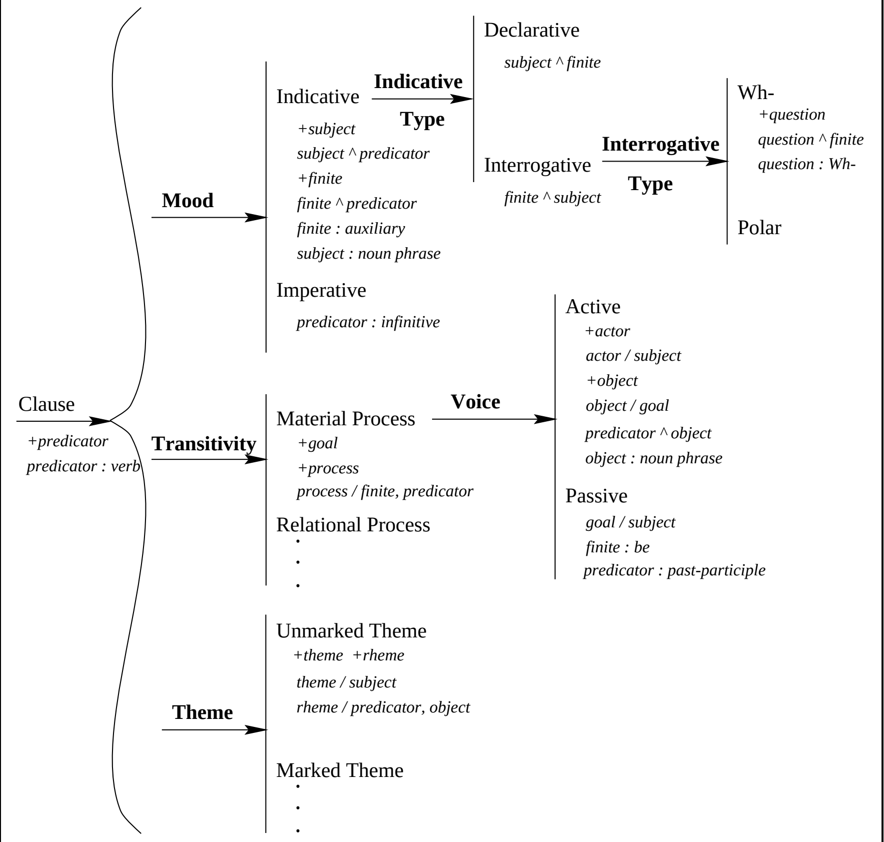


Figure 19.2 A simple systemic grammar overlap

$X : A$ Classify the function X with the lexical or grammatical feature A. These classifications signal a recursive pass through the grammar at a lower level. The grammar would include other networks similar to the clause network that apply to phrases, lexical items, and morphology. As an example, note that the indicative feature inserts a subject function that must be a noun phrase. This phrase will be further specified by another pass through the grammar.

$X!L$ Assign function X the lexical item L. Although none are shown in figure 19.2, the lexical items are all closed class items at this level of the grammar.

Given a fully specified system network, the procedure for generation is to:

1. Traverse the network from left to right, choosing the appropriate features and collecting the associated realization statements;
2. Build an intermediate expression that reconciles the constraints set by the realization statements collected during this traversal;
3. Recurse back through the grammar at a lower level for any function that is not fully specified;

To illustrate this process, we will use the sample grammar to generate example 2 ("The system will save the document"). We will use the following specification as input[5]:

```
(
  :process   save-1
  :actor     system-1
  :goal      document-1
  :speechact assertion
  :tense     future
  )
```

Here, the `save-1` knowledgebase instance is identified as the process of the intended expression. We will assume all knowledgebase objects to be KLONE-styled instances (Brachman, 1979) for which proper lexical entries exist. The actor and goal are similarly specified as `system-1` and `document-1` respectively. The input also specifies that the expression be in the form of an assertion in the future tense.

The generation process starts with the clause feature in figure 19.2, inserting a predicator and classifying it as a verb. It then proceeds to the mood system. Given that the input specifies an assertion, it chooses the indicative and declarative features. How these choices are made depends on the implementation of the system, but a common way is to associate a simple query or decision network with each choice point. This query or decision network accesses the relevant information from the input specification and from the knowledgebase, and makes the appropriate choice. The realization statements associated with the indicative and declarative features will insert subject and fi-

[5] This input specification is loosely based on the spl-constructor interface to the PENMAN system (Mann, 1983), a systemic generation system. The Sentence Planning Language (SPL), a more flexible input language, is discussed in the bibliographic notes below.

nite functions, and order them as subject then finite then predicator. The resulting function structure would be as follows:

Mood	subject	finite	predicator

We will assume that the `save-1` action is marked as a material process in the knowledgebase, which causes the transitivity system to choose the material process feature. This inserts the goal and process functions, and conflates the process with the finite/predicator pair. Because there is no indication in either the input or the knowledgebase to use a passive, the system chooses the active feature, which: (1) inserts the actor and conflates it with the subject, and (2) inserts the object, conflating it with the goal and ordering it after the predicator. This results in:

Mood	subject	finite	predicator	object
Transitivity	actor	process		goal

Finally, because there is no thematic specification in the input, the theme network chooses unmarked theme, which inserts theme and rheme, conflating theme with subject and conflating rheme with the finite/predicator/object group. This results in the full function structure discussed above (repeated here):

Mood	subject	finite	predicator	object
Transitivity	actor	process		goal
Theme	theme	rheme		

At this point, the generation process recursively enters the grammar a number of times at lower levels to fully specify the phrases, lexical items, and morphology. The noun phrase network will use a process like the one shown here to create "the system" and "the document". Systems in the auxiliary network will insert the lexical item "will". The choice of the lexical items "system", "document", and "save" can be handled in a number of ways, most typically by retrieving the lexical item associated with the relevant knowledgebase instances.

Although the example grammar is very simple, it does illustrate the expressiveness and utility of the systemic view of language for NLG.

Functional Unification Grammar

As discussed in Chapter 11, Functional Unification Grammar uses unification (discussed in Chapter 11) to manipulate and reason about feature structures. With a few modifications, this technique can be applied to NLG. The basic idea is to build the generation grammar as a feature structure with lists of potential alternations, and then to unify this grammar with an input specification built using the same sort of feature structure. The unification process then takes the features specified in the input and reconciles them with those in the grammar, producing a full feature structure which can then be *linearized* to form sentence output.

In this section we will illustrate this mechanism by generating example 2 again. We will use the simple functional unification grammar shown in figure 19.3. This grammar, expressed as an attribute-value matrix (cf. Chapter 11), supports simple transitive sentences in present or future tense and enforces subject-verb agreement on number. We'll now walk through the structure, explaining the features.

At its highest level, this grammar provides alternatives for sentences (cat s), noun phrases (cat np) and verb phrases (cat vp). This alternation is specified with the alt feature on the far left. We use the curly braces to indicate that any one of the three enclosed alternatives may be followed. This level also specifies a pattern that indicates the order of the features specified at this level, in this case, actor, process, then goal.

At the sentence level, this grammar supports actor, process, and goal features which are prespecified as NP, VP and NP respectively. Subject-verb agreement on number is enforced using the number feature inside the process feature. Here we see that the number of the process must unify with the path {actor number}. A path is a list of features specifying a path from the root to a particular feature. In this case, the number of the process must unify with the number of the actor. While this path is given explicitly, we can also have relative paths such as the number feature of the head feature of the NP. The path here, {$\uparrow\uparrow$ number}, indicates that the number of the head of the noun phrase must unify with the number of the feature 2 levels up. We'll see how this is useful in the example below.

The VP level is similar in nature to the NP level except that it has its own alternation between present and future tense. Given the

$$
\left[\text{alt}\left\{\begin{array}{l}
\begin{bmatrix}
\text{cat} & \text{s} \\
\text{actor} & \begin{bmatrix}\text{cat} & \text{np}\end{bmatrix} \\
\text{process} & \begin{bmatrix}\text{cat} & \text{vp} \\ \text{number} & \left\{\text{actor number}\right\}\end{bmatrix} \\
\text{goal} & \begin{bmatrix}\text{cat} & \text{np}\end{bmatrix} \\
\text{pattern} & \text{(actor process goal)}
\end{bmatrix} \\[2ex]
\begin{bmatrix}
\text{cat} & \text{np} \\
\text{head} & \begin{bmatrix}\text{cat} & \text{noun} \\ \text{number} & \left\{\uparrow\uparrow\text{number}\right\}\end{bmatrix} \\
\text{determiner} & \begin{bmatrix}\text{cat} & \text{article} \\ \text{lex} & \text{“the”}\end{bmatrix} \\
\text{pattern} & \text{(determiner head)}
\end{bmatrix} \\[2ex]
\begin{bmatrix}
\text{cat} & \text{vp} \\
\text{alt} & \left\{\begin{array}{l}
\begin{bmatrix}
\text{tense} & \text{present} \\
\text{head} & \begin{bmatrix}\text{cat} & \text{verb} \\ \text{number} & \left\{\uparrow\uparrow\text{number}\right\}\end{bmatrix} \\
\text{pattern} & \text{(head)}
\end{bmatrix} \\[2ex]
\begin{bmatrix}
\text{tense} & \text{future} \\
\text{auxiliary} & \begin{bmatrix}\text{cat} & \text{modal} \\ \text{lex} & \text{“will”}\end{bmatrix} \\
\text{head} & \begin{bmatrix}\text{cat} & \text{verb} \\ \text{ending} & \text{root}\end{bmatrix} \\
\text{pattern} & \text{(auxiliary head)}
\end{bmatrix}
\end{array}\right\}
\end{bmatrix}
\end{array}\right\}\right]
$$

Figure 19.3 A simple FUF grammar
overlap

tense, which we will see specified in the input feature structure, the unification will select the alternation that matches and then proceed to unify the associated features. If the tense is present, for example, the head will be single verb. If, on the other hand, the tense is future, we will insert the modal auxiliary “will” before the head verb.

This grammar is similar to the systemic grammar from the previous section in that it supports multiple levels that are entered recursively during the generation process. We now turn to the input feature structure, which specifies the details of the particular sentence we want to generate. The input structure, called a **functional description** (FD), is a feature structure just like the grammar. An FD for example 2 is as follows:

FUNCTIONAL DESCRIPTION

$$\begin{bmatrix} \text{cat} & \text{s} \\ \text{actor} & \begin{bmatrix} \text{head} & \begin{bmatrix} \text{lex} & \text{system} \end{bmatrix} \end{bmatrix} \\ \text{process} & \begin{bmatrix} \text{head} & \begin{bmatrix} \text{lex} & \text{save} \end{bmatrix} \\ \text{tense} & \text{future} \end{bmatrix} \\ \text{goal} & \begin{bmatrix} \text{head} & \begin{bmatrix} \text{lex} & \text{document} \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

Here we see a sentence specification with a particular actor, the system, and a particular goal, the document. The process is the saving of the document by the system in the future. The input structure specifies the particular verbs and nouns to be used as well as the tense. This differs from the input to the systemic grammar. In the systemic grammar, the lexical items were retrieved from the knowledgebase entities associated with the actor and goal. The tense, though not included in the example systemic grammar, would be determined by a decision network that distinguishes the relative points in time relevant to the content of the expression. This unification grammar, therefore, requires that more decisions be made by the discourse planning component.

To produce the output, this input is unified with the grammar shown in figure 19.3. This requires multiple passes through the grammar. The preliminary unification unifies the input FD with the “S” level in the grammar (i.e., the first alternative at the top level). The result of this process is as follows:

$$\begin{bmatrix} \text{cat} & \text{s} \\ \text{actor} & \begin{bmatrix} \text{cat} & \text{np} \\ \text{head} & \begin{bmatrix} \text{lex} & \text{system} \end{bmatrix} \end{bmatrix} \\ \text{process} & \begin{bmatrix} \text{cat} & \text{vp} \\ \text{number} & \left\{ \text{actor number} \right\} \\ \text{head} & \begin{bmatrix} \text{lex} & \text{save} \end{bmatrix} \\ \text{tense} & \text{future} \end{bmatrix} \\ \text{goal} & \begin{bmatrix} \text{cat} & \text{np} \\ \text{head} & \begin{bmatrix} \text{lex} & \text{document} \end{bmatrix} \end{bmatrix} \\ \text{pattern} & \text{(actor process goal)} \end{bmatrix}$$

Here we see that the features specified in the input structure have been merged and unified with the features at the top level of the grammar. For example, the features associated with “actor” include the lexical item “system” from the input FD and the category “np” from the grammar. Similarly, the process feature combines the lexical item and tense from the input FD with the category and number features from the grammar.

The generation mechanism now recursively enters the grammar for each of the sub-constituents. It enters the NP level twice, once for the actor and again for the goal, and it enters the VP level once for the process. The FD that results from this is shown in figure 19.4. There we see that every constituent feature that is internally complex has a pattern specification, and that every simple constituent feature has a lexical specification. The system now uses the pattern specifications to linearize the output, producing “The system will save the document.”

This particular example did not specify that the actor be plural. We could do this by adding the feature-value pair “number plural” to the actor structure in the input FD. Subject-verb agreement would then be enforced by the unification process. The grammar requires that number of the heads of the NP and the VP match with the number of the actor that was specified in the input FD. The details of this process are left as an exercise.

$$\begin{bmatrix}
\text{cat} & \text{s} \\
\text{actor} & \begin{bmatrix}
\text{cat} & \text{np} \\
\text{head} & \begin{bmatrix} \text{cat} & \text{noun} \\ \text{lex} & \text{system} \\ \text{number} & \left\{\uparrow\uparrow \text{number}\right\} \end{bmatrix} \\
\text{determiner} & \begin{bmatrix} \text{cat} & \text{article} \\ \text{lex} & \text{“the”} \end{bmatrix} \\
\text{pattern} & \text{(determiner head)}
\end{bmatrix} \\
\text{process} & \begin{bmatrix}
\text{cat} & \text{vp} \\
\text{number} & \left\{\text{actor number}\right\} \\
\text{head} & \begin{bmatrix} \text{cat} & \text{verb} \\ \text{lex} & \text{save} \\ \text{ending} & \text{root} \end{bmatrix} \\
\text{auxiliary} & \begin{bmatrix} \text{cat} & \text{modal} \\ \text{lex} & \text{“will”} \end{bmatrix} \\
\text{tense} & \text{future} \\
\text{pattern} & \text{(auxiliary root)}
\end{bmatrix} \\
\text{goal} & \begin{bmatrix}
\text{cat} & \text{np} \\
\text{head} & \begin{bmatrix} \text{cat} & \text{noun} \\ \text{lex} & \text{document} \\ \text{number} & \left\{\uparrow\uparrow \text{number}\right\} \end{bmatrix} \\
\text{determiner} & \begin{bmatrix} \text{cat} & \text{article} \\ \text{lex} & \text{“the”} \end{bmatrix} \\
\text{pattern} & \text{(determiner head)}
\end{bmatrix} \\
\text{pattern} & \text{(actor process goal)}
\end{bmatrix}$$

Figure 19.4 The fully unified FD
overlap

Summary

The two surface generation grammars we've seen in this section illustrate the nature of computational grammars for generation. Both used functional categorizations. One might wonder if it would be possible to use a single grammar for both generation and understanding. These grammars, called **bidirectional grammars**, are currently under investigation but have not found widespread use NLG (cf. Chapter 20). This is largely due to the additional semantic and contextual information required as input to the generator.

BIDIRECTIONAL GRAMMARS

19.4 Discourse Planning

The surface realization component discussed in the previous section takes a specified input and generates single sentences. Thus, it has little or no control over either the discourse structure in which the sentence resides or the content of the sentence itself. These things are controlled by the discourse planner. This section will introduce the two predominant mechanisms for building discourse structures: text schemata and rhetorical relations.

The focus on discourse rather than just sentences has been a key feature of much work done in NLG. Many applications require that the system produce multi-sentence or multi-utterance output. This can be done by simply producing a sentence for each component of the intended meaning, but frequently more care is required in selecting and structuring the meaning in an appropriate way. For example, consider the following alternate revision of the "hello, world" output discussed in the introduction:

(3) You've just compiled a simple C program. You've just run a simple C program. Your environment is configured properly.

These sentences are fine in isolation, but the text is more disjointed than the one given in example 1 and is probably harder to understand. Although it orders the sentences in a helpful way, it doesn't give any indication of the relationship between them. These are the sorts of issues that drive discourse planning.

This section will also discuss the closely related problem of content selection, which, as we saw earlier, is the process of selecting

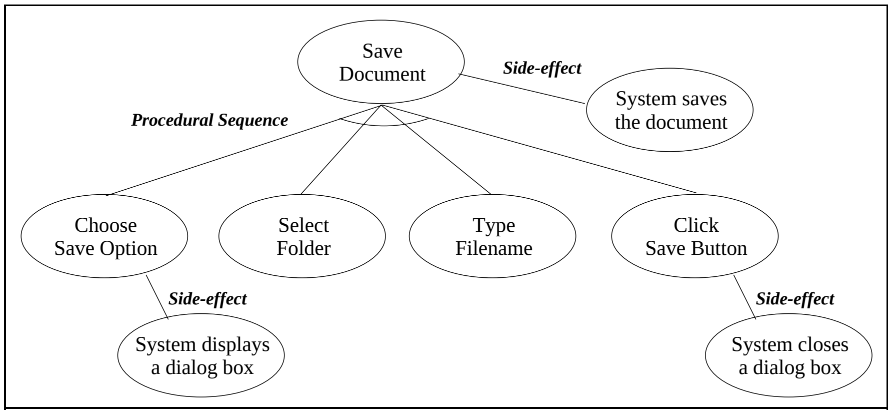


Figure 19.5 A portion of the saving procedure knowledgebase overlap

propositional content from the input knowledgebase based on a communicative goal. Because the form of this knowledgebase and the nature of the communicative goal varies widely from one application to another, it is difficult to make general statements about the content selection process. To make things more concrete, therefore, this section will focus on the task of generating instructions for a simple word-processing application. We'll assume that the knowledgebase, whatever its underlying structure, can be viewed as a KLONE-styled knowledgebase. We'll also assume that the communicative goal is to explain the represented procedure to a new user of the system. The knowledgebase will represent the procedure for saving a file as a simple procedural hierarchy, as shown in figure 19.5. The procedure specified there requires that the user choose the save option from the file menu, select the appropriate folder and file name, and then click on the save button. As a side-effect, the system automatically displays and removes the save-as dialog box in response to the appropriate user actions. This representation gives the procedural relationships between the basic actions but it doesn't show any of the domain knowledge concerning the structure of the interface (e.g., which choices are on which menus) or the particular entities that are used in the procedure (e.g., the document, the user). We'll assume that these are accessible in the knowledgebase as well.

Text Schemata

Apart from the rigidly structured canned texts and slot-filler templates discussed in the opening of this chapter, the simplest way to build texts is to key the text structure to the structure of the input knowledgebase. For example, we might choose to describe a game of tic-tac-toe or checkers by reviewing the moves in the sequence in which they were taken. This strategy soon breaks down, however, when we have a large amount of information that could potentially be expressed in order to achieve a variety of communicative goals. The knowledgebase the contains the fragment shown in figure 19.5, for example, could be expressed as a sequence of instructions such as one might find in a tutorial manual, or it could be expressed as an alphabetized set of program functions such as one might find in a reference manual.

One approach to this problem rests on the observation that texts tend to follow consistent structural patterns. For example, written directions explaining how to carry out an activity typically express the required actions in order of their execution. Any preconditions of these actions are mentioned before the actions themselves. Similarly, side-effects of these actions are mentioned after the actions themselves. In some domains, patterns such as these are rarely broken. Armed with this information, we can build a **schema** representing this structure, such as the one shown in figure 19.6. This schema is represented as an **augmented transition network** (ATN) in which each node is a state and each arc is an optional transition. Control starts in the small black node in the upper left and proceeds to follow arcs as appropriate until execution stops in the terminal node of the lower left. Node S0 allows the expression of any number of preconditions. Transitioning to S1 forces the expression of the action itself. S1 allows recursive calls to the network to express any sub-steps. The transition to S2 requires no action, and S2 allows any number of side-effects to be expressed before halting execution.

SCHEMA

AUGMENTED TRANSITION NETWORK

We can use this schema to plan the expression of the example procedure shown in figure 19.5. When the system is asked to describe how to save a document, the procedure schema can be activated. We'll assume that the knowledgebase specifies no preconditions for the action of saving a file, so we proceed directly to state S1, forcing the expression of the main action: "Save the document". In state S2, we recursively call the network for each of the four sub-steps specified

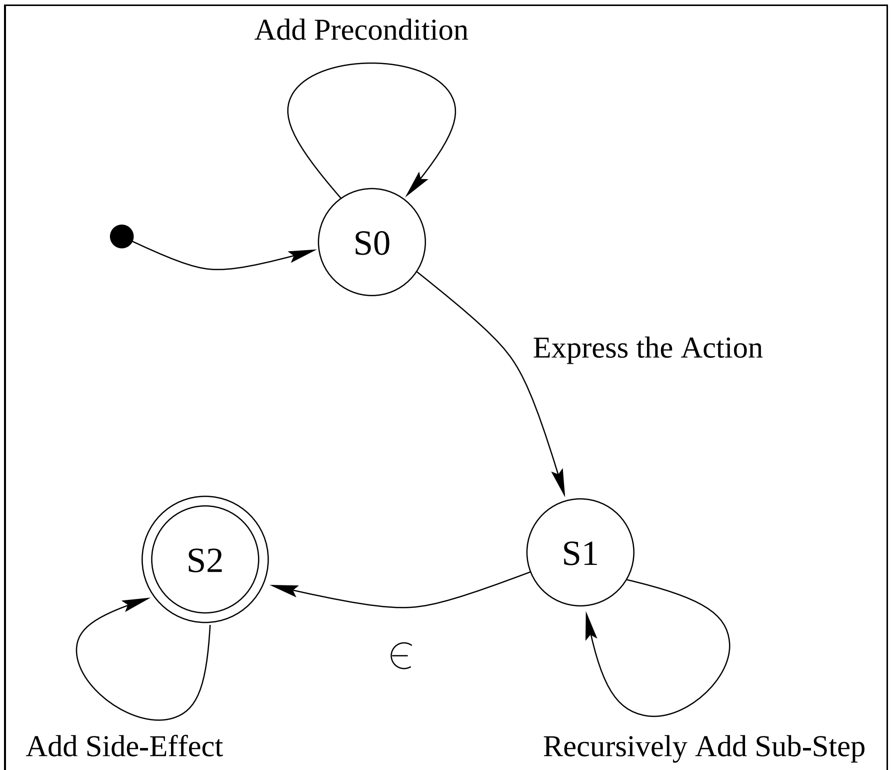


Figure 19.6 A schema for expressing procedures overlap

in the input. This expresses the first sub-step, "choose the save option", along with its side-effect, "this causes the system to display the save-as dialog box". The first sub-step has no preconditions or sub-steps. Each of the other sub-steps is done in the same manner and execution finally returns to the main action execution in step S2 which expresses the result of the whole process, "this causes the system to save the document" and then terminates. Depending on the details of the planning, the final text might be as follows:

> Save the document: First, choose the save option from the file menu. This causes the system to display the Save-As dialog box. Next, choose the destination folder and type the filename. Finally, press the save button. This causes

the system to save the document.[6]

Each one of these sentences can be generated using one of the surface realizers discussed in the previous section. As we can see, the schema mechanism is more flexible than templates or canned text. It structures the output according to known patterns of expression, but, with appropriate constraints, is able to insert optional material collected from the knowledgebase in a variety of orders. In addition, it is not required to express everything in the knowledgebase; the side-effect of the "click save button" action, for example, was not included.

This schema mechanism produced only a high-level discourse structure. The problem of specifying of the detailed form of each of the sentences, commonly called microplanning, is discussed in section 19.5.

Rhetorical Relations

Schemata are useful for discourse planning provided a discrete set of consistent patterns of expression can be found and encoded. However, they suffer from two basic problems. First, they become impractical when the text being generated requires more structural variety and richness of expression. For example, we may find that certain conditions dictate that we format our procedural instructions in a different manner. Some contexts may dictate that we explicitly enumerate the steps in the procedure, or that we express certain segments of the text in a different manner or in a different order. While in principle these variations could be supported either by adding constraints and operational code to the schema or by adding new schemata, the more variations that are required, the more difficult the schema-based approach becomes.

The second problem with schema-based mechanisms is that the discourse structure they produce is a simple sequence of sentence generation requests. It includes no higher-level structure relating the sentences together. In some domains, particularly in interactive ones (cf. Chapter 18), the structure of the previous discourse is relevant for future planning. For example, if we have explained a process in some detail, we might not want to do it again. It's easier to do these things when there is a record of the structure of previous discourse.

[6] This example is adapted from the McKeown (1990).

A useful approach here is to take a look under the hood of the schema in order to discover the more fundamental rhetorical dynamics at work in a text. A system informed by these dynamics could develop its own schemata based on the situations it confronts. A number of theories that attempt to formalize these rhetorical dynamics have been proposed, as discussed in some detail in Chapter 17. One such theory, **Rhetorical Structure Theory** (RST), is a descriptive theory of text organization based on the relationships that hold between parts of the text. As an example, consider the following two texts[7]:

RHETORICAL STRUCTURE THEORY

(4) I love to collect classic automobiles. My favorite car is my 1899 Duryea.

(5) I love to collect classic automobiles. My favorite car is my 1999 Toyota.

The first text makes sense. The fact that the writer likes the 1899 Duryea follows naturally from the fact that they like classic automobiles, after all, the Duryea is an example of a classic automobile. The second text is problematic. The problem is not with the individual sentences, they work perfectly well in isolation. The problem is rather with their combination. The fact that the two sentences are in sequence implies that there is some coherent relationship between them. In the case of the first text, that relationship could be characterized as one of elaboration. The second text could be characterized as one of contrast and would thus be more appropriately expressed as:

(6) I love to collect classic automobiles. However, my favorite car is my 1999 Toyota.

Here, the "however", overtly signals the contrast relation to the reader. RST claims that an inventory of 23 rhetorical relations, including ELABORATION and CONTRAST, is sufficient to describe the rhetorical structure a wide variety of texts. In practice, analysts tend to make use of a subset of the relations that are appropriate for their domain of application.

Most RST relations designate a central segment of text ("I love to collect..."), called the **nucleus**, and a more peripheral segment ("My favorite car is..."), called the **satellite**. This encodes the fact that many rhetorical relations are asymmetric. Here the second text is being interpreted in terms of the first, and not vice-versa. As we will

NUCLEUS

SATELLITE

[7] These texts are adapted from Mann and Thompson (Mann & Thompson, 1986).

see below, not all rhetorical relations are asymmetric. RST relations are defined in terms of the constraints they place on the nucleus, on the satellite, and on the combination of the nucleus and satellite. Here are definitions of some common RST relations:

ELABORATION — The satellite presents some additional detail concerning the content of the nucleus. This detail may be of many forms:

- a member of a given set
- an instance of a given abstract class
- a part of a given whole
- a step of a given process
- an attribute of a given object
- a specific instance of a given generalization

CONTRAST — The nuclei present things that, while similar in some respects, are different in some relevant way. This relation is **multi-nuclear** in that it doesn't distinguish between a nucleus and a satellite.

MULTI-NUCLEAR

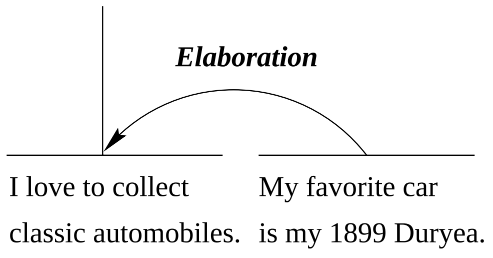
CONDITION — The satellite presents something that must occur before the situation presented in the nucleus can occur.

PURPOSE — The satellite presents the goal of performing the activity presented in the nucleus.

SEQUENCE — This relation is multi-nuclear. The set of nuclei are realized in succession.

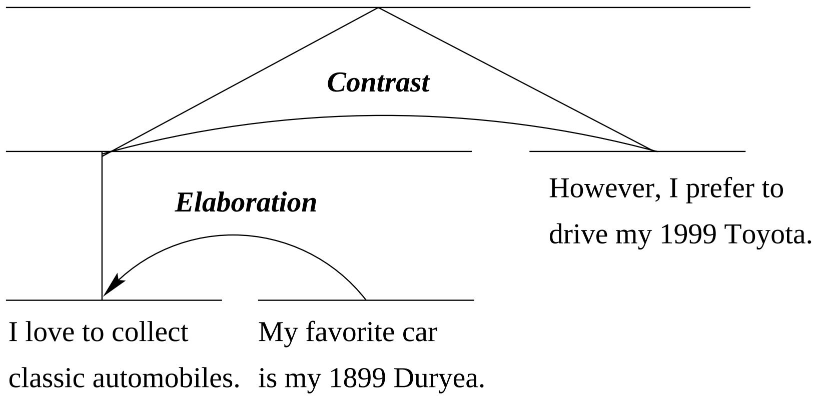
RESULT — The situation presented in the nucleus results from the one presented in the satellite.

RST relations are typically graphed as follows:


Elaboration
I love to collect classic automobiles.
My favorite car is my 1899 Duryea.

Here we see a graphical representation of the rhetorical relation from example 4. The segments of text are ordered sequentially along the bottom of the diagram with the rhetorical relations built above them. The individual text segments are usually clauses.

Rhetorical structure analyses are built up hierarchically, so we may use one pair of related texts as a satellite or nucleus in another higher-level relation. Consider the following three-sentence structure:


Contrast
Elaboration
However, I prefer to drive my 1999 Toyota.
I love to collect classic automobiles.
My favorite car is my 1899 Duryea.

Here we see that the first two clauses are related to one another via an elaboration relationship, and are related, as a pair, to the third clause via a contrast relationship. Note also how the multi-nuclear contrast relation is depicted. Recursive structuring such as this allows RST to build a single analysis tree for extended texts.

Although RST was originally proposed as a descriptive tool, it can also be used as a constructive tool for NLG. In order to do this, the rhetorical relations are typically recast as operators for an AI-style planner. As an example of this, we will look at a general-purpose, top-down, hierarchical planner that can be used for rhetorically-based text planning.[8]

The basic approach with this sort of planner is for the generation system to post a high level communicative goal stated in terms of the effect that the text should have on the reader. For our instructional text example, we will request that the planner build a structure to achieve the goal of making the reader competent to save a file. The highest level plan operator that achieves this goal will insert a rhetorical node appropriate for the goal and insert sub-goals for the nucleus and satellite of that rhetorical relation. These sub-goals will then be recursively expanded until the planning process reaches the bottom of the rhetorical structure tree, inserting a node that can be expressed

[8] This text planner is adapted from the work of Moore and Paris (1993).

as a simple clause.

For our example, we would post the goal:

(COMPETENT hearer (DO-ACTION <some-action>))

Here, the action would be some action in the knowledgebase, in this case, the root node from the procedural hierarchy shown in figure 19.5. A text plan operator that would fire for this goal would be as follows:

```
Name: Expand Purpose
Effect:
        (COMPETENT hearer (DO-ACTION ?action))
Constraints:
        (AND
              (c-get-all-substeps ?action ?sub-actions)
              (NOT (singular-list? ?sub-actions))
Nucleus:
        (COMPETENT hearer (DO-SEQUENCE ?sub-actions))
Satellites:
        (((RST-PURPOSE (INFORM s hearer (DO ?action)))
              *required*))
```

Note that the effect field of this operator matches the posted goal. An operator is applicable when its constraints hold. In this case, the main action ("?action") must have a non-unitary list of sub-actions. If this is true, the operator inserts a rhetorical purpose node into the discourse structure along with its satellite and nucleus. The satellite informs the hearer of the purpose of performing the main action, and the nucleus lists the sub-actions required to achieve this goal. Note that the effect, constraints, nucleus and satellite fields of the operator make use of variables (identifiers starting with "?") that are unified when the operator is applied. Thus, the goal action is bound to "?action" and can be accessed throughout the rest of the plan operator.

One other thing to notice about the plan operator is the way in which content selection is done. The constraint field specifies that there must be substeps and that there must be more than one of them. Determining whether the first constraint holds requires that the system retrieve the sub-steps from the knowledgebase. These sub-steps are then used as the content of the nucleus node that is constructed. Thus,

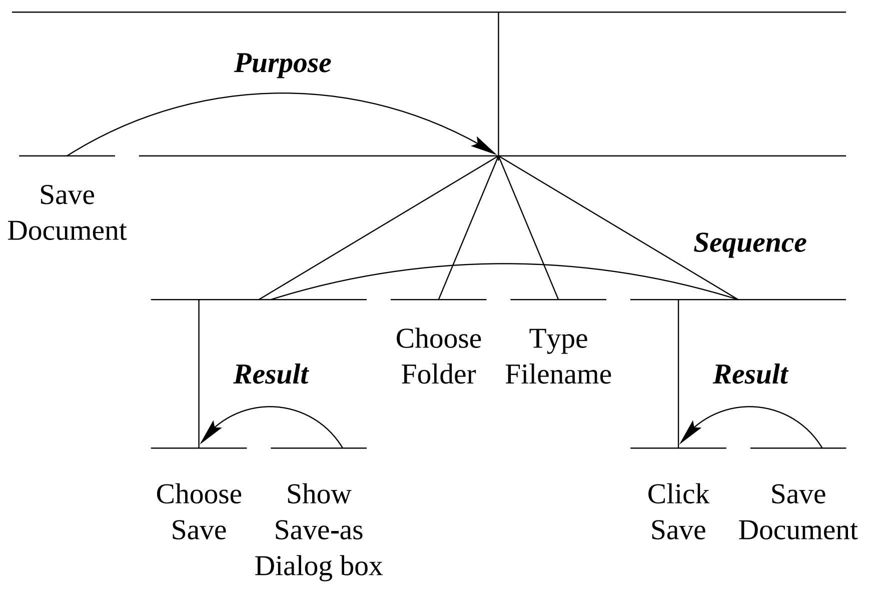


Figure 19.7 The full rhetorical structure for the example text overlap

the plan operators themselves do the content selection as required by the discourse planning process.

The full text structure produced by the planner is shown in figure 19.7. The root node of this tree is the node produced by the previous plan operator. The first nucleus node in figure 19.7 is the multi-nuclear node comprising all the sub-actions. The plan operator that produces this node is as follows:

Name: Expand Sub-Actions
Effect:
    (COMPETENT hearer (DO-SEQUENCE ?actions))
Constraints:
    NIL
Nucleus:
    (foreach ?actions (RST-SEQUENCE
        (COMPETENT hearer (DO-ACTION ?actions))))
Satellites:
    NIL

This operator achieves the nucleus goal posted by the previous operator. It posts a rhetorical node with multiple nuclei, one for each sub-action required to achieve the main goal. With an appropriate set of plan operators, this planning system could produce the discourse structure shown in figure 19.7, which could then be linearized into the following text:

To save a new file

1. Choose the save option from the file menu.
   The system will display the save-file dialog box.
2. Choose the folder.
3. Type the file name.
4. Click the save button.
   The system will save the document.

All of these sentences can be generated by a sentence generator. The last one, in particular, was identified as example 2 in the previous sections. As in the section on schema-based discourse planning, the problem of microplanning the sentences has been deferred to section 19.5.

Summary

In this section, we have seen how schema-based mechanisms can take advantage of consistent patterns of discourse structure. Although this approach has proven effective in the many contexts, it is not flexible enough to handle more varied generation tasks. Discourse planning based on rhetorical relations was introduced to add the flexibility required to handle these sorts of tasks.

19.5 Other Issues

This section introduces microplanning and lexical selection, two issues that were not discussed in detail in the previous sections.

Microplanning

The previous sections did not detail the process of mapping from the discourse plans described in the examples to the inputs to the surface realizers. The discourse structures, such as the one shown in figure 19.7, specified the high-level or macro structure of the text, but few of the details expected as input to the surface realizers. The problem of doing this more detailed planning is called **microplanning**.

MICROPLANNING

In most generation applications, microplanning is simply hardwired. For example, in instruction generation systems, all the user actions can be expressed as separate imperative sentences, which greatly simplifies the problem, but tends to produce monotonous texts such as the one shown in example 3. Two of the primary areas of concern in microplanning are the planning of **referring expressions** and **aggregation**.

REFERRING EXPRESSIONS
AGGREGATION

Planning a referring expression requires that we determine those aspects of an entity that should be used when referring to that entity in a particular context. If the object is the focus of discussion and has just been mentioned, we might be able to use a simple "it", whereas introducing a new entity may require more elaborate expressions like "a new document to hold your term paper". These issues are discussed in some detail in Chapter 17.

Aggregation is the problem of apportioning the content from the knowledgebase into phrase, clause, and sentence-sized chunks. We saw an example of this in the introduction where two of the actions mentioned in example 1 were conjoined within the first clause as "you've just *compiled and run* a simple C program". This is more readable than the non-aggregated version given in example 3 ("You've just compiled a simple C program. You've just run a simple C program").

Microplanning can be done by a separate module in the NLG architecture, placed between the discourse planner and the surface realizer. Indeed, more recent work has emphasized microplanning to the point that it is viewed as a task of importance equal to that of discourse planning and surface realization. It is also possible to add planning operators to the RST-based planning mechanism described in the chapter in order to perform microplanning tasks. However the microplanning is done, it serves to map from the output of the discourse planner to the input of the surface realizer.

Lexical Selection

Lexical selection refers to the general problem of choosing the appropriate words with which to express the chosen content. The surface realizers discussed in this chapter explicitly inserted closed-class lexical items as they were required, but deferred the choice of the content words to the discourse or micro planners. Many planners simplify this issue by associating a single lexical item with each entity in the knowledgebase.

Handling lexical selection in a principled way requires that the generation system deal with two issues. First, it must be able to choose the appropriate lexical item when more than one alternative exists. In the document-saving text from the previous section, for instance, the system generated "Click the save button". There are alternatives to the lexical item "click", including "hit" and "press mouse left on". The choice between these alternatives could consider: (1) style, in this case "hit" is perhaps more informal that "click", (2) collocation, in this case "click" probably co-occurs with buttons more often in this domain, and (3) user knowledge, in this case a brand new computer user might need the more fully specified "press mouse left on".

Second, the generation system must be able to choose the appropriate grammatical form for the expression of the concept. For example, the system could title the section "Saving a new file" rather than "To save a new file". This choice between the participle and the infinitive form is frequently made based on the forms most commonly employed in a corpus of instructions.

Evaluating Generation Systems

In early work on NLG, the quality of the output of the system was assessed by the system builders. If the output sounded good, then the system was judged a success. Because this is not a very effective test of system quality, much recent interest has been focussed on the rigorous evaluation of NLG systems. Several techniques have emerged.

One technique is to statistically compare the output of the generator with the characteristics of a corpus of target text. If the form chosen by the generator matches the form most commonly used in the corpus, it is judged as correct. The danger with this approach is that the corpus most often is produced by writers that may make errors. The assumption is that, as Tolstoy put it (Tolstoy, 1977), "All happy families are alike, but an unhappy family is unhappy after its own fashion." In other words, good text displays a consistent set of characteristics, while the characteristics of bad text will not statistically accumulate.

Another technique is to convene a panel of experts to judge the output of the language generator in comparison with text produced by human authors. In this variation of the Turing test, the judges do not know which texts were generated by the system and which were written by human authors. Computer generated text typically scores lower than human written text, but its quality approaches that of human authors in some restricted domains.

A final technique is to judge how effective the generated text is at achieving its goal. For example, if the text is intended to describe some object, its quality can be measured in terms of how well the reader does on a quiz given after reading the output text. If the text is intended to explain how to perform some process, its quality can be measured in terms of the number errors made by the reader after reading the text.

Generating Speech

This chapter has focussed on generating text rather than on generating speech. There are, however, many situations in which speech output is preferable if not absolutely necessary. These include situations where there is no textual display, such as when the user is using a telephone, and situations where the users are unable to look at a textual display, such as when the user is driving or when the user is disabled.

Given the existence of mechanisms for speech synthesis, as discussed in Chapter 4 and Chapter 5, it is tempting to simply hook up a speech synthesizer as the back end to a generation system. There are difficulties, however, with this approach. One such difficulty is the treatment of **homographs**. Consider the following example: HOMO-GRAPHS

(7) *Articulate* people can clearly *articulate* the issues.

Here, the two instances of the spelling "articulate" must be pronounced differently. Another, problem is the treatment of **prosody**, which requires that appropriate pitch contours and stress patterns be assigned to the speech being produced (cf. Chapter 9). In text-to-speech systems, these problems can be addressed by analyzing the input text. Homographs can frequently be distinguished using part-of-speech tagging. In the example of above, the adjective and verb forms of "articulate" are pronounced differently. Prosody, which is much harder, can occasionally be treated by distinguishing questions from non-questions, and by looking for commas and periods. In gen- PROSODY eral, however, it is not easy to extract the required information from the input text. A typical NLG system, on the other hand, can provide information on the parts of speech used and on the discourse structure of the output. This sort of information could prove useful in the treatment of homographs and prosody. To date, there has been very little work on this area in NLG.

19.6 Summary

Language Generation is the process of constructing natural language outputs from non-linguistic inputs. As a field of study, it usually does not include the study of simpler generation mechanisms such as **canned text** and **template filling**.

- Language generation differs from language understanding in that it focuses on linguistic **choice** rather than on resolving ambiguity. Issues of choice in generation include **content selection**, **lexical selection**, **aggregation**, **referring expression generation**, and **discourse structuring**.
- Language generation systems include a component that plans the structure of the discourse, called a **discourse planner**, and one that generates single sentences, called a **surface realizer**. Approaches for discourse planning include **text schemata** and **rhetorical relation planning**. Approaches for surface realization include **Systemic Grammar** and **Functional Unification Grammar**.
- **Microplanners** map the discourse planner output to the surface generator input, which includes the fine-grained tasks of **referring expression** generation, **aggregation**, and **lexical selection**.

19.7 Bibliographical and Historical Notes

Natural language generation, excluding canned text and template filling mechanisms, is a young field relative to the rest of language processing. Some minor forays into the field occurred in the 50's and 60's, mostly in the context of machine translation, but work focusing on generation didn't arise until the 70's. Simmons and Slocum's system (1972) used ATN's to generate discourse from semantic networks, Goldman's BABEL (1975) used decision networks to perform lexical choice, and Davey's PROTEUS (1979) produced descriptions of tic-tac-toe games. The 80's saw the establishment of generation as a distinct field of research. Influential contributions on surface realization were made by McDonald (1980) and the PENMAN project (Mann, 1983), and on text planning by McKeown (1985), and Appelt (1985). The 90's have seen continuing interest with the rise of generation-focussed workshops, both European and international, and organizations (cf. the Special Interest Group on language GENeration, http://www.aclweb.org/siggen).

As of the writing of this chapter, no textbooks on generation exist. However, a text on applied generation is in press (Reiter & Dale, 2000), and a number of survey papers have been written (Dale *et al.*, 1998a; Uszkoreit, 1996; McDonald, 1992; Bateman & Hovy, 1992; McKeown & Swartout, 1988). A number of these references discuss the history of NLG and its relationship to the rest of language processing. McDonald (1992) introduces the distinction between hypothesis management and choice.

Generation architectures have typically pipelined the tasks of planning and realization. The pipelining is used to constrain the search space within each of the modules and thus to make the generation task more tractable (Thompson, 1976; McDonald, 1988). However, these architectures have the well-known problem that decisions made by the discourse planner cannot easily be undone by the realizer (Meteer, 1992). Appelt's KAMP (1985) employed a unified architecture for planning and realization based on AI planning. This approach, however, has proven computationally impractical in larger domains. Blackboard architectures have also been proposed for language generation systems (Nirenburg, Lesser, & Nyberg, 1989). More recent work tends to include microplanning as an intermediate pipelined module in figure 19.1, placed between the discourse planner and the surface realizer (Reiter & Dale, 2000). The various concerns of microplanning itself have been the subject of considerable interest, including work on referring expressions (Appelt, 1985; Dale, 1992), aggregation (Mann & Moore, 1981; Dalianis, 1999), and other grammatical issues (Meteer, 1992; Vander Linden & Martin, 1995). The related issues of lexical selection (Goldman, 1975; Reiter, 1990; Stede, 1998) and tailoring the output text to particular audiences (Paris, 1993; Hovy, 1988a) have also received attention.

The late 80's and early 90's saw the construction of several reusable NLG systems, including two that have been distributed publicly: KPML (Bateman, 1997a, 1997b) and FUF (Elhadad, 1993). These tools can be downloaded through the SIGGEN web site. Most of this work was done in Lisp, but recent efforts have been made to port the systems to other languages and/or platforms.

Systemic functional linguistics (SFL) was developed by Halliday (1985). It has remained largely independent of generative linguistics and is relatively unknown in the language processing community as a whole. Attempts to use it in parsing have had limited success (O'Donnell, 1994; Kasper, 1988). However, it has had a deep impact on NLG, being used in one form or another by a number of generation systems, including Winograd's SHRDLU (1972), Davey's PROTEUS, Patten's SLANG (1988), PENMAN, and FUF. The example in this chapter is based in part on Winograd's (1972). SFL's most complete computational implementation is the Komet-Penman MultiLingual development environment (KPML), which is a descendent of PENMAN. KPML is packaged with NIGEL, a large English generation grammar, as well as an environment for developing multilingual grammars. It also includes a Sentence Planning Language (SPL) that forms a more usable interface to the systemic grammar itself. SPL specifications are considerably simpler to build than specifications that must include all the information required to make all the choices in the system network. Consider the following SPL specification:

```
(s1 / save
    :actor (a1 / system
               :determiner the)
    :actee (a2 / document
               :determiner the)
    :tense future
    )
```

The SPL interpreter will expand this into the series of feature choices required for the Nigel grammar to generate example 2 ("The system will save the document."). Each term in this specification gives the role of the entity (e.g., actor, actee) as well as the semantic type (e.g., save, system, document). The semantic types are KLONE-styled concepts subordinated to a general ontology (cf. Chapter 16) of concepts called the **upper model** (Bateman, Kasper, Moore, & Whitney, 1990). This ontology, which represents semantic distinctions that have grammatical consequences, is used by SPL to determine the type of entity being expressed and thus to reduce the amount of information explicitly contained in the SPL specification. This example leaves out the `:speechact assertion` term included in the example in the chapter; SPL uses this as a default value if left unspecified.

UPPER MODEL

Functional Unification Grammar was developed by Kay (1979). Its most influential implementation for generation is the Functional Unification Formalism (FUF) developed by Elhadad (Elhadad, 1992, 1993). It is distributed with the English grammar SURGE. Although the example given in the chapter used a simple phrase-structure approach to grammatical categorization (cf. (Elhadad, 1992)), the SURGE grammar uses systemic categorizations.

Another linguistic theory that has been influential in language generation in Mel'*c*uk's Meaning Text Theory (MTT) (1988). MTT postulates a number of levels ranging from deep syntax all the way to surface structure. Surface realizers that use it, including CoGenTex's REALPRO (Lavoie & Rambow, 1997) and ERLI's AlethGen (Coch, 1996b), start with the deep levels and map from level to level until they reach the surface level.

Discourse generation has been a concern of NLG from the beginning. Davey's PROTEUS, for example, produced paragraph-length summaries of tic-tac-toe games. His system structured its output based heavily upon the structure of the trace of the game which the application system recorded. Schema-based text structuring, pioneered by McKeown (1985), is more flexible and has been used in a number of applications (McCoy, 1985; Paris, 1993; Milosavljevic, 1999). The schema-based example presented in this chapter is based on the COMET instruction generation system (McKeown *et al.*, 1990). Although other theories of discourse structure (cf. Chapter 17) have influenced NLG, including theories by Grosz and Sidner (1986), Hobbs (Hobbs, 1979), and Kamp's DRT (Kamp, 1981), Rhetorical Structure Theory (RST), developed by Mann and Thompson (Mann & Thompson, 1987), has had the most influence (Hovy, 1988b; Scott & Souza, 1990). The RST-based planning example in this chapter is based on Moore and Paris' text planner (Moore & Paris, 1993) as it was used in the DRAFTER (Paris & Vander Linden, 1996; Paris et al., 1995) and ISOLDE (Paris, Vander Linden, & Lu, 1998) projects. The use of this planner in the context of an interactive dialog system is described by Moore and Paris (1993).

Applications of NLG tend to focus on relatively restricted sublanguages (cf. Chapter 20), including weather reports (Goldberg, Driedger, & Kittredge, 1994; Coch, 1998), instructions (Paris & Vander Linden, 1996; Paris *et al.*, 1998; Wahlster, André, Finkler, Profitlich, & Rist, 1993), encyclopedia-like descriptions (Milosavljevic, 1999; Dale, Oberlander, Milosavljevic, & Knott, 1998b), and letters (Reiter, Robertson, & Osman, 1999b). Output forms include simple text or hypertext (Lavoie, Rambow, & Reiter, 1997; Paris & Vander Linden, 1996), dynamically generated hypertext (Dale *et al.*, 1998b), multimedia presentation (Wahlster *et al.*, 1993), and speech (Van Deemter & Odijk, 1997). Information on a number of these systems is available on-line at the SIGGEN web site.

The evaluation of NLG systems has received much recent attention. Evaluations have assessed the similarity of the output with a representative corpus (Vander Linden & Martin, 1995; Yeh & Mellish, 1997), convened panels of experts to review the text (Lester & Porter, 1997; Coch, 1996a), and tested how effective the text was at achieving its communicative purpose (Reiter *et al.*, 1999b). It is also becoming more common for the usability of the NLG system itself to be evaluated.

Other issues of interest in NLG include the use of connectionist and statistical techniques (Ward, 1992; Langkilde & Knight, 1998), and the viability of multilingual generation as an alternative to machine translation (Hartley & Paris, 1997; Goldberg *et al.*, 1994).

Exercises

19.1 Use the systemic grammar given in the chapter to build a multiple-layer analysis of the following sentences:

a. The document will be saved by the system.
b. Will the document be saved by the system?
c. Save the document.

19.2 Extend the systemic grammar given in the chapter to handle the following sentences:

a. The document is large. (a relational process)
b. Give the document to Mary.
c. Is the document saved? (a polar interrogative)

19.3 Use the FUF grammar given in the chapter to build a fully unified FD for the following sentences:

a. The system saves the document.
b. The systems save the document.
c. The system saves the documents.

19.4 Extend the FUF grammar given in the chapter to handle the following sentences:

a. The document will be saved by the system. (i.e., the passive)
b. Will the document be saved by the system? (i.e., wh- questions)
c. Save the document. (i.e., imperative commands)

19.5 (Adapted from McKeown (1985)) Build an ATN appropriate for structuring a typical encyclopedia entry. Would it be in any way different from an ATN for a dictionary entry, and if so, could you adapt the same ATN for both purposes?

19.6 (Adapted from Bateman (1997b)) Build a system network for using mr, ms, mrs, miss in expressions like “Miss. Jones” and “Mr. Smith”. What information would the knowledgebase need to contain to make the appropriate choices in your network?

19.7 Do an RST analysis for the following text:

Temperature Adjustment
Before you begin, be sure that you have administrator access to the system. If you do, you can perform the following steps:

a. From the EMPLOYEE menu select the Adjust Temperature item. The system displays the Adjust Temperature dialog box.
b. Select the room. You may either type the room number or click on the appropriate room’s icon.
c. Set the temperature. In general you shouldn’t change the temperature too drastically.

d. Click the ok button. The system sets the room temperature.

By entering a desired temperature, you are pretending that you just adjusted the thermostat of the room that you are in.

The chapter lists a subset of the RST relations. Does it give you all the relations you need? How do you think your analysis would compare with the analyses produced by other analysts?

19.8 How does RST compare with Grosz and Sidner's theory of discourse presented in Chapter 17? Does one encompass the other or do they address different issues? Why do you think that RST has had a greater influence on NLG?

19.9 Would RST be useful for interactive dialog? If so, how would you use it? If not, what changes would you make to get it to work

19.10 (Adapted from ISOLDE (Paris *et al.*, 1998)) Speculate on how you would enhance an RST-based discourse planner to plan multimodal discourse, which would include diagrams and formatting (such as html formatting).

19.11 (Adapted from Reiter (1999a)). This chapter did not discuss template generators in any detail, it simply mentioned that they are easy to implement but inflexible. Try writing a simple template generator that produces persuasive letters addressed to people trying to convince them to stop smoking. The letter should include the standard elements of a letter as well as a discussion of the dangers of smoking and the advantages of quitting.

How flexible can you make the mechanism within the confines of template generation? Can you extend the system to take a case file on a particular patient that contains their medical history and produces a customized letter?

19.12 (Adapted from Milosavljevic (1999)). In the manner discussed in exercise 19.11, write a template generator that produces encyclopedia-like descriptions of animals.

19.8 References for the main bibliography

Appelt, D. E. (1985). *Planning English sentences.* Cambridge University Press, Cambridge.

Bateman, J. A. (1997a). Enabling technology for multilingual natural language generation: the KPML development environment. *Journal of Natural Language Engineering*, *3*(1), 15–55.

Bateman, J. A. (1997b). KPML Development Environment — Multilingual linguistic resource development and sentence generation release 1.1. Tech. rep., Institut für Integrierte Publikations- und Informationssysteme (IPSI), GMD, Darmstadt.

Bateman, J. A., & Hovy, E. H. (1992). An overview of computational text generation. In Butler, C. S. (Ed.), *Computers and Texts: An Applied Perspective*, pp. 53–74. Basil Blackwell, Oxford.

Bateman, J. A., Kasper, R. T., Moore, J. D., & Whitney, R. (1990). A general organization of knowledge for natural language processing: The Penman Upper Model. Tech. rep., USC/ISI.

Brachman, R. J. (1979). On the epistemological status of semantic networks. In Findler, N. V. (Ed.), *Associative Networks: Representation and Use of Knowledge by Computers.* Academic Press, New York.

Coch, J. (1996a). Evaluating and comparing three text-production techniques. In *the Proceedings of the 16th International Conference on Computational Linguistics,* August 5–9, Copenhagen, pp. 249–254.

Coch, J. (1996b). Overview of alethgen. In *Demonstration overview for the Proceedings of the Eighth International Workshop on Natural Language Generation,* Herstmonceux, England, 13–15 June 1996, pp. 25–28.

Coch, J. (1998). Interactive generation and knowledge administration in multimeteo. In *Proceedings of the 9th International Workshop on Natural Language Generation, Niagra-on-the-Lake, Ontario, CA, August 5–7*, pp. 300–303. System Demonstration.

Copeland, J. (1993). *Artificial Intelligence: A Philosophical Introduction.* Blackwell, Oxford.

Dale, R. (1992). *Generating Referring Expressions: Constructing Descriptions in a Domain of Objects and Processes.* MIT Press, Cambridge, MA.

Dale, R., Eugenio, B. D., & Scott, D. (1998a). Introduction to the special issue on natural language generation. *Computational Linguistics*, *24*(3), 345–353.

Dale, R., Oberlander, J., Milosavljevic, M., & Knott, A. (1998b). Integrating natural language generation and hypertext to produce dynamic documents. *Interactive with Computers*, *11*(2), 109–135.

Dalianis, H. (1999). Aggregation in natural language generation. *Journal of Computational Intelligence*, *15*(4).

Davey, A. (1979). *Discourse Production: A Computer Model of Some Aspects of a Speaker*. Edinburgh University Press.

Elhadad, M. (1992). *Using Argumentation to Control Lexical Choice: A Functional Unification-Based Approach*. Ph.D. thesis, Columbia University.

Elhadad, M. (1993). FUF: The universal unifier — User Manual, version 5.2. Tech. rep., Ben Gurion University of the Negev.

Fillmore, C. (1968). The case for case. In Bach, E., & Harms, R. (Eds.), *Universals in Linguistic Theory*, pp. 1–88. Holt-Rinehart-Winston.

Firbas, J. (1966). On defining the theme in functional sentence analysis. *Travaux Linguistiques de Prague*, *1*, 267–280.

Goldberg, E., Driedger, N., & Kittredge, R. (1994). Using natural-language processing to produce weather forecasts. *IEEE Expert*, *9*(2), 45–53.

Goldman, N. (1975). Conceptual generation. In Schank, R. C. (Ed.), *Conceptual Information Processing*, chap. 6. North-Holland.

Grosz, B. J., & Sidner, C. L. (1986). Attention, intentions, and the structure of discourse. *Computational Linguistics*, *12*(3), 175–204.

Halliday, M. A. K. (1985). *An Introduction to Functional Grammar*. Edward Arnold, London.

Hartley, A., & Paris, C. (1997). Multilingual document production from support for translating to support for authoring. *Machine Translation*, *12*, 109–128.

Hobbs, J. R. (1979). Coherence and coreference. *Cognitive Science*, *3*, 67–90.

Hovy, E. H. (1988a). *Generating Natural Language Under Pragmatic Constraints*. Lawrence Erlbaum Associates, Hillsdale, NJ.

Hovy, E. H. (1988b). Planning coherent multisentential text. In *Proceedings of the Twenty-Sixth Annual Meeting of the Association for Computational Linguistics*. State University of New York, Buffalo, NY, USA, June 7-10.

Kamp, H. (1981). A theory of truth and semantic representation. In Groenendijk, J. A. G., Janssen, T. M. V., & Stokhof, M. B. J. (Eds.), *Formal Methods in the Study of Language*, Vol. 1, pp. 277–322. Mathematisch Centrum, Amsterdam.

Kasper, R. (1988). An experimental parser for systemic grammars. In *Proceedings of the 12th International Conference on Computational Linguistics,* August 22–27, Budapest, Hungary. Association for Computational Linguistics.

Kay, M. (1979). Functional grammar. In *Proceedings of the 5th Annual Meeting of the Berkeley Linguistics Society, 17–19 February, Berkeley, CA*.

Langkilde, I., & Knight, K. (1998). The practical value of n-grams in generation. In *Proceedings of the Ninth International Workshop on Natural Language Generation,* Niagara-on-the-Lake, Ontaria, Canada, 5–7 August 1998, pp. 248–255.

Lavoie, B., & Rambow, O. (1997). A fast and portable realizer for text generation systems. In *Proceedings of the Fifth Conference on Applied Natural Language Processing, Washington DC.*

Lavoie, B., Rambow, O., & Reiter, E. (1997). Customizable descriptions of object-oriented models. In *Proceedings of the Fifth Conference on Applied Natural Language Processing, Washington DC*, pp. 265–268.

Lester, J., & Porter, B. (1997). Developing and empirically evaluating robust explanation generators: The KNIGHT experiments. *Computational Linguistics*, *23*(2), 65–101.

Mann, W., & Moore, J. (1981). Computer generation of multiparagraph text. *American Journal of Computational Linguistics*, *7*(1), 17–29.

Mann, W. C. (1983). An overview of the PENMAN text generation system. In *Proceedings of the National Conference on Artificial Intelligence*, pp. 261–265. AAAI. Also appears as USC/Information Sciences Institute, RR-83-114.

Mann, W. C., & Thompson, S. A. (1986). Relational propositions in discourse. *Discourse Processes*, *9*(1), 57–90.

Mann, W. C., & Thompson, S. A. (1987). Rhetorical structure theory: A theory of text organization. Tech. rep., USC Information Sciences Institute.

McCoy, K. F. (1985). *Correcting Object-Related Misconceptions.* Ph.D. thesis, University of Pennsylvania.

McDonald, D. D. (1980). *Natural Language Production as a Process of Decision Making.* Ph.D. thesis, MIT, Cambridge, MA.

McDonald, D. D. (1988). Modularity in natural language generation: Methodological issues. In *Proceedings of the AAAI Workshop on Text Planning and Realization*, pp. 91–98.

McDonald, D. D. (1992). Natural-language generation. In Shapiro, S. C. (Ed.), *Encyclopedia of Artificial Intelligence* (2nd edition). John Wiley, New York.

McKeown, K. R. (1985). *Text Generation.* Cambridge University Press, New York.

McKeown, K. R., *et al.* (1990). Natural language generation in COMET. In Dale, R., Mellish, C., & Zock, M. (Eds.), *Current Research in Natural Language Generation*, chap. 5. Academic Press.

McKeown, K. R., & Swartout, W. R. (1988). Language generation and explanation. In Zock, M., & Sabah, G. (Eds.), *Advances in Natural Language Generation - An Interdisciplinary Perspective*, Vol. 1, chap. 1. Ablex, Norwood, NJ.

Mel'cuk, I. A. (1988). *Dependency Syntax: Theory and Practice.* SUNY Series in Linguistics, Mark Aronoff, series editor. State University of New York Press, Albany.

Meteer, M. W. (1992). *Expressibility and the problem of efficient text planning.* Pinter, London.

Milosavljevic, M. (1999). *Maximising the Coherence of Descriptions via Comparison.* Ph.D. thesis, Macquarie University.

Moore, J. D., & Paris, C. L. (1993). Planning text for advisory dialogues: Capturing intentional and rhetorical information. *Computational Linguistics*, *19*(4), 651–694.

Nirenburg, S., Lesser, V., & Nyberg, E. (1989). Controlling a language generation planner. In *the Proceedings of the International Joint Conference on Artificial Intelligence*, pp. 1524–1530.

O'Donnell, M. J. (1994). *Sentence Analysis and generation: A Systemic Perspective.* Ph.D. thesis, University of Sydney.

Paris, C. (1993). *User Modelling in Text Generation.* Fances Pinter.

Paris, C., *et al.* (1995). A support tool for writing multilingual instructions. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence,* August 20–25, Montréal, Canada, pp. 1398–1404. Also available as ITRI report ITRI-95-11.

Paris, C., & Vander Linden, K. (1996). Drafter: An interactive support tool for writing multilingual instructions. *IEEE Computer*, *29*(7), 49–56. (Special Issue on Interactive Natural Language Processing).

Paris, C., Vander Linden, K., & Lu, S. (1998). Automatic document creation from software specifications. In Kay, J., & Milosavljevic, M. (Eds.), *Proceedings of the 3rd Australian Document Computing Symposium (ADCS98), Sydney, August.*

Patten, T. (1988). *Systemic Text Generation as Problem Solving.* Cambridge University Press.

Reiter, E. (1990). A new model for lexical choice for open-class words. In McKeown, K. R., Moore, J. D., & Nirenburg, S. (Eds.), *Proceedings of the Fifth International Workshop on Natural Language Generation,* June 3–6, Dawson, PA, pp. 23–30.

Reiter, E., & Dale, R. (2000). *Building Natural Language Generation Systems.* Cambridge University Press, Cambridge.

Reiter, E., Robertson, R., & Osman, L. (1999a). Types of knowledge required to personalise smoking cessation letters. In *Proceedings of the Joint European Conference on Artificial Intelligence in Medicine and Medical Decision Making.* Springer-Verlag. To appear.

Reiter, E., Robertson, R., & Osman, L. (1999b). Types of knowledge required to personalise smoking cessation letters. In *Proceedings of the Joint European Conference on Artificial Intelligence in Medicine and Medical Decision Making.* Springer-Verlag.

Scott, D. R., & Souza, C. (1990). Getting the message across in RST-based text generation. In Dale, R., Mellish, C., & Zock, M. (Eds.), *Current Research in Natural Language Generation*, chap. 3. Academic Press.

Simmons, R., & Slocum, J. (1972). Generating English discourse from semantic networks. *Communications of the ACM*, *15*(10), 891–905.

Stede, M. (1998). A generative perspective on verb alternations. *Computational Linguistics*, *24*(3), 401–430.

Thompson, H. (1976). Towards a model of language production: Linguistic and computational foundations. *Statistical Methods in Linguistics.*

Tolstoy, L. (1977). *Anna Karanin.* Penquin Classics. Translated by Rosemary Edmonds.

Uszkoreit, H. (Ed.). (1996). *Language Generation*, chap. 4. available at: http://cslu.cse.ogi.edu/HLTsurvey/.

Van Deemter, K., & Odijk, J. (1997). Context modeling and the generation of spoken discourse. *Speech Communication*, *21*(1/2), 101–121.

Vander Linden, K., & Martin, J. H. (1995). Expressing local rhetorical relations in instructional text: A case-study of the purpose relation. *Computational Linguistics*, *21*(1), 29–57.

Wahlster, W., André, E., Finkler, W., Profitlich, H.-J., & Rist, T. (1993). Plan-based Integration of Natural Language and Graphics Generation. *Artificial Intelligence, Special Volume on Natural Language Processing*, *63*(1–2), 387–428.

Ward, N. (1992). *A Connectionist Language Generator*. Ablex.

Winograd, T. (1972). *Understanding Natural Language*. Academic Press, New York.

Yeh, C.-L., & Mellish, C. (1997). An empirical study on the generation of anaphora in Chinese. *Computational Linguistics*, *23*(1), 169–190.