Locally Organized Text Generation

Penelope Sibun

COINS Technical Report 91-73

Department of Computer and Information Science
University of Massachusetts

Xerox Palo Alto Research Center
3333 Coyote Hill Road
Palo Alto, CA 94304

sibun@parc.xerox.com

This report is a slightly edited version of a dissertation submitted to the Graduate School of the University of Massachusetts in partial fulfillment of the requirements for the degree of Doctor of Philosophy in the Department of Computer and Information Science, September 1991. Much of the research described here was completed while the author was at the Xerox Palo Alto Research Center.

Abstract

In this thesis I present an architecture for generating extended text. This architecture is implemented in a system, Salix, which incrementally generates natural language texts whose structure is derived from the domain structure of the subject matter. The architecture is composed of data driven domain independent strategies for producing increments of text and metastrategies that combine or choose among all strategies that are applicable at each increment or decide what to do if no strategy applies. Salix's capabilities are demonstrated in generating texts, in the domains of houses and families, that are comparable to descriptions elicited from human speakers. Generating texts describing physical processes has also been explored. The approach to text generation presented here is compared to others in the literature along the dimensions of local organization, incremental generation, coherence, focusing, and domain independence. An argument is made for the approach presented here that locally organizes and incrementally generates coherent texts.

Acknowledgments

I would like to acknowledge and thank those who enabled the completion of this thesis in a variety of ways.

I start with thanking my family. My parents, John and Kitsy Sibun, have been there from the beginning, applauding and refraining from asking too often if I would ever get done. My sister, Barbara Sibun, has always been there too, especially, fortunately, at moving times. While shirking many of her feline duties, the Tudes has provided comfort and company when I've been too incoherent for the humans.

I thank the graduate students I have known over the years, especially for providing me with speech error data. Brad Blumenthal I have known from undergraduate days, and we haven't stopped our friendship and intellectual engagement though we have lived in different states and on different continents. Some of my fellow gradstuds were especially important during my years at the University of Massachusetts. I believe I will never enjoy coauthorship with anyone as much as with Scott Anderson, with whom I would cheerfully wrangle over every word. I am glad to have had the opportunity to collaborate with Al Huettner on linguistic analyses, some of which have made their way into this thesis. David Forster and I shared office space for many years, and he displayed admirable patience and expertise the many times I wound up swearing at the Lisp machine. Dan Neiman has always been good for an argument and performed the supreme act of friendship in helping me drive my car across country. As our research paths have slowly converged over the years, David Lewis has become both a friend and colleague.

Several faculty members at UMass deserve special thanks. Just as I was starting work on my thesis, it was disrupted by the departure of my advisor. Bev Woolf stepped in with a project to employ us orphans and James Pustejovsky kept me in touch with computational linguistics through biweekly meetings at the Black Sheep. Victor Lesser agreed to chair my thesis and provided guidance through the entire process. Allen Hanson and Robbie Moll brought to my committee different perspectives and challenged me to view my work in different lights. Roger Higgins gently reminded me of what linguists would think of what I was saying.

I thank many of the computational linguists at the University of Pennsylvania, which for a year was my intellectual home away from home, which I visited monthly by train. Bonnie Webber has been an important member of my thesis committee, and her constant, cheerful belief in me and continued enthusiasm for my ideas have been crucial. Many students at Penn provided me with warm hospitality and stimulating discussion, especially Barbara di Eugenio, Lyn Walker, Owen Rambow, and Libby Levinson.

Trekking west a year before finishing this thesis leads to my thanking a great many Californians. Kris Halvorsen was instrumental in this move, by providing me with a Summer Internship at Xerox PARC, and then employing me part-time so that I could finish my thesis in California. The summer was highlighted by meetings of Lingua Dranca, which, in addition to people mentioned elsewhere,

included Steve Whittaker, Susan Newman, and the horses on Coyote Hill. Jeanette Figueroa's kindness and ebullience has been a constant source of cheer, and her administrative competence has been never-failing. Marti Hearst has been an exemplary officemate, and a highly capable commenter on my prose.

I particularly thank those who provided me with human contact as I all but disappeared into work. Bernardo Huberman once read my mind at a crucial juncture, and has occupied me with his stories since. Randomly, Jeff Shrager's path has occasionally intersected mine. Ramin Zabih has kept me sane via chatter by the lights of the Connection Machine and movies of dubious quality. During the long winter of endless 12-hour days at PARC, John Batali was on the other side of the wall working on his thesis too, and on that day in January he told me first that "they've started." Gregor Kiczales provided great motivation to finish by constantly calling me "worker bee," and Jill Steinberg was a reminder that there are nontechies in the world.

I thank the friends in the two activities that punctuated my work mode at PARC. Lunchtime bridge has provided an opportunity to wrap my mind around something different for a while, and this was made possible by many partners/opponents, especially the regulars Marshall Bern, Aaron Goodisman, Barry Hayes, Pavel Curtis, and Natalie Glance. Lori Hannigan has led a stimulating aerobics class, and Russ Atkinson, Jock Mackinlay, Polle Zellweger, and Anne Chiang have been a core group of companions in sweat and endorphins.

Most I thank David Chapman.

Contents

List of Figures

Chapter 1

Introduction

In this thesis I present an architecture for generating extended text. This architecture is implemented in a system, Salix, which incrementally generates natural language texts whose structure is derived from the domain structure of the subject matter. The architecture is composed of data driven domain independent strategies for producing increments of text and metastrategies that combine or choose among all strategies that are applicable at each increment. The approach I present here differs from most approaches to text generation along several dimensions: the process of generation in Salix is a unified one and not divided into distinct components; the text is locally organized and generated incrementally, rather than generated top-down by a hierarchical planner; the text is structured by domain structure rather than imposed rhetorical structure; focusing—what to say next and when to use pronouns—is accomplished without additional, special-purpose architecture; and Salix's architecture achieves domain independence in a data driven manner.

Salix's capabilities are demonstrated in generating texts, in the domains of houses and families, that are comparable to descriptions elicited from human speakers. Salix also generates texts about physical processes. Figure 1.1 shows an example of text that Salix produces; figure 1.2 is a sketch of the house that is the subject of the text.

The research in this thesis has focused on a particular sort of text. People use language in a variety of ways, and computational linguists who study natural language generation have developed a variety of models of the process and a variety of approaches to the issues involved. No one model is yet comprehensive and work to date has addressed particular kinds of text production that are constrained in certain ways; the work reported in this thesis is no exception.

One set of constraints brought to bear on a text is the constraints of a multi-speaker dialogue. In this setting any speaker is very much constrained by the fact that she does not always have control over whether she gets to speak, and in general she is constrained by the rules of conversation which she can exploit or flout to get things said. For discussion of text constrained by the rules of conversation see Levinson (1983) and Luff *et al.* (1990).

A somewhat different set of constraints arises in texts that are organized around a task, such as building a water pump (Grosz & Sidner 1986) or specifying a plan for action (Linde & Goguen 1978). These tasks are often represented by plans and the plan structure constrains the text for the conversational participants.

Some texts are structured less by the interactions among speakers: these texts have more internal structure. Examples include forming an argument, writing a letter (Mann & Thompson 1987), telling a story (Hobbs 1985), and explaining a concept (Moore & Swartout 1989, Cawsey 1990).

```
OK
we can start at the side door
and then there's a entrance hallway
then
in the kitchen
there is a window
which is large
and is a picture window
it has two flanking windows
and faces the backyard
and if we're facing that window
on the right is the sliding glass door
and a window
which is small
if we're facing the backyard
on the left is the stove
and a refrigerator
and if we're facing that window
underneath is the sink
then a dishwasher
then there's a living room
it has
a window
which is large
in it
and then there's a short hallway
and
Ann's bedroom on the left
Claire's bedroom on the right
and
a bathroom in the middle
oh yeah
and then there's Penni's bedroom
and a long hallway
that's it
```

Figure 1.1: House description generated by Salix.

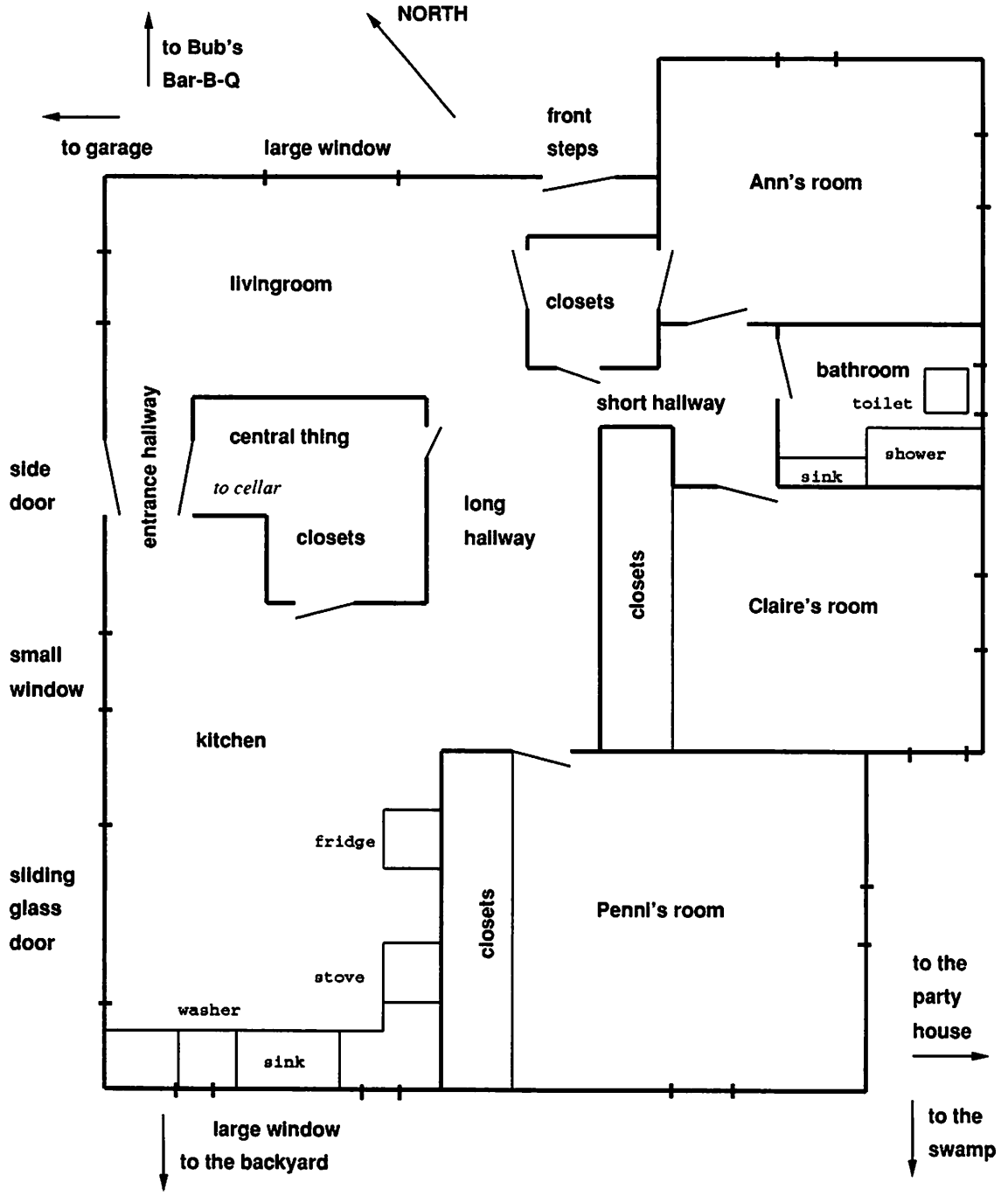

Figure 1.2: Sketch of house.

Many of these texts have a rhetorical aspect, and to the extent that they do, rhetorical structure constrains the structure of the text.

Finally, some texts are mostly *about* something—about the travails of one's day, the idiosyncracies of one's relatives, or the workings of one's car. For such texts, the major source of constraints is the subject matter itself; the structure of the text reflects the structure of the domain. It is the building of this sort of text that I address in this work.

The various types of text characterized above do not form mutually exclusive categories: any extended text, whether spoken or written will look sometimes more like one type and sometimes more like another.[1] Even more importantly, in general, text gets its structure from many sources, and a complete model of how people structure the language they use and how systems can build similar texts will account for all these sources of structure.

In this first chapter, I describe natural language generation and sketch the state of the art in approaches to this problem. I then give an overview of the issues in generation that are addressed in this thesis. The approach to text generation presented here is compared to others in the literature along the dimensions of incremental generation, local organization, coherence, focusing, and domain independence. Salix, the implemented architecture that embodies the approach taken to these issues, is introduced. I present an example of the text generated by Salix (see figure 1.1) and describe how the system operates to produce this text. I conclude this chapter with an overview of the rest of the thesis.

1.1 Natural Language Generation

Natural language generation is the process of producing text. Text is any segment of language, brief or extended, written or spoken. Generation is a process of making choices. These choices range in granularity from choices of words and syntactic structures to choices of what to say and not to say. A generation system's success is judged by the text it can produce; indeed, the goal of generation can be seen as a version of the Turing test—building a system that can speak (or write) like a human.

The study of natural language generation is that branch of computational linguistics, and, more broadly, artificial intelligence (AI) research, concerned with enabling computer systems to produce natural language. Generation applications are many: people are used to conducting transactions in natural language, and are comfortable when they can use this means of communication with machines. Thus, we would like to be able to equip systems that respond to database queries, advise, explain, give directions, and teach with the ability to converse with their users. In addition, we would like to enable our systems to write reports and papers and translate documents in a way that makes sense to humans. The field as a whole is working to answer the questions involved in making machines language users. While not addressing all of the multitudinous issues involved in generating natural language, the research presented here will provide a deeper understanding of how extended coherent text is structured.

Many sorts of choices occur in the process of producing text. The process must start with some "urge to speak"—a reason for engaging in this activity. Natural language generation is a complex process that involves many different issues. These issues include: choosing knowledge to be conveyed; achieving particular goals such as getting someone to do something or satisfactorily

[1] Written text is not immune to other conversational participants; most of the electronic communications supported by computers have conversational characteristics. Conversely, spoken texts which involve only one speaker often occur and can be quite long.

answering his question (there may also be more subtle pragmatic goals such as being polite); keeping track of what the hearer knows and what we want her to know; ensuring that the entire text forms some sort of unit; choosing the appropriate lexical items (words, phrases); and combining the lexical items according to the rules of syntax.

Choosing what knowledge to convey is an important first step in the process of generation. Some generators assume that some other component will make this choice for them. For example, a system that proves theorems might give a generator the problem of producing text about a theorem. Some generation systems take it upon themselves to select the information to be conveyed, for example, systems that answer questions about a database. Most generators, in the process of producing text, need to make decisions about including particular pieces of information; this issue of what to include remains an open and difficult issue.

Often the goal of generating a text is simply to convey information. Sometimes, however, other goals are involved. For example, if a system needs to acquire information from a person, then the generator needs to know how to ask a question. Asking a question involves both syntactic knowledge about how a question differs from a declarative sentence and more general knowledge about the background against which a question may be framed. Some goals a generation system may have do not have such clear-cut textual realizations. For example, being polite might involve using the word "please," but it also might involve selecting more verbose constructions or using verbs in the subjunctive mood. Thus, "perhaps you would like to come tomorrow" is more polite than "come tomorrow."

Keeping track of what the hearer knows and using that information to guide the construction of text is often called user, or hearer, modeling. For example, if the hearer can be assumed to be familiar with the subject matter of the text, shorter noun phrases, or even pronouns, may be used, whereas if the topic is new, more information must be included when reference to the topic is included in the text. User modeling is particularly important in generation systems involved in answering questions or providing explanations.

Ensuring that the text forms a unit, or "hangs together," is sometimes referred to as *coherence*. Coherence can be defined as the property of unity of a text that arises from associations between the underlying concepts and the organization of these concepts and associations. A model of coherence may be used to ensure that the text includes material in an orderly fashion. This might, for example, involve making sure an object is named before it is described, or making sure that if an object's components are mentioned, they are mentioned in one place, not scattered about the text.

The term "coherence" is used technically in the computational linguistics community to refer to a criterion with which to evaluate text. However, there is no coherent definition of this term in the literature. For instance, Hobbs, in a paper entitled "On the Coherence and Structure of Discourse" (1985), entitles the first section "Discourse is Coherent." The first sentence of this section is "Let us begin with a fact: discourse has structure." In lieu of a definition of coherence, the reader is led to believe that "coherence" and "structure" are synonymous. Few authors do better at explaining what they mean by this important word.

Focus is another important term with a technical meaning within computational linguistics. Focus is a constraint on what can be said next. What can be said next has two parts: those items that have not been mentioned yet are appropriate candidates for mentioning now; and which of these appropriate candidates have already been said. In addition, focusing, by distinguishing items which the text is currently about from those which it is not, predicts when the use of anaphora is proscribed or required. Generally speaking, an *anaphor* is a word or phrase which is understood only in terms of its *coreferent*, which is another word or phrase in the text (the coreferent usually precedes the anaphor). For example, in the following sentence, "the cat" is the coreferent of the

anaphor "she."

"If the cat weren't such a wimp, she would be jumping about the rafters right now."

Lexical choice is choosing the right words or phrases to convey concepts and relations. Lexical choice is interwoven with many of the previous issues: goals such as degree of politeness and decisions about how much information to include in a noun phrase have an impact on the component of a generator that chooses words and phrases. Even in the general case, choices need to be made between closely related words such as "boy" and "man."

The final step in the process of generation is actually producing the text; this requires a knowledge of syntax, which is knowledge of how words may be put together to form larger structures, such as noun phrases and prepositional phrases. Typically, a syntactic component builds sentences out of words and phrases, one sentence at a time. Syntax is a hard problem: while grammars exist that account for a substantial fraction of the syntax of English, no grammar is complete, and thus it may be difficult or impossible to find the syntactic structure appropriate for generating the text that the generator would like to produce.

In very early natural language work, the complexity of language production was largely finessed by using "canned text," though there was some work in constructing simple sentences (for example, Simmons & Slocum 1972). An exception could be found in Schank's group's work in conceptual dependency (Goldman 1975, Cullingford 1986, Meehan 1977), a holistic approach to natural language. Many of the early systems, especially those producing "canned text" in a single domain, were domain dependent. Domain independence is a desirable property for natural language generation systems for practical reasons of portability between domains and for cognitive plausibility, since people certainly use language in a variety of domains.

Most work in natural language generation has been concentrated in the last decade. By 1981, a report on the state of the art in generation (Mann *et al.* 1981) found the field still in its infancy, with only a dozen or so active researchers. Ten years later, the field is somewhat bigger; approximately 80 people attended the Fifth International Workshop on Natural Language Generation in 1990. However, the issues remain the same.

Certainly, no one yet has a handle on how to accomplish everything at once. Many researchers have developed elaborate models of modules necessary for various subtasks. For instance, Levelt proposes specialists for message generation, self-monitoring, grammatical encoding, phonological encoding, and articulation, using as knowledge sources a lexicon of words and structures, a discourse model, situation knowledge, an encyclopedia, etc., with a complex control structure involving extensive feedback (Levelt 1989, p. 9). Kempen and his coworkers envision a similar model with modules working in parallel while information cascades from conceptual to lexico-syntactic to morpho-phonological to articulatory (De Smedt & Kempen 1987). Nirenburg postulates an arbitrary number of specialists responsible for everything from modifier selection to prosody; modules may proliferate because they all communicate via a blackboard architecture (Nirenburg *et al.* 1988). A few years ago, some researchers (McDonald *et al.* 1987) surveyed the extant generation systems and tried to compare them in terms of a standard stratification into three stages: gathering information, turning it into an intermediate representation, and "reading the message out," which is the process of turning the intermediate representation into text. The authors' original aim had been to compare the efficiency of generation systems; their conclusion was rather that the systems were so diverse as to be essentially incomparable.

1.2 Parts of the Process of Generation

In 1977, Thompson presented a paper (Thompson 1977) in which he speculated on a computational model of natural language production, or generation. He suggested that approaches to the problem of generation be composed of two parts: a *strategic* level and a *tactical* level. He described the strategic level as more abstract and closer to the cognitive component(s) of a system, and the tactical level as the "locus...for the nitty-gritty details of syntax" (p. 656). Thompson expressed the hope that the tactical component could expediently be taken care of and that these nitty-gritty details could be encapsulated in their own module, leaving researchers to tackle the more interesting problems of generation, which are clustered in the strategic component. These problems include interfacing between language and other cognitive processes, using language to serve goals (such as asking a question or making a promise), ordering the information to be conveyed, and choosing the appropriate words or lexical items to be produced.

Thompson was describing a theory of language production rather than a computational model, and he allowed that his bifurcation of the process was more for explanatory convenience than because the process of generation falls neatly into two distinct parts. However, the division of the process into a strategic phase followed by a tactical phase has become a cornerstone of the field of natural language generation's conception of itself. (This sort of division of language into syntax and semantics/pragmatics/lexical choice is not a new idea, of course. Much of linguistics has maintained a similar distinction, as well as relegating morphology and phonology to still other distinct components.) Terminology varies from author to author: text planning and realization (Meteer 1990); text planning/sentence planning and linguistic realization (Rambow 1990); text planning/mapping and syntactic realization (Defrise & Nirenberg 1990). Since *text planning* is commonly used to refer to the strategic component and a variation of *surface structure realization* to the tactical component, I will try to stick with this usage.

1.2.1 Surface Structure Realization

Research that focuses on the tactical phase of generation—the syntactic nitty-gritty—tends to take the sentence as the unit of concern. This line of research has resulted in a few attempts at "off-the-shelf" generators (e.g., Mumble-86 (Meteer *et al.* 1987) and Penman (The Penman Project 1989)). These surface structure components expect input that is fully specified except for syntactic detail, which the surface structure component provides. The syntax may be more or less fully specified by the input. What is not specified is left up to the surface structure component, which will attempt to make the best syntactic fit for the input. The surface structure component's best fit can be overridden by the text planner: for example, the text planner might specify *there-insertion*, so that the surface structure component will produce

```
There is a large window in the kitchen.
```

instead of the default

```
A large window is in the kitchen.
```

Such off-the-shelf systems may be functional in simple, limited applications, but the restrictions they impose on their input are so limiting and often idiosyncratic that most natural language generation projects write their own tactical components to ensure compatibility with their other research goals, and to be able to focus their efforts on the strategic issues of interest rather than on the interface to another program.

Especially focused work in generation can be found in research on particular grammatical

formalisms. Most such research has emphasized the role of grammatical formalisms in parsing language, though some formalisms, notably Tree Adjoining Grammars (Joshi 1987), have been investigated in the context of generation as well. A recent trend in computational linguistics has been "bidirectional" grammars (Russell *et al.* 1990, Strzalkowski & Peng 1990). Such grammars have the property that they can be "run" in one direction to parse text into some internal representation, and run in the opposite direction to generate from the internal representation into text.

1.2.2 Text Planning and Other Strategic Work

Research in text planning, which is concerned with the "everything else" of the strategic component of generation, looks at units of text that may be smaller or larger than a sentence. The focus is less on the syntactic form of the linguistic output and more on issues such as deciding what is an appropriate answer to a question, choosing between two words for a concept, deciding what order should be imposed on the content to be conveyed, and determining what can be inferred from what has already been said. Extreme instances of such generation systems do not produce any text at all. Some explanation systems, for example that of Cawsey (1990), produce intermediate representations that could be reasonably turned into sentences were appropriate machinery available. Other research efforts, such as those concerned with lexical selection, (Reiter 1990, Pustejovsky & Nirenburg 87), are not embedded in a text producing system at all. Lexical selection is the process of choosing the word or phrase that most accurately conveys the concept to be expressed, such as the choice between "chair" and "easy chair." Which selection is more appropriate may depend on the context (under some definition of context); additional information may be conveyed through lexical choice, such as the difference in attitude between referring to a pet cat as a "monster" and as a "magnificent beast."

1.3 Typical Generation Systems

In this section, I outline a "typical" generation system. This is not a description of any specific system, but a characterization of what could be called the state of the art in the field of natural language generation. This typical system has the following characteristics: it has separate strategic and tactical components; it uses a hierarchical plan or a schema to structure text; it uses rhetorical relations to insure a domain independent method of generating text; and it employs special purpose focusing mechanism to handle what to say next and when to use anaphora.

While I have mentioned some research projects that fall squarely on one side or the other of Thompson's strategic/tactical line, most work in natural language generation actually concerns issues on both sides of this line. Further, this line, with the weight of tradition behind it, is compelling; generation systems that produce text output in some sort of context that requires more than simply producing sentences from a well-specified representation usually cannot avoid making some sort of commitment on where the strategic/tactical line is drawn. That is to say, most systems have a component that takes care of the nitty-gritty syntax, and most systems consider the interesting issues to, by definition, happen outside of that component. While no one actually tries to claim that the problem of surface structure realization has been solved, today most of the emphasis is the field is on text planning.

For most text planning systems, the *text plan* is a central data structure (e.g., Appelt 1985, Hovy 1990, Moore & Swartout 1989, Cawsey 1990, Wahlster *et al.* 1991). These systems build a hierarchical plan, using technology borrowed from artificial intelligence (Sacerdoti 1977). McKeown's

(1985) and Paris's (1988) systems use *schemas* (implemented as augmented transition networks) to serve the same purpose as text plans, and the implementations are functionally similar. For researchers in the text plan/schema tradition, this tree shaped structure suffices as an account of coherence and text structure. (Remember that "coherence" is typically defined simply as structure.)

Typically, domain independent approaches to text generation rely on *rhetorical relations.* Rhetoric is the art of persuading an audience of any proposition. This attractive property of rhetoric—that it can be used in any domain—has given rise to the notion of "rhetorical relations." as reflected in the name of Rhetorical Structure Theory (RST) (Mann & Thompson 1987), the most popular theory of this sort. RST comprises 23 relations between clauses. Some of these relations, such as **justify** and **motivation** are *presentational*, that is, meant to "increase some inclination" of the audience. The other relations, like **sequence**, **elaboration**, and **non-volitional cause**, are *subject matter* relations which, in effect, convey information about relations in the domain.

However some domain dependent knowledge clearly is required to appropriately incorporate domain knowledge into the use of rhetorical relations: domain relations must be categorized in terms of the roles they can play in rhetorical structure. For example, **non-volitional cause** is not the same relation in the domain of exploding bombs as it is in the domain of international affairs, and a generation system needs to know *how* to recognize and exploit **non-volitional cause** in each domain.

Systems that explicitly use RST include (Hovy 1990, Moore & Swartout 1989, Cawsey 1990, Wahlster *et al.* 1991). McKeown and Paris's schemas use a similar set of rhetorical relations, and many other systems have versions of their own.

Usually the structural components of the text plan are rhetorical relations. When the text plan tree is composed of rhetorical relations, global structure, coherence, and domain independence are combined in one mechanism that results in a single structure that determines the text.

Some systems have additional mechanisms for focus. Focusing is represented as a constraint on what can be said next, and how it can be said (e.g., the surface order of noun phrases and whether a noun phrase can be pronominalized); the focus is the item or items which are currently what the text is about. "What can be said next" has two parts: which items that have not been mentioned yet are appropriate candidates for mentioning now; and which of those appropriate candidates have already been mentioned. To address the first part of the issue of focus, McKeown's system (1985) kept a relevant knowledge pool to draw from; to address the second part, it used a stack mechanism based on Sidner's focusing algorithm (1979); the focusing mechanism keeps one or more objects in focus and applies focus constraints which order propositions, by considering which proposition should follow the current one depending on the objects in focus that they have in common. More recently, focus trees have been used to provide a more complete record of what has been mentioned and in what context (Hovy & McCoy 1989). In this case a tree is built during the construction of the text to record not only what has been said and what can be said next, but also the relations among the things said. The tree provides a more complete record than the stack, since essentially a tree is a "stack with memory."

1.4 The Text Generation Problem That Salix Addresses

Salix exploits domain structure to generate texts that are organized by the structure of their subject matter. In this section I present in more detail the data and domains which are used as a resource for the types of text that Salix currently generates, and discuss characteristics of domain structure. In the next section, I will give an overview of Salix's architecture and present an example of the

texts that Salix generates.

In its current implementation Salix organizes and generates texts that describe house layouts, family relations, and physical processes. The first two domains were used in developing the system, the third was chosen to demonstrate the system's generality.

Throughout this research, a data set of house and family descriptions serves as both a model of texts in a domain and a test of what the system produces. These descriptions, nine of houses (or apartments) and 15 of a family, were tape-recorded and transcribed and were given in response to the following questions:

1. "Could you please describe for me the layout of your house (apartment)."
2. "Can you tell me how everyone who comes to Thanksgiving is related to each other?"

Descriptions of houses and families are good examples of texts structured by subject matter. That they are good examples of course does not imply that they are the only texts that exploit domain structure. In fact, they are good examples primarily from the researchers' point of view: houses and families are familiar things, as are the ways we talk about them in our culture. We can also draw pictures of houses and families: the ways of drawing these pictures represent some of the kinds of information that texts about the same subject matter do (as well as representing some very different information). Houses have the additional property of being concrete: we can look at or walk through a house to learn more about it. All these features make houses and families good subject matter for examining texts which are structured by exploiting domain structure, just as letters to the editor (Mann & Thompson 1987) are a good site for examining rhetorical structure of text. Once a method has been demonstrated to work in relatively clear cases, there is a basis for extending it and showing its generality, and certainly the effect of the structure of subject matter on the structure of text is a general phenomenon.

Houses and families are structured; their components are interconnected. A family is completely related by parent/child and spouse relations, and has other relationships, such as sibling; the *basic relation* between family members is that they are related by blood or marriage. Parts and contents of houses are often related by shared functionality or attributes. There is again a basic relation: in a domain like a house that has spatial properties it is physical proximity—everything is next to something else. In addition to its parts being strongly interconnected, a house is naturally circumscribed; as Linde (1974) found, a house description can stop whenever all the rooms have been mentioned. The boundaries of a family may be more ill-defined; this consideration prompted creating the more specific task of describing the part of the family that came to Thanksgiving.

1.5 The Strategy-based Architecture

In Salix, the process of organizing and surface structure realization is unified; there is no distinction between the strategic and the tactical. Organization of the text and surface structure realization are both incremental, and increments of generated text, ranging in size from a word to a clause, result from increments in organization. Most of Salix's strategies that build text structure are directly responsible for producing text as well.

Salix builds descriptions by means of description *strategies*, *metastrategies* that combine or choose from among competing strategies, and *null strategies* that decide what to do when there is nothing to say. Salix generates text from knowledge bases that are in the form of networks

that encode important aspects of a domain. The nodes in the knowledge base are objects (e.g., refrigerator, bathroom), properties (e.g., large), and relations (e.g., next-to, has-property) in the domain. The system keeps track of one node in the knowledge base at any one time: this *current node* is the node which the next increment of the description will be about. The system also keeps track of a dynamic *context*, which includes a variety of information such as what has just been described, whether an object has been explicitly mentioned, and what strategy has just been used. The context can also include domain specific information, such as, for house descriptions, what spatial direction the description has been progressing in.

Salix employs a simple dispatch process to decide how to continue the description from the current node. At any increment, a number of strategies that are applicable match. The result of a match falls into one of three cases. If there is exactly one candidate, then the strategy is applied. If there are no candidates, which means that no strategy is applicable, then this impasse is noted and dispatched upon. If there are multiple matches, then the dispatchee and the list of candidates are combined to form a new dispatchee, which is then dispatched on.

There are two types of strategies that are triggered when the dispatchee is a node: the *saying* strategies and the *finding* strategies; these exploit the elaborating and basic relations in the domain respectively. The saying strategies produce text about the current node and the finding strategies find a next node to talk about. As will be discussed in chapter 4, the distinction between saying and finding strategies allows the system to handle phenomena usually associated with a separate focusing mechanism. These phenomena include pronominalization and cues that signal the introduction of new elements into the text.

Metastrategies are triggered when there have been multiple matches. Metastrategies are responsible for resolving conflicts among competing applicable strategies. Two approaches to conflict resolution have been implemented in Salix. The *choose* metastrategy chooses one strategy from the set of candidates by finding the one that is most appropriate as determined by a partial order of preference on all the strategies that may apply in this domain. *Combining* metastrategies, on the other hand, are triggered when some subset of the available strategies may be felicitously combined.

Null strategies are triggered when no strategies have matched. One of three things can happen: if a strategy has inhibited matches, control is returned to that strategy. If there is a node that should be included in the text but has not been marked as having been said it is dispatched on. If there are no such nodes remaining, there is nothing left to include, and the process stops.

Salix's architecture is similar to that of many rule based systems, and in fact the strategies can be considered rules and the set of multiple matches a conflict set. However, deliberation is much more flexible than the usual conflict resolution mechanism of rule based systems, which simply selects one rule from the conflict set according to a hard-wired criterion. The conflict resolution scheme used in Salix is similar to subgoaling in Soar (Laird, Newell, & Rosenbloom 1987). Salix's architecture is based on that of unpublished systems designed by Phil Agre, John Batali, and David Chapman. For a diagram representing Salix's mechanism, see figure 1.3.

Houses and families may or may not have discernible global structures: a house may have a hierarchical structure; family relations may be capturable in an almost tree-like graph containing just the two relations parent/child and spouse. However, my analysis of the data suggests that the production of the descriptions does not depend on such global structures. Instead, the descriptions can be modeled as exploiting the rich local structure of the subject matter. The structure of the texts in both domains can be accounted for by a small set (approximately 10) of *domain independent* strategies, which are responsible for increments of text which vary in size from word to clause.

The strategies exploit the relations of a domain in a data driven manner; additional domain

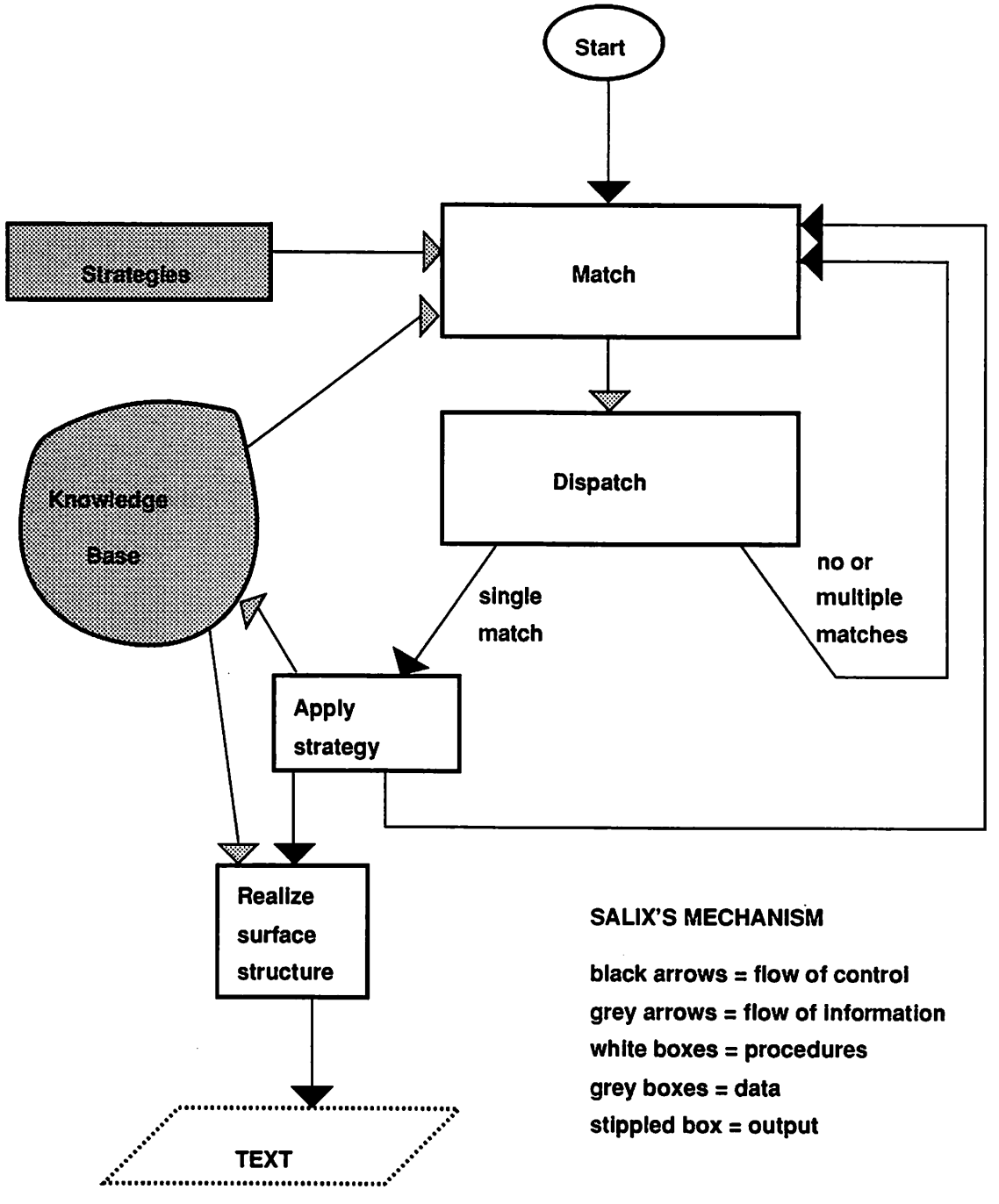

Figure 1.3: A diagram of Salix's mechanism.

specific ways of expressing domain relations are encoded in some of the metastrategies which can combine strategies. The process of incrementally choosing *what to say next* from the locally available choices accounts for the organization of the text. This choice process is based on a context of what has been said before and what is available to say now. There is thus no hierarchical organization imposed on the text in the process of its construction. That a hierarchical structure, like the plan produced by a hierarchical planner, may be found in a text after it is produced is independent of the process used to generate it. The structure of the text is derived from the domain structure of the subject matter. This is a consequence of the local organization of the text which is sensitive at each increment to the structure of the subject matter in deciding what to say next.

The approach implemented in Salix provides a domain independent *data driven* method for exploiting domain structure to structure text. The work in this thesis approaches domain dependence in a principled manner; while no generator can be completely domain independent, the domain dependence in Salix is minimized. Any implemented system, unless it does exactly one thing, will vary in its behavior from run to run (for example, produce different texts). This variation needs to be controlled in some way by the user of the system. One way to control this variation is within the control architecture of the system itself, however, this approach may make the system harder for a user to modify, and thus less domain independent. The data driven approach is an alternative way to control program variation. Additional information from the user is provided in a minimal (hence easier to use) format, which (usually) does not involve actual coded procedures. Salix incorporates three data driven formats, which range from completely declarative data to procedural attachment hooks. All of Salix's domain dependence is in one of these three formats, or in the combining metastrategies.

The first format of data driven information is simple *lookup tables*. Salix has such tables for each domain that list the preference order on available strategies at each point in the description; these tables are used by the metastrategy **choose**. The second format of data driven information is *annotations* on knowledge base elements. To be principled and easy to use, such annotations must be drawn from a small restricted set. An example in Salix, the distinction between basic and elaborating relations, is discussed below. The third format is *procedural attachment*. In this case, a *hook* is provided at a single locus in the code, where if necessary a small, domain dependent procedure may be called. Salix has one such hook, the *trajectory hook*, that is called during house descriptions to update how the text reflects the spatial organization of the house. This information is used to select appropriate deictic phrases, such as "on the left."

For Salix to exploit domain structure relations, these relations need to be annotated as to which of a small number of strategies may be used to express them. This annotation categorizes domain relations as either *basic*—exploitable for domain structure—or *elaborating*—contributing additional descriptive information. (A relation may be both: basic information can also be used for elaboration.)

To illustrate the difference between basic and elaborating relations, let us consider examples from both the house and the family domains. In the house domain the basic relation is physical proximity and both Salix and human speakers usually exploit this relation to find the next thing to say, for example the next room or the next piece of furnishing. Elaborating relations include having a property, being composed of parts, and having a particular orientation. So, for example, the kitchen window can be found by the basic relation of physical proximity and included in the text. Then the description of the kitchen window may be elaborated on by adding text about other relations it is involved in, such as being a picture window, having two flanking windows, and facing the backyard. Occasionally a relation that is usually an elaborating relation may be used to find a next thing to say. For instance, if two objects have the same property, the second object may

become the next thing to talk about.

In the family domain, Salix's texts differ slightly from people's: Salix builds texts that only involve the basic relation of relatedness. In my data, people elaborated their descriptions in a number of ways, including idiosyncratic remarks about their relatives such as how much they earned and what poor judgement they showed in naming their children. However, I did not model these remarks, so Salix's family texts stick to the basics of who is related to whom how.

Incrementally choosing what to say next accounts for most *focusing* constraints. The phenomenon of focus naturally falls out of the process of local organization. Focus is a constraint on what can be said next; what can be said next is exactly what is addressed by local organization in Salix. In addition, focusing, by distinguishing items which the text is currently about from those which it is not, predicts when the use of anaphora is proscribed or required. Most of these constraints on anaphora are local and are incorporated into the local organization model without resorting to special purpose focusing mechanisms.

In order to demonstrate the system on domain structured text in a domain for which it was not initially developed, Salix's approach has been extended to descriptions of physical processes, such as those that occur when water is boiled in a sealed container (Forbus 1985).

1.6 Example of Salix's Execution

In this section, I will give a synopsis of Salix's construction of a text in the house domain. The text, which appeared earlier in this chapter, is repeated in figure 1.4, with a sketch of the house in figure 1.5. The house sketch is annotated to indicate the approximate spatial locus of concern to some parts of the text. This is a complete text given by Salix when asked to describe the house, starting at the **side door**. When Salix uses a finding strategy to find a *next* thing to mention, this next thing is next to the current thing in terms of the basic relation of physical proximity.

Each line of text in the figure is an *increment*—a segment of text generated by a single source. Most of these increments result from the selection of a single strategy, based on the current context, by the strategy interpreter described in the previous section. Some increments, such as those in (16), result from the application of a combining metastrategy that collects and orders a set of matching strategies. Still other increments result from the control structure of the process, such as the discourse cues at (1), (17), and (20).

The text begins with an introductory cue (1), and an introduction of the first object, which is described with a special saying strategy because it is first (2). The entrance hallway is found next, using the default finding strategy and following a next-to link, and is described by the default saying strategy (3). The kitchen is an available next thing to the entrance hallway; the kitchen has a number of features. These features include a window that has several features in its own right and is next to many other things in the kitchen; therefore it is particularly *salient* and is selected for mention first. The kitchen is mentioned in a prepositional phrase to provide a context for the mentioning of the salient object (4), and then then salient object itself is mentioned (5). A metastrategy **say-multiple-sweep** organizes part of the text (4-12) around the salient object, the window. Several properties of the window are described, using a saying strategy specific to properties (6-9), and then the window is used to anchor the ordering in which related objects are mentioned. In a *sweep*, objects that lie along the same trajectory (e.g., a spatial trajectory toward the north) are mentioned in the order in which they lie on that trajectory. In a multiple sweep, sweeps are made along several trajectories, all starting at the same point. In this case, the kitchen window anchors three sweeps of objects in the room (10, 11, 12). Each sweep begins with an

```
(1)   OK
(2)   we can start at the side door
(3)   and then there's a entrance hallway
      then
(4)   in the kitchen
(5)   there is a window
(6)   which is large
(7)   and is a picture window
(8)   it has two flanking windows
(9)   and faces the backyard
(10a) and if we're facing that window
(10b) on the right is the sliding glass door
      and a window
      which is small
(11)  if we're facing the backyard
      on the left is the stove
      and a refrigerator
(12)  and if we're facing that window
      underneath is the sink
      then a dishwasher
(13)  then there's a living room
(14)  it has
      a window
      which is large
      in it
(15)  and then there's a short hallway
(16)  and
      Ann's bedroom on the left
      Claire's bedroom on the right
      and
      a bathroom in the middle
(17)  oh yeah
(18)  and then there's Penni's bedroom
(19)  and a long hallway
(20)  that's it
```

Figure 1.4: House description generated by Salix.

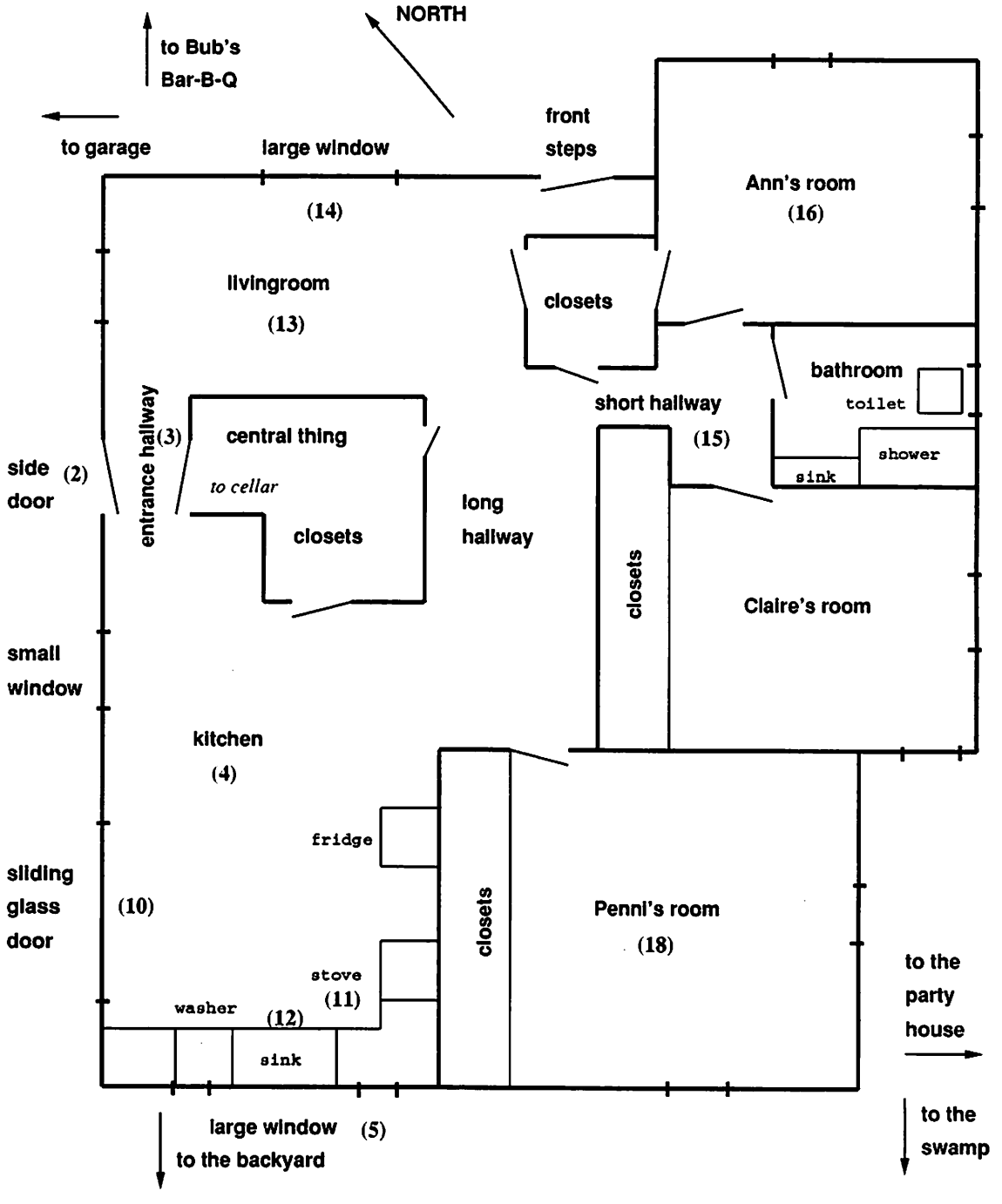

Figure 1.5: Sketch of house.

orientation to the anchoring salient object (10a) and uses a deictic expression (e.g., "on the right") to indicate the relative direction of the sweep (10b). These relative directions must be computed dynamically, since they depend on the objects involved, orientation, and which trajectory between the objects is described. (12) completes the multiple sweep and the metastrategy relinquishes control. The living room is next to the kitchen, so it is mentioned next (13). It contains a window, which is mentioned as the living room's contents (14). Containment is an elaborating relation, and this increment is produced by the strategy for saying things that are related to the current object under an elaborating relation. Next, there is a short hallway (15). From here, there is another opportunity to use deixis in describing the rooms that are connected to the other end of the hallway (16); this set of increments is produced by the metastrategy **say-left-right-and-center**, which is triggered when there are three things available to say, one of which lies along the current spatial trajectory and two of which are orthogonal to that trajectory. Now, there is no interesting available next thing to say, so perhaps the description can stop. Salix checks the completeness criterion, which for the house domain states that all the major rooms (those that are not hallways or closets) must be mentioned. One of these rooms has in fact not been mentioned, and Salix signals its oversight (17) and mentions the forgotten room (18). The long hallway happens to be next to Penni's bedroom, and it gets mentioned (19). It is odd that, as it turns out, the last room mentioned is a hallway, but since Salix does not look ahead, it cannot know that this will be the last room. Salix checks again that it has not forgotten anything important, and indicates that its description is done (20).

1.7 Contributions of This Thesis

This thesis presents Salix, an implemented system that generates extended texts. The system and the theory behind it have a number of distinguishing characteristics, which represent contributions to the field of natural language generation.

1. A single, unified process decides what to say next and chooses the surface forms of the next increment of text. In contrast to most generation systems, there is no distinction between "levels" of the process of constructing a text; there is no distinction, for example between deciding what to say and deciding how to say it.

2. The text is locally organized by the system incrementally choosing what to say next. This choice is based on a context of what has been said before and what is available to say now. Whereas most systems use top-down hierarchical planners to organize text, in Salix there is no hierarchical organization imposed on the text in the process of its construction. That a hierarchical structure may be found in a text after it is produced is independent of the process used to generate it.

3. The structure of the text, in virtue of which it is coherent, is the domain structure of the subject matter. This is a consequence of the local organization of the text which is sensitive at each increment to the structure of the subject matter in deciding what to say next. Most text generation systems rely on rhetorical structure to achieve coherence. In Salix, no additional rhetorical structure needs to be built to make the text coherent.

4. The phenomenon of focus naturally falls out of the process of local organization. Focus is a constraint on what can be said next; what can be said next is exactly what is addressed by local organization in Salix. In addition, focusing, by distinguishing items which the text is currently about from those which it is not, predicts when the use of anaphora is proscribed

or required. Most of these constraints on anaphora are local and are incorporated into the local organization model without resorting to the special purpose focusing mechanisms found in other systems.

5. Salix is a domain independent architecture for generating texts whose subject matter is about some domain. Typically, domain independent approaches rely on rhetorical relations to ensure domain independence: rhetorical relations are selected to provide structure to the text, and domain knowledge is fitted into them. However some domain dependent knowledge clearly is required to appropriately incorporate domain knowledge into the rhetorical structure: domain relations must be categorized in terms of the roles they can play in rhetorical structure. The approach implemented in Salix provides a domain independent data driven method for exploiting domain structure to structure text. As with rhetorical structuring schemes, domain relations need to be identified to the system. For Salix to exploit these domain structure relations, these relations need to be categorized as either basic—exploitable for domain structure—or elaborating.

Evaluation of a natural language generation system is both easy and difficult. It is easy because good text is something that "you know it when you see it." So it is easy to tell if a system is producing reasonable text. By the same token, one can apply any sort of criteria to machine generated text that one can apply to human generated text, though such criteria are often subjective and value laden. I will show by demonstration throughout this thesis that Salix's texts share characteristics with the texts that I have gathered from speakers about the same domains. I will also show that Salix can generate text in a domain for which it was not originally developed.

Evaluation is hard because there is no "gold standard," no suite of tests or set of benchmarks for generation systems or indeed natural language systems in general. In recent years, several workshops have been held on the topic of evaluation (see, e.g., Palmer & Finin 1990) but the major conclusion to have been reached is that there is no basis for comparison between systems, especially in generation. This is in part because the problem is still being defined; this thesis is part of that process of definition. Another reason is that different domains inspire different approaches; if researchers were restricted to generating a small fixed set of texts, their ability and incentive to explore broader issues in generation would be curtailed.

I cannot say that my system is faster or smaller or wider-ranging than other systems because statistics on such parameters are not available, and even if they were, comparison would be difficult, given differences in architectures and domains. I can point out differences between my system and others, and suggest that my system is easier to use or more appropriate, but I cannot make any claims along these lines because there is nothing that can be proven. What I can do is describe the capabilities of Salix as accurately and completely as possible, and explicate what Salix can and cannot do. These issues are addressed throughout the thesis and summarized in chapter 8.

1.8 Overview of the Thesis

This section describes how the rest of the thesis is laid out.

Chapter 2 describes in detail the data on which much of this research is based. I discuss why I selected the domains I did and how I collected and transcribed the texts. I present an analysis of the organization of the texts in both the family and house domains and a more detailed surface structure analysis of the house texts.

Chapter 3 describes in detail the data driven, domain independent, strategy based architecture of Salix. In this chapter, particular attention is given to the domain independent aspects of the system, including the strategy interpreter and the strategies, including the saying strategies, the finding strategies, the metastrategies, and the null strategies. Also discussed are the elements of the context used in the process of local organization, and an inference mechanism that forms the basis of a hearer model. The chapter includes a review of AI problem solving architectures which are related to the architecture implemented in this thesis, and concludes with a review of text planning systems.

Chapter 4 discusses the construction of family text, which does not rely on any metastrategies besides the choose metastrategy. This chapter explores in particular the notion of focus and how focus is handled in Salix.

Chapter 5 described the construction of text in the house domain. This chapter includes the model for representing deixis that I developed jointly with Alison Huettner and have implemented in Salix. The chapter then details the construction of house texts without the use of metastrategies besides the choose metastrategy. Two spatial metastrategies are then introduced and a example of text construction that employs them is presented.

Chapter 6 demonstrates that Salix's technology can be easily applied to generating text in a new domain, that of the processes involved when water is boiled in a sealed container. In this case, Salix's texts are not modeled on texts elicited from people but on a textbook description.

Chapter 7 presents an overview of the state of the art in surface structure realization. A characteristic of this approach is that it is sentence-based. I then describe the surface structure realization in Salix, in which the sentence does not play a distinguished role.

Chapter 8 is concerned with evaluation and future directions. Salix's text generation has been evaluated by selectively disabling components of the system and asking people to judge the resulting texts. The contributions of these components is evaluated in light of these judgements. The issue of where local organization is problematic is considered. Next, this chapter includes a consideration of Salix's requirements on structure in a knowledge base for generating text. In the remainder of this chapter, I explore future directions for the research, focusing on phenomena in the data that the current model does not yet account for.

Chapter 9 recapitulates the themes of this thesis. The approach to text generation presented here and embodied in Salix is evaluated on the dimensions of a unified process of generation, local organization, coherence, focus, and domain independence and compared to other approaches in the field. This evaluation highlights Salix's contributions as a domain independent system for exploiting domain structure for generating coherent text.

This thesis includes four appendices. Appendix A contains the house and apartment description data elicited from speakers. Appendix B contains the orders of mention of family members in speakers' descriptions. Appendix C contains the portion of the knowledge base representing the house and the family in the implemented system. Appendix D contains sample traces of Salix's execution in constructing texts.

Chapter 2

Data

In order to model natural language use, I have studied actual language use. In order, more particularly, to study how texts structured by domain structure are constructed, I have studied actual language use in domains with strong domain structure, as discussed in section 1.4.

In this chapter, I discuss the data sets of transcribed spoken language that have informed this work. I describe the methodology for collecting and transcribing the data and then examine in greater detail the empirical findings that have influenced the model of natural language generation presented in this thesis. Many other researchers have studied living space description; I review some of this work, particularly that of Linde (1974).

There are three corpora that I have been working with.

1. The *House Corpus* is a set of seven recorded descriptions of a one-story house.
2. The *Apartment Corpus* is a set of two recorded descriptions of an apartment in a house.
3. The *Family Corpus* is a set of fifteen recorded descriptions of an extended family.

2.1 The House and Apartment Corpora

2.1.1 Gathering the Data

The first set of data was gathered for a project of generating descriptions of living space, which became my master's thesis (Sibun 1987). The corpus was meant to be similar to the apartment descriptions collected by Linde (1974). As part of a longer interview, which usually took place in the apartment in question, Linde asked New York apartment dwellers

"Could you describe for me how this apartment is laid out?"

Her thesis (discussed in more detail in section 2.1.3) is a detailed examination of the regularities in the descriptions.

The area around the University of Massachusetts at Amherst does not have New York City style apartment buildings, so I chose as the site to be described a house in Sunderland, MA, in which I was living at the time. A representative excerpt of a house description can be seen in figure 2.1; full texts may be found in appendix A.

then in the kitchen
....there's a large window which faces the backyard
with two smaller windows directly flanking it
and....if we're facing....towards the backyard now
on the righthand side is....a sliding glass door
and....a few feet from that is a smaller window
towards the living room
then....on the wall which....would
....partition the kitchen from the living room
there is a closet
....and behind that closet would be the stairwell that goes down to the basement
so there's like a block between the kitchen and the living room
um....in the kitchen
if we're again facing the back yard
on the lefthand side is the stove
....then....a refrigerator
and beneath that large window....is the sink
and next to that
on the righthand side
is the dishwasher
boy if somebody tried to reconstruct it from this they would never get it
ok

Figure 2.1: Representative extract of a house description.

The conversational setting, what a speaker knows about the house, what she knows about language, what her mental models of houses might be, are all uncontrolled variables in collecting house descriptions. The only controllable variables, if one is eliciting speech in as natural a way as possible, involve who gives the interview, where it is given, and what questions are asked. Where the interview was given was controlled for, and what *initial* questions were asked was regulated, but, as I have pointed out above, it is difficult to keep this constant across speakers.

It is possible that a speaker's description will differ depending on how well he knows the interviewer, and how well he thinks the interviewer knows what the speaker is describing. When I collected the house descriptions, all the speakers knew I was very familiar with the house, and they all knew me moderately to very well. All certainly knew of my familiarity with the house being described. The criteria used for selecting people for interviews were that I thought they had been in most parts of the house and that they had spent at least one night there. Spending a night in a house makes it more likely that one will visit more of the rooms than one might on a casual visit. The speakers must have spent this time in the house within the previous year.

Six of the interviews were collected outside of the house: three in the backyard; one in a car; and two over the phone. One interview was conducted in the speaker's room (which did seem to have the effect of making her description start at that point). The interviews were taped on a portable cassette tape recorder. (In the case of the phone interviews, the speakers taped themselves.) All interviews started with the question

"Could you please describe for me the layout of this house?"

Speakers were sometimes prompted for more detail. For example,

Penni
could you please describe the layout of our house for me
Ann
the layout of our house
in terms of directions?
Penni
in terms of like a floor plan
Ann
ok

Claire
the bathroom is fairly uh....not square
I don't know, does this—the bathroom stick out?
Penni
no!
Claire
it doesn't?
Penni
I don't think you're supposed—
Claire
it doesn't?
Penni
—to ask me though!
Claire
well, I don't—ok—anyway

Figure 2.2: Examples of negotiating the production of a house description.

"Can you give more detail of what the rooms are like?"

These prompts almost certainly had an impact on what people said. In fact, the question of what to say was often negotiated, both in determining what I was asking for and discussing the appropriateness of some aspect of the speaker's descriptions. Examples of both sorts of negotiation are found in figure 2.2. While the descriptions were actually embedded in dialogue, their conversational aspect is unfortunately beyond the scope of this study.

Several years after collecting the house corpus, I gathered some new descriptions against which to test the hypotheses of text structure that I had formulated. By this time, I was living in another house in Sunderland, MA, which had been divided into four apartments. I asked the married couple in the apartment across the hall to help me with my thesis and, in order to control for interviewer knowledge, asked a colleague of mine to conduct the interviews. The speakers knew the interviewer slightly, but to their knowledge he had never been inside their apartment. The interviews were conducted in the hall outside the apartment, again recorded on a portable cassette recorder.

There are no striking differences between the house corpus and the apartment corpus that can be attributed to the speakers having a different relationship with the interviewer. The descriptive text exhibits a similar set of features, and the same interactive processes of negotiation and explanation occur.

2.1.2 Transcribing the Data

The house and apartment corpora were recorded on cassette tape. There were occasional technical problems, such as the sound of wind blowing interfering with one speaker and some words on all the tapes simply being inaudible. My goals in transcription were twofold: to transcribe all the words I could (including filled pauses and interjections like "ah" and "mm") and to record the *segments* in which the text was produced. The segments correspond to my judgement of text that composes a unit or "happens all of a piece." This judgement is necessarily subjective. It is based primarily on pauses (which separate segments) and intonational contours (which span segments). There are methodologies and notations for recording this information (for example, see Pierrehumbert (1980) on intonation and Atkinson & Heritage (1984, pp ix-xvi) for other phenomena such as pauses, restarts, and overlaps). However, due to lack of appropriate equipment, I was not able to conduct such analyses.

The transcription conventions I have used are the following:

1. Each line of text consists of one segment.
2. No sentence punctuation is used. The characters "!" and "?" indicate intonation only.
3. An ellipsis, "...", indicates a discernible pause.
4. A dash, "—", indicates a restart, self-interruption, or other-interruption . This may either occur within a word or the interruption may be followed by something completely different.
5. When two speakers talk at once, overlap is not indicated but the speakers' talk is interleaved as accurately as possible.

2.1.3 Related Work in Living Space Description

In this section, I discuss related research on descriptions of houses and apartments, or "living space." Linde's work on apartment layouts (1974) is seminal in this area. Ullmer-Ehrich (Ullmer-Ehrich 1982, Ehrich & Koster 1983), has focused on individual room descriptions rather than layouts of rooms. Shanon (1984) conducted similar research, though he studied written descriptions. At the conclusion of this section I summarize psycholinguistic researchers' observations on the structure of living space descriptions.

Linde

The choice of domain for this thesis has been motivated by Linde's work with apartment layouts (1974). Linde gathered 72 spoken descriptions of apartment layouts from New York City residents, who were unaware that she was doing research in sociolinguistics. For about half the descriptions, she also obtained a sketch of the layout by the speaker.

Linde conceived of an underlying knowledge model of apartment layouts as "a system of spatial relations involving points and positions." She formalized this model as a phrase structure network. The points in her network are rooms, and she feels that the unmarked (default) strategy for a layout description is to build up the apartment room by room, starting at the front door and traversing the rooms as though one were conducting a tour. Some rooms, *one-room branches*, which have a single entrance, may be visited by the tour guide "standing in the doorway" and indicating them. *Multi-room branches* must be "entered," so that each room may be visited.

Linde found that people divided the living space in an apartment into three classes: *hallways*; *major rooms*, e.g, kitchen, livingroom; and *minor rooms*, which are non-obligatory, like a study, or have special (e.g., taboo) social status like a bathroom. The existence of major rooms can be presupposed, with the consequence that such a room can be introduced with "the," even if it has not been previously mentioned.

As the apartment is described, two pieces of information must be mentioned about each room: its *existence* and its *location*. Generally, these pieces of information are realized as a noun phrase and a locative phrase (such as "on the left"). Based on her analysis of the descriptions, Linde suggests four levels of structuring decisions in a layout description. First, the speaker chooses a strategy, either a *map*—describing the apartment as though one were looking at an architectural drawing of it, or a *tour*—describing the apartment as though one were leading someone through the rooms, in some hypothetical sense.

Second,[1] the speaker must decide the order in which to visit the rooms of the apartment, that is, how to conduct the tour. Linde claims that a tour is composed of optional introductory and closing summaries, and an establishment of a starting point (usually the front door) followed by the actual tour. Clearly, establishing a starting point is an important step; it constrains the course of the tour and provides an anchor for deictic terms, such as "left" and "right," which speakers use to explain spatial relationships.

Third in Linde's hierarchy of decisions is "chunking" the information perspicuously. Only so much can go into one sentence; conversely, some items of information and positions in the tour require the beginning of a new sentence. For example, Linde claims that sentence boundaries are obligatory after a multiroom branch and before a branch with a major room, while they are optional before a minor room.[2]

The fourth stage includes those choices made between two sentence boundaries. These choices include tense, subject, and definite and indefinite articles.

In presenting her four levels, Linde does not claim that a speaker makes all the choices necessary at one level before proceeding to the next, more finely grained, one. The levels analysis is not a description of a hierarchy of planning, with an enforced sequence of decision-making. Instead, the levels represent a description of the processes which convert knowledge to speech: the earlier levels contain decisions which are less language-dependent, while in the later levels the decisions are closer to language. There is, of course, some necessary ordering: Linde finds it reasonable to assume that a strategy is selected before actual sentences are constructed (though the strategy may subsequently be switched). However, it is also quite plausible that availability of certain lexical and syntactic constructions will motivate the choice of strategy.

It is clear that these levels are interleaved. Linde's analysis shows that hesitations—pauses in the production of language utterances—are most frequent just before difficult parts of the description, as well as at the beginning of the description itself. (Difficult parts, including the beginning, are points in the description where many choices are available to make.) People are also likely to produce sentences with less dense semantic content and simpler syntactic form in these places. It would seem that speakers temporize by "thinning out" their linguistic output while they are planning the difficult bits to follow.

[1] Since the bulk of Linde's protocols used the tour strategy, her theory tends to assume it. Though she is not explicit on this point, I suspect that most of her claims have been validated only for descriptions using the tour strategy.

[2] Since Linde's data are transcriptions of spoken descriptions, which do not have punctuation, these sentence boundaries are inferred.

As Linde is careful to point out, her data are colored by being drawn from a particular linguistic community, living in a particular sort of dwelling. This does not detract from her analysis, which is extensive and thorough. However, it does suggest that we should not be surprised that her data differ from mine in certain respects.[3] For instance, Linde's speakers rarely make any reference to the shape of the perimeter of their apartments, in contrast to the speakers in my sample (see, for example, the second excerpt in figure 2.2). It is perhaps not surprising that people living in houses with yards are more aware of the overall shape of their living space than are people who live in apartments in large buildings. It is possible that this lack of awareness is partly responsible for the rarity of the map strategy in Linde's corpus–she found it in only three descriptions out of 72. At least two of my seven speakers (Ann and Keith) used the map strategy for part of their descriptions; another speaker (Claire) used a strategy of naming a room, one of its walls, and the room on the other side of the wall. Given the prevalence of these counterexamples in my data, the tour strategy may not be as deeply cognitively motivated as Linde suggests. However, whether the strategy is tour, map, or wall-to-wall, the basic underlying organization remains spatial proximity.

Ullmer-Ehrich

Ullmer-Ehrich (1982) gathered descriptions of individual dormitory rooms, embedded in interviews on dormitory life. The descriptions focused on the spatial relationships among the furnishings of a room. She found, as might be expected, that little movement was involved in a description; typically, everything can be "seen" from a single point of view. Physical proximity has a strong influence on the order in which objects are introduced—a common organizing strategy has the form of a *gaze tour*, which sweeps from one object to the next.

Ullmer-Ehrich demonstrates that locations are described with respect to a *reference frame* consisting of at least a *reference object* (e.g., "the stove is to the left of the sink"), and possibly also including a *reference place* and the *reference orientation* (respectively, a statement of an imaginary observer's position, and the direction he is facing). Then the gaze tour typically follows the establishment of the reference frame, with the reference frame held constant. If the description starts at the door, the reference orientation may suffice. The room's shape and size are often mentioned. Mention of the reference object tends to precede reference to the object being located; place and orientation, if included, are most frequently introduced in a conditional clause (e.g., "if we're facing the living room from the kitchen area or hallway"). The reference orientation tends to remain constant within a single room; thus once a gaze tour is initiated, further steps can be abbreviated to "and then", indicating that the tour continues in an unmarked way.

Ullmer-Ehrich found two principal ways that speakers accomplished the gaze tour: *round about* which follows adjacent walls, and, less common, *parallel line*, which involves organizing mention of contents by following several imaginary parallel lines in the room. At a lower level of organization were again found two different principles: *sequencing*, which locates each object with respect to the previously mentioned one and *grouping*, which locates several objects with respect to a single one located among them. Functional arrangements of furniture tend to be described by the latter strategy. Grouping tends to induce the parallel line strategy, while sequencing induces the round about strategy. Subjects try to avoid perspective ambiguities through temporal expressions (which mark the next "spot" on the tour) or explicit links to the reference frame.

Ullmer-Ehrich's analysis leaves open the questions of how descriptive strategies are chosen and what factors take precedence in their selection.

[3]Somewhat problematically, Linde does not include all of her raw data in her thesis, and data she does include have been cleaned up. (I have compared them with some of the raw data that I have obtained from her.)

Ehrich and Koster

Ehrich and Koster (1983) built on Ehrich's previous work, this time examining room descriptions in Dutch.[4] This study was experimental: it involved familiarizing subjects with a miniature room with dollhouse furnishings and then asking for a description of the room or part thereof. They manipulated the arrangement of the furnishings; they were particularly interested in the effect on room layout description of how subgroups of furnishings were described. In one set-up the furniture was functionally arranged (into sitting, studying, and dining areas) and in another there was no functional arrangement. In addition, the position of the room's door, through which subjects were required to view the scene, was moved to different points along one wall of the room.

The classification of strategies described by Ullmer-Ehrich (1982) was used. It was hypothesized that the position of the door, and thus the viewpoint, would have an effect on the choice between parallel line and round about strategies, but this hypothesis was not borne out. Nor did the position of the door have an impact on the overwhelming preference of speakers to start at the left rather than at the right. As the researchers expected, speakers were more likely to give a parallel line description of a functional arrangement and a round about description of a non-functional arrangement. However, the grouping substrategy of the parallel line strategy was not only used for functional groups: it was used as well to describe a group of furniture in which one piece, the bookcase, was much larger; the other pieces were described in relation to it.

Ehrich and Koster examined clauses that related an object (using a noun phrase) to its location (using an adverbial phrase, such as "in the corner"). While either order is grammatically acceptable in Dutch, the location was almost always mentioned before the located object. They also noticed that speakers who were more likely to mention objects before their locations gave descriptions that were less systematic and less complete. Ehrich and Koster infer from these observations that speakers use spatial information to structure their descriptions.

Shanon

Shanon (1984) studied descriptions written in Hebrew by subjects about rooms familiar to them. He was interested in discovering regularities in how the parts of the rooms were related to each other. He found the following rank ordering, in which items in the later classes of items are more likely to be mentioned by relating them to items in the earlier classes, while items in the earlier classes are more likely to be used as anchoring points for items in later classes. This is probably related to both the size and the mobility of the objects in question. This ranking is supported by the observation that different syntactic constructions (such as definite articles and relative clauses) cluster at different levels in the ranking.

1. the room proper
2. parts of the room (e.g., walls)
3. windows, doors
4. major pieces of furniture
5. objects with a definite place of their own (e.g., a phone on the wall)
6. objects without a definite place of their own (e.g., books)

[4]Ehrich and Ullmer-Ehrich are the same person.

Taking pairings of an item at a lower level related to an item at a higher level as the unit of the description, Shanon next examined how these units were put together. He found that the same constraints applied at this larger grained level, with the overall effect that items at each level usually appeared earlier in the description than most items at the next level down. Simply, this means that the room itself was mentioned before the furnishings that were mentioned before the movable objects. Shanon found, however, that if tree structures were used to represent the order of mention of objects and what each object was mentioned in relation to, then a single tree did not suffice. That is, the description did not relate every object to a root *room* node. Instead, most descriptions could best be represented by an ordered set of trees, each one rooted at some level 2 object. The subtrees themselves are ordered (no node dominates a node of a higher level). Single node subtrees usually indicated a forgotten object added as an afterthought. When a description violates these constraints, this is usually linguistically marked in the text, with, e.g., "I forgot."

Shanon advocates the view suggested by his analysis that there is a global top-down organization to a room description, implying a planning phase. The fact that the descriptions he studied are written perhaps accounts for the more top-down structure found in his texts than found in spoken texts. On the other hand, an alternative analysis of his data might reveal that locally organized description strategies account equally effectively for the texts.

2.1.4 Structure and Linearization

The work in the preceding section provides a welter of evidence that spatial proximity and the spatial structure of whatever is being described is an important resource to a speaker or writer who is structuring a text. Similar observations have been made in the study of other domains, such as the arrangement of place settings on a table (Ehrlich & Johnson-Laird 1982).

Living space and other spatial description has been of special interest to psycholinguists because it is a good site for the study of *linearization.* Space is three-dimensional and language is one-dimensional—spoken language happens in time, and must be produced sequentially. The story goes that somewhere between a speaker's perception of a three-dimensional environment (or a two-dimensions map or diagram) and his uttering of a one-dimensional stream of text, the three (or two) dimensions need to be collapsed into one; this is linearization. One branch of the investigation into this phenomenon addresses the issue of what someone's representation, or *mental model*, of a two- or three-dimensional domain is like. Extensive work on how features of diagrams and maps affect people's texts about them and how features of texts can affect reader's subsequent drawings is being conducted, most notable by Tversky and her students (Tversky 1990, Franklin & Tversky 1990, Taylor & Tversky 1990).

In an experiment to specifically test Linde's claims about linearization in house layout descriptions, Levelt (1982) accepted Linde's assumptions that an apartment is mentally represented as a network of rooms, and gave subjects the task of describing pictures of nodes and arcs. Levelt claims that the strategies people use for linearizing are domain independent.

The networks included *choice points* (branches) or *loops.* Levelt developed two models for how speakers can handle choice points and loops, one of which is easier on the speaker and the other of which is easier on the hearer. The *speaker-oriented model* builds tour-like descriptions preserving maximal connectedness and makes jumps only when necessary (and, even then, only back to the nearest choice point). This works fine with choice points, but loops can be handled in two ways. In what Levelt calls the *dumb* way, the choice point at the beginning of the loop is pushed onto the stack (so it can be jumped back to) but then must be popped unnecessarily because it was not used. The *clever* way would be to recognize the loop and not do a push (in essence, tail recursion). While

the clever loop strategy reduces the load on the *speaker* in this model, a *listener*, if not warned that a loop is being entered, would have a bit of a problem. In this case, the listener would not be able to use the clever procedure, and if the network gets complicated, the recursion gets deep.

Levelt suggested that a *listener-oriented* model would be iterative, and instead of pushing choice points, simply counts the number of unfinished nodes. It uses another network to return back along a loop until it encounters an unfinished choice point. Because of this, loops are difficult for the speaker, and can only be dealt with at all by the clever method of recognizing a loop whenever it is encountered. However, the descriptions will be longer, because of the return moves, and if the speaker fails to note a loop, a description will not be complete.

While Levelt found that speakers did use description strategies according to one of these two models when describing linear or branching networks, all subjects had troubles with loops, and often treated a loop as a two-branch structure without reconvergence, just as did Linde's apartment describers. (Linde pointed out that if a speaker lived in an apartment in which some set of rooms were arranged in a loop, the speaker usually described the loop in two parts, as though it were a binary branch, and never alluded to the existence of the loop.) This observation argues for local construction of descriptions: the information that a loop exists may not be readily available to a speaker who is choosing what to say next from the rooms that are next to the one he has just mentioned.

2.1.5 Analysis of House and Apartment Data

In this section, I look at the structure of the living space descriptions and then examine more closely some of the language used to express the objects and relations that people use in building descriptions in the house domain.

The house and apartment data provide evidence that people's descriptions of living space are structured by the spatial structure of the subject matter. (They also provide evidence that the descriptions are structured by other considerations; however, it is only the spatial structuring that is explicitly modeled in Salix.) As described in section 2.1.3, Linde found that her speakers constructed tree structured traversals of the rooms of an apartment. My speakers, perhaps in part because they talked about more than just the rooms, did not build such nice structures. To demonstrate this, I took a sketch of the house and for each description I constructed a traversal of the picture, putting a mark on top of each object mentioned, and connecting the marks with lines. Thus I had a diagram of the path each speaker's description took with respect to the physical layout of the house. These paths look like plates of spaghetti. One is reproduced in figure 2.3; unfortunately the computer drawing program does not allow me to do it justice. Each of the choices makes local sense, as a choice of *what to say next.* The description itself is globally coherent in virtue of these local choices, but the global structure itself is not much to look at.

Another reason that Linde's descriptions seemed to mostly fall into a particular pattern is that most of her speakers started at the front door of their apartments. This is probably an artifact of New York apartments, which are generally approached only via the front door, whereas houses, and even apartments in houses, can be approached from all directions. Flexibility in choosing a starting point for a description again suggests local organization—you can start anywhere and go from there. Starting points for descriptions in my data (both house and apartment) are given in figure 2.4. In cases where a speaker started over, only the first starting point is included.

The five house descriptions and two apartment descriptions that I studied took up about an hour's worth of tape and together comprised about 5000 words of transcribed text. One could spend

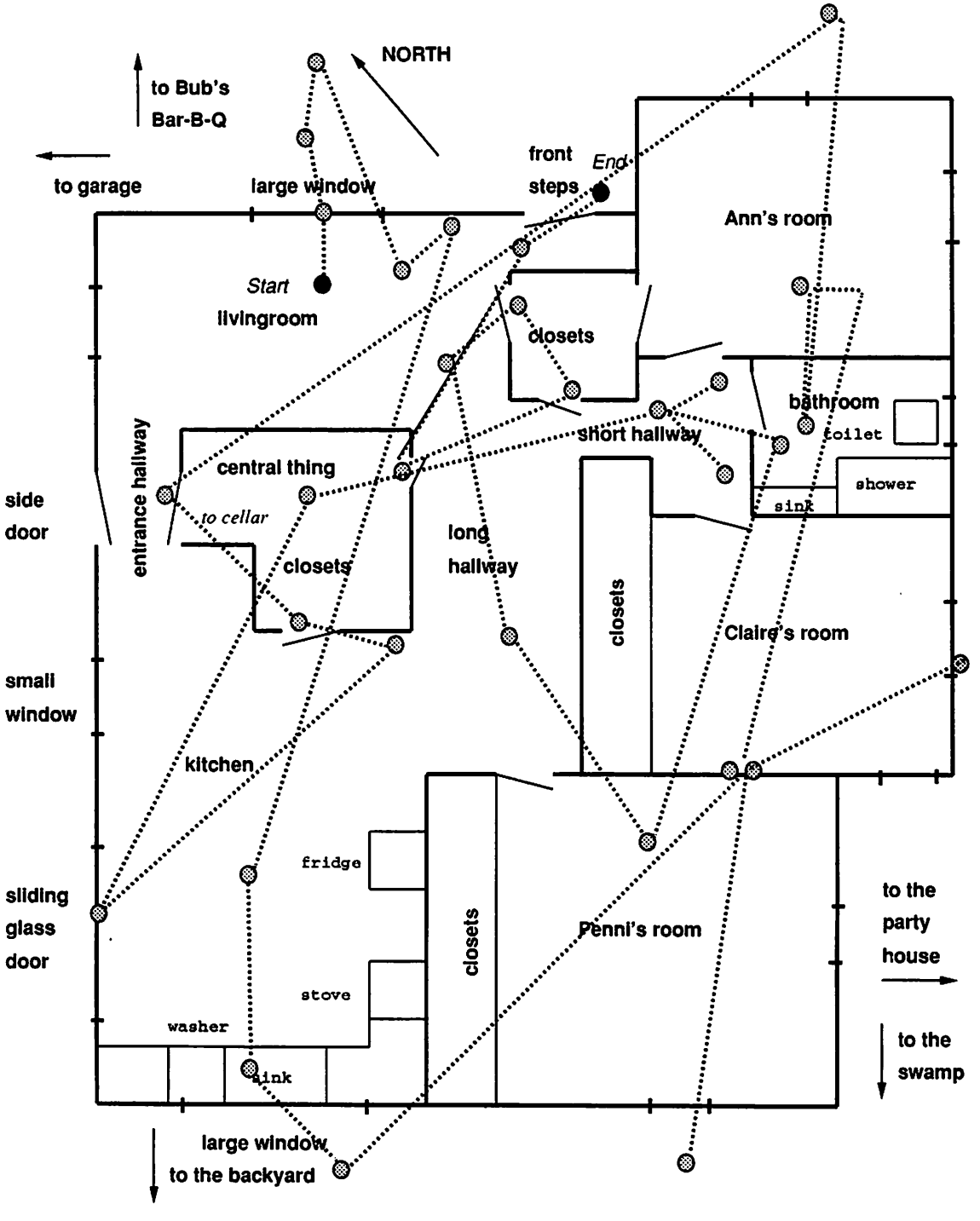

Figure 2.3: Diagram of an order of mention of physical objects in the house. This order emerged in the course of the house description by Dick (see text in A.1.3).

Starting point	Speakers
Own room	1
Own room/front room	1
Front of house/livingroom	1
Large window in living room	1
Garage	2
Den	1

Figure 2.4: Starting points in house and apartment descriptions.

years analyzing that much data; every time I examine it I discover something different. These data inform the type of text that Salix produces in the house domain, but much of that in an inexplicit way results from my seeing what Salix comes up with, going back to the data, and trying to tune Salix to be (in my opinion) more in line with the data.

The house text that Salix produces forms a small subset of the sorts of text in the corpora; it is not nearly so varied. The reason for this is simple: most of the time I have no model for why one locution is used one time, and then at another time, a very different one in a similar context, by a different, or even the same, speaker. Most of the options I do model, such as the choice between indefinite and definite article ("a" and "the") for introducing objects are modeled by random selection. This is the best model for two reasons: I do not know of a better algorithm; and the variety makes for much better sounding text than consistently choosing one option or the other.

I looked at how objects are introduced in the house and apartment corpora; the results of this analysis are summarized in figure 2.5. In this grammar <xthere> is "there", <copula> is a form of "be", <def-np> is a definite noun phrase (using the definite article "the"), <indef-np> is an indefinite noun phrase (with "a"), and <deictic> is a phrase like "on the left." This grammar for object-copula-deictic phrases is the grammar used by **say-object**, section 7.4.7. 199z As it turns out, the use of deictic terms interacts with both the use of existential "there" and (indirectly) with the indefinite/definite article distinction. The existential "there" usually takes an indefinite noun phrase, and it always requires a copula. If there is a copula, there must also be either an existential "there" or a deictic phrase.

Definite and Indefinite Articles

Linde points out that in her data, rooms may be introduced with either the definite or the indefinite article (Linde 1974, p 169). (Recall that Linde considered only mention of rooms in her apartment layout descriptions.) Contrary to the general supposition that an item must be introduced into a conversation with the indefinite article "a" and then may subsequently be referred to with the definite article "the", Linde found that 38% of room introductions were made with "the". She suggested that rooms more often introduced by "the" were ones that could reasonably be supposed to be known to be part of a house, whereas more unusual rooms were more often introduced with "a". Her data showed that the kitchen was introduced with "the" 28 times while the dining room, which is not a feature of all apartments, was introduced with the definite article only five times. Linde hypothesized that in general speakers could introduce a room with either article, but the

```
                    <def-np/indef-np>
<xthere>   <copula>  <indef-np> (usually)
<deictic>            <def-np/indef-np>
<deictic>  <copula>  <def-np/indef-np>
                    <def-np/indef-np>     <deictic>
<xthere>   <copula>  <indef-np> (usually)  <deictic>
```

Figure 2.5: Grammar for introducing objects in living space descriptions.

more usual a room was the more likely speakers were to use "the" and the more unusual the room the more likely to use "a".

Since my interest lies more in modeling possible variation than in coming up with an exact description of my corpora, I turned the question around. Instead of detailing all the instances of definite and indefinite articles, I examined how in fact my speakers introduced rooms and other features of the house and apartment. Some examples may be found in figure 2.6. We can make several general observations. First, some of these introductory noun phrases are quite elaborate. Second, there are several instances in which no article is used at all, or at least none is audible on the tape. (Linde does not address the phenomenon of a missing article). And finally, each of these objects may be introduced by either a definite or an indefinite noun phrase.

A striking example of the mixture of definite and indefinite articles appears in an excerpt from the description in figure 2.1. Here we see "the stove" immediately followed by "a refrigerator"; it is hard to imagine a distinction between these two objects that would account for the difference in article type.

An *existential there* is so called because it posits the existence of something. It is found in constructions such as:

```
then in the kitchen
there's a large window
```

Existential-there constructions are more likely to be used with indefinite than definite noun phrases, presumably for the same reason of introducing an object into the text for the first time. An existential there has the additional effect of indicating that this object is distinct from whatever was just under discussion. Existential-there constructions also interact with locative expressions.

Deictic Locatives

As Linde notes, many clauses in a living space description contain three parts: an object, a copula (usually a form of "be"), and a locative phrase. A locative phrase is one that expresses a spatial location, like "here," "next," or "on the left." In this section, I examine the cases of such clauses in my data in which the locative includes an explicitly spatial deictic phrase. These deictics are "front," "back," "left," and "right." While these spatial terms seem characteristic of descriptions of apartments and houses, which are, after all, spatial domains, three of the seven speakers in my corpora (Keith, Hannah, and Mike) never use any of these terms.

Object	Introductory noun phrase
	In the house corpus
kitchen	the kitchen (4)
	a combination kitchen-dining room (1)
bathroom	the bathroom (2)
	three bedrooms bathroom (1)
	three bedrooms bathroom ah livingroom (1)
garage	the garage (1)
	a garage (1)
	a two-car garage (1)
basement	the basement (4)
	a basement (1)
front door	the front door (2)
	a front door (2)
	this funny front door of ours (1)
sliding glass door	the sliding glass door (1)
	a sliding glass door (1)
	sliding glass doors (2)
	a front door side door and sliding glass door (1)
piano	the piano (2)
	a piano (1)
	In the apartment corpus
kitchen	a sunporch which has been converted into a kitchen (1)
	livingroom kitchen together (1)
bathroom	a large bathroom (1)
	a very large bathroom (1)

Figure 2.6: Definite and indefinite articles in introductory noun phrases. Examples are drawn from both the house descriptions and the apartment descriptions.

These are typical examples:

```
there's a front door on the righthand side
on the lefthand side is the door to the basement
```

2.2 The Family Corpus

2.2.1 Gathering the Data

The family corpus is a set of descriptions of my extended family, collected at a Thanksgiving dinner. I took family members individually into a separate room and asked:

"Can you tell me how everyone who comes to Thanksgiving is related to each other?"

Once again, the interviews were taped on a portable cassette recorder.

Fifteen of my relatives provided texts describing the family. All of the speakers are over 18. As with the house descriptions, speakers were sometimes prompted for more detail, for example, when they offered answers like "we're all Smiths" or "we're grandparents, aunts and uncles, and cousins." In these cases, I asked them to be more specific, with varying degrees of success.

That both I and the speakers were part of the family being described almost certainly had an effect on the texts produced, but since there are no control texts gathered by an interviewer outside the family, this effect cannot be evaluated. I was rather surprised by the emotional reactions the question engendered. I had thought that this would be a straightforward question for my family, since I knew that the complexities of the relationships (including a half generation) and the terminology (such as the difference between a second cousin and a cousin twice removed[5]) were frequent Thanksgiving dinner table topics. But some speakers felt I was challenging or testing them in some way. Others invoked arcane rules of courtesy and saw the task as one of saying the right things about the right relatives. Still others managed to make rather pointed, and not necessarily repeatable, observations about family members when they mentioned them.

My goal in collecting the family descriptions was to see how people structured texts organized by family relationships. The texts I gathered certainly shed light on this issue. However, the sorts of things that people said about family members once they had mentioned them were too free-ranging and idiosyncratic to model. For this reason, and out of consideration for my family's privacy, I do not include the entire set of texts in this thesis (though an example text appears in figure 2.7). Also, the family texts that Salix generates are restricted only to mentioning family members and the relationships between them. I have extracted from each text the order in which the family members are mentioned. These orderings are included in appendix B. Graphic representations of two of these orderings, including the one corresponding to the text in figure 2.7, may be found in figures 2.8 and 2.9. This ordering information serves as a model for how family texts are structured.

[5]For the curious, this is the difference: your *nth* cousins belong to the same generation you do. The degree is determined by how many generations back you have ancestors in common. Your first cousins are the children of your parents' siblings; thus you and your first cousins have grandparents in common. Your children and your first cousins' children are second cousins—they have great-grandparents in common. Your *removed* cousins are never in the same generation that you are. The number of times the cousins are removed indicates the number of generations that separate you. Your cousins once removed are your parents' cousins or your children's cousins; your cousins twice removed are your grandparents' or grandchildren's cousins. You can in fact have second cousins twice removed (and almost certainly do), but very few people go to such extremes of calculation!

there's there's my mother Katharine
n my father John n my sister Penni and me
and my mother's—it's all my mother's relatives
that we go to see
um the people whose whose house we go to are Margaret and Bill
who are Mommy's
....um sh—Margaret's my great-aunt
so it must be Mommy's aunt
um and Uncle Bill
um Margaret's sister Alice sometimes comes
depending on what else is going on
um Margaret's children come
and there's Becky
who's the youngest
n Billy
....and Ann and her husband Bennett
and their children Nathan and Rebecca
um then there's Martha sometimes comes
Martha with her husband Pat
and their children Lara and Samara
and the oldest is Diana whose husband is Paul and they have two
children Kathy and David
um Alice's children
um sometimes come
there's David
who's married to Jane
there's Carlton
who's married to Carol
and they have Christopher and Sarah for children
um there's George
who's also known as Jeff
who's married to Vi
and has three children
Alison Jeffrey and....Andrew
um and....Alice has another daughter Dorothy
who I don't think has been to Thanksgiving in years
and she has four ch—she's married to Jim she has four children
who are Jonathan Ben Andrew and....Sarah
no not Sarah
I don't remember her name
um....let's see if that all Alice's children
....yes
and also um Eleanor and Elizabeth come
who are....cousins of....all of us
um I don't know what generation cousins they are
um....n I uh who else has come in the last years
....I think that's about it

Figure 2.7: Representative family description.

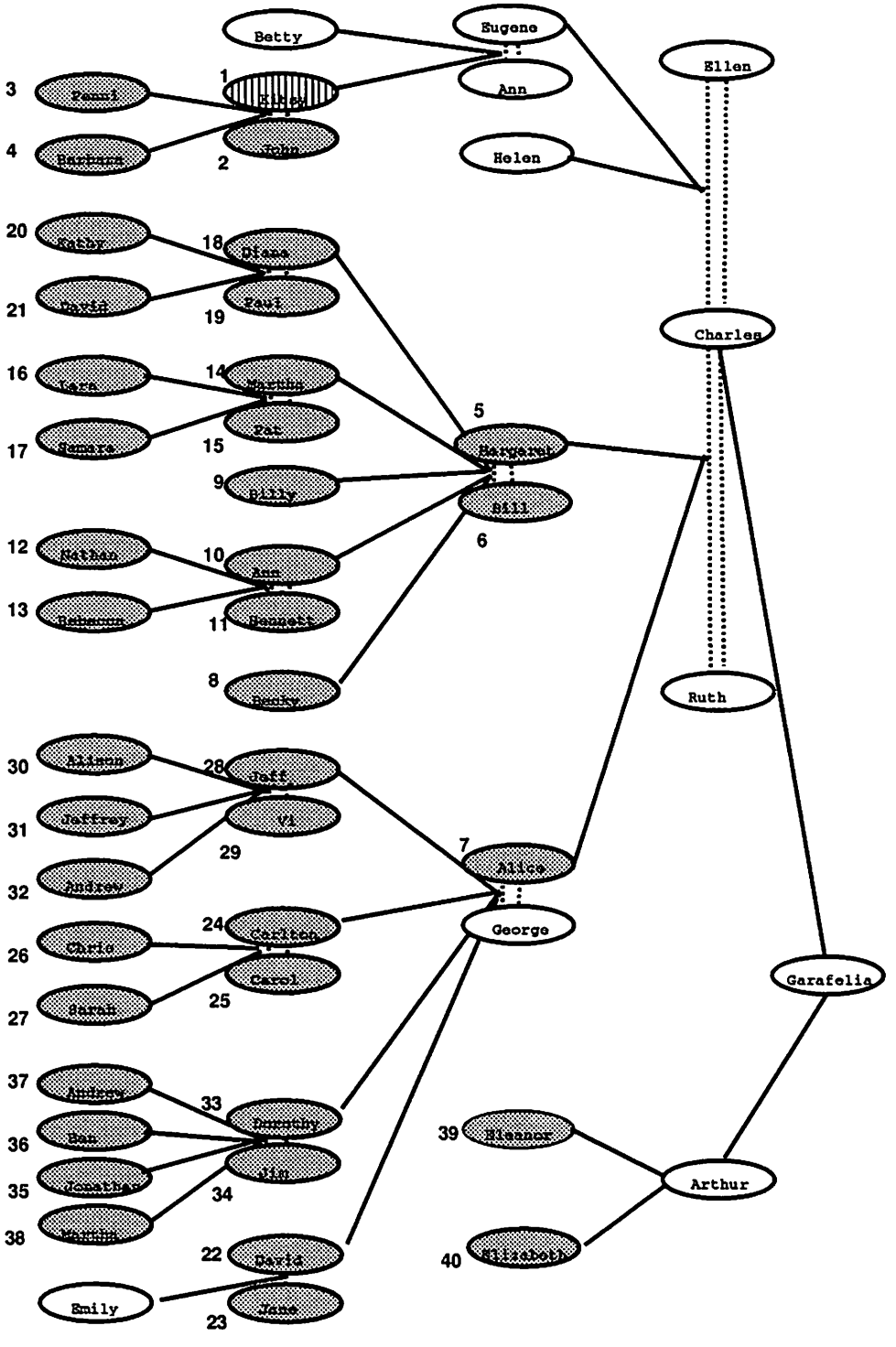

Figure 2.8: Diagram corresponding to Barbara's text. (See figure 2.7.) Double stippled lines indicate spouse relationships; solid lines parent-child.

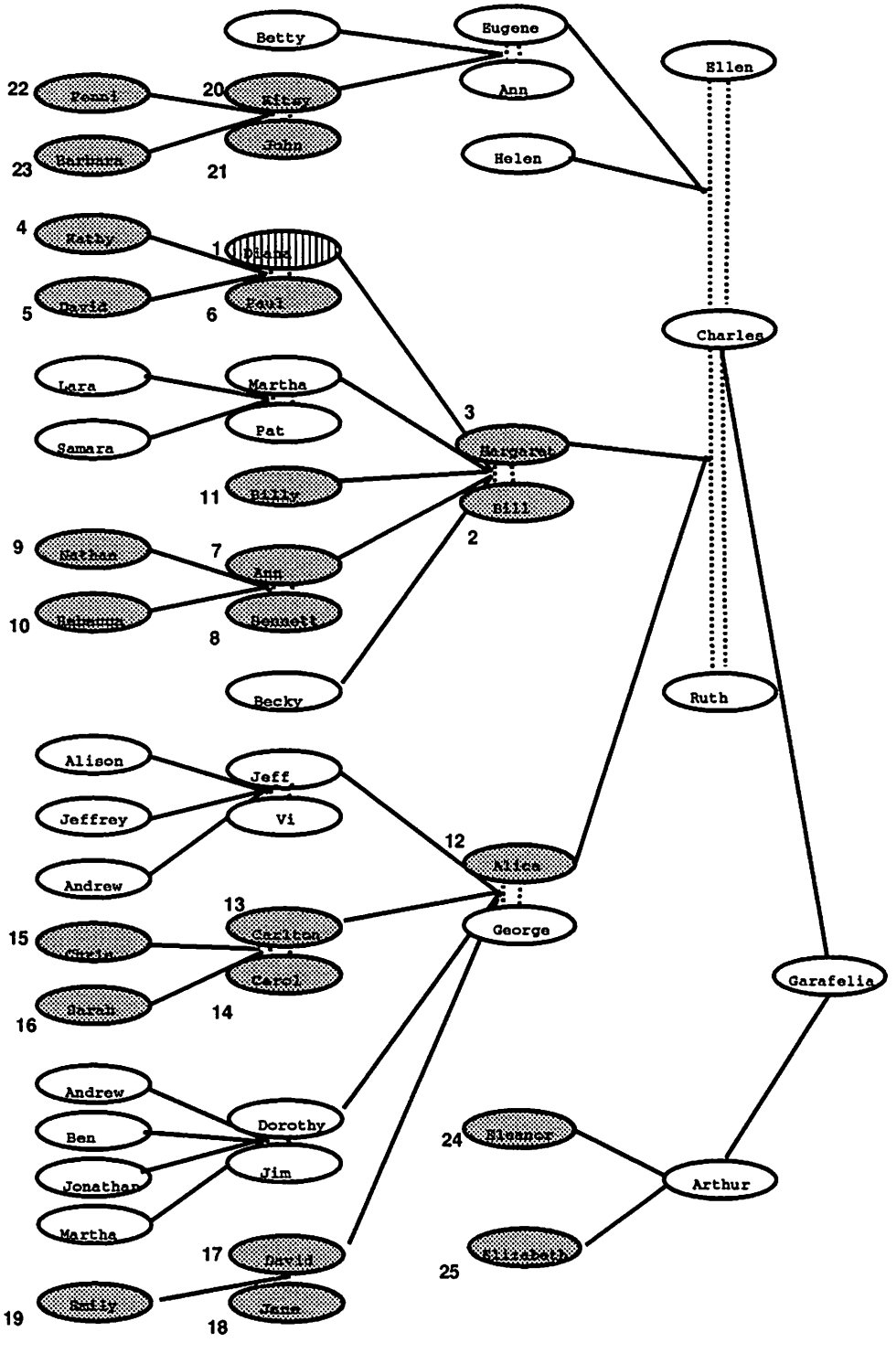

Figure 2.9: Diagram corresponding to another family text. Double stippled lines indicate spouse relationships; solid lines parent-child.

2.2.2 Analysis of the Family Data

Starting point	Speakers
Self	1
Interviewer	1
Hostess (and host)	8
Own mother	2
Own mother/hostess	1
Common ancestor	1
None	1

Figure 2.10: Starting points in house and apartment descriptions.

In determining the structure of a description, the aspect of most interest in the family descriptions is the order in which family members are mentioned. Figures 2.8 and 2.9 diagram the order of mention in two descriptions; the first figure corresponds to the text in figure 2.7. Starting point varied, as can be seen in figure 2.10.

It can be seen that by far the most frequent starting point is the hostess of Thanksgiving dinner (sometimes mentioned in conjunction with the host). The next most common starting point is the

```
<mention>:
there's <persons>
<persons> come(s)
<persons> is/are here

<persons>:
<person>
<person> and <persons>

<person>:
<name>
<relationship> <name>
<name> who is <relationship>

<relationship>:
<possessive> <rel>
<rel>

<rel>:
mother, sister, etc.

<possessive>:
<persons>'s
my, their, etc.
```

Figure 2.11: Partial grammar of language used in family descriptions.

um the people whose whose house we go to are Margaret and Bill
who are Mommy's
....um
sh—Margaret's my great-aunt
so it must be Mommy's aunt

you are my niece
no you're my grand-niece
uh your mother is my niece

and also um Eleanor and Elizabeth come
who are....cousins of....all of us
um I don't know what generation cousins they are

Figure 2.12: Expressing relations in family descriptions.

speaker's mother; in one case the speaker's mother was also the hostess. One speaker started with herself and one with the interviewer. One speaker started with a common ancestor; interestingly, this was the hostess. One speaker was unable to give a specific description at all, and thus there was no starting point.

The most common form of description is to name someone, name his or her spouse and children, if any, or parents, and name someone else. This someone else may be one of the named spouse/children/parents, or an as-yet unmentioned person. This person is usually a close relative of the first person mentioned (e.g., is a sibling of that first person). When there is no close relative left who is unmentioned, a more distant relative is chosen.

Family members are referred to in two ways: by name or by relationship. Examples of the latter include "my wife," "your mother," "her children." Often, both forms of reference are used (e.g., "his wife Carol") but sometime just one or the other. Most relationships mentioned are one-step, i.e., parent, child, spouse, or sibling. Sometimes longer distance relationships are worked out. Most two-step relationships mentioned—grandparents, grandchildren, aunts, uncles, nieces, nephews—are given fluently. But more distant relationships are worked out out loud, and often involve consultation with the interview. (They are also often mistaken.) Figure 2.12 shows two examples, the first drawn from the text in figure 2.7.[6] Often the precise relationship is inexplicable, as in the last fragment in figure 2.12 (this fragment is also drawn from the text in figure 2.7).

The descriptions suggest that local relationships are accurately remembered by speakers but that relationships between people less closely related are vaguely and also incorrectly remembered. This observations provides support for the representation of families that is used in Salix. In Salix's representation, the default relation of belonging to the same family connects all the family members each to the other. Beyond that default relation, only one-step relations are explicitly represented.

The text in the bulk of the family data is similar to that in figure 2.7. That is, patterns like those represented in the grammar in figure 2.11 are used.

[6]Some people use the term "grand" for nieces and aunts removed one generation, and some use the term "great." Both terms are considered correct.

The phrase "who's married to" in figure 2.7 is a variant way to mention a spouse relationship; it is not modeled in Salix, though it easily could be. Mention of family members is elaborated in various ways, such as by reference to their birth order or how frequently they come to Thanksgiving. None of this elaboration is modeled in Salix's descriptions.

Notice that in the text in figure 2.7, "come" (as in "<person> comes") functions similarly to "there's" (as in "there's an <object>") in the house domain. That is, the verb is used in an essentially existential, or semantically neutral, way. In other speakers' texts, "is/are here" performs a similar role.

Chapter 3

Implementation

This chapter describes the implementation of Salix exclusive of the spatial-specific strategies described in chapter 5 and the surface structure realization functions described in chapter 7. This description includes the knowledge base representation language, the strategy interpreter, and the domain independent strategies. Salix is implemented in Common Lisp and comprises about 3700 lines of code.

3.1 The Knowledge Base

Salix represents its knowledge base (KB) as a network. The knowledge base representation is completely domain independent; in fact, the KB can contain several domains at once.

3.1.1 The Nodes in the Knowledge Base

Everything in the KB is of type **node**. A **node** is a structure.

```
(defstruct (node)
  print-name
  lex-item)
```

There are four specializations of **node**: **property**; **object**; **relation**; and **lex-item**. There is a fifth, **direction**, for spatial domains such as a house.

A **property** is a **node** with the additional type of *property*. Example **propertys** are female and picture-window-ness.

An **object** is a **node** with additional fields.

```
(defstruct (object (:include node))
  types
  already-said?
  relations)
```

An **object** has one or more *types*: the types classify an object more finely and constrain its suitability for the application of particular strategies. For example, in the house domain, a room is

a *room*, a window is a *structure*, and a piece of furniture or a large kitchen appliance is a *furnishing*. Other objects in the house domain have more than one type. A hallway is a *path* as well as a *room*, because it serves as a way to travel from one place to another. A walkway or a stairway is also a *path*. A door is a *doorway*, which is something that gives access from one room to another; it is also, like a window, a *structure*, which is something that is an integral part of a house.

Each object has an *already-said?* field. This field is initialized to *nil*; when an object has been explicitly mentioned in the text, this field is set to *t*. The strategies which find something to say next will not be triggered by an object that is *already-said?*. An object which is *already-said?* may be mentioned again, but only in relation to another object. For example, in the following text:

```
(1) there's Penni
(2) and then there's Barbara
(3) who is Penni's sister
```

in (1), Penni is explicitly mentioned, and marked as *already-said?*; in (3) she is mentioned in passing, in relation to Barbara.

Finally, each object has a list of *relations* in which it participates. The structure of the contents of the *relations* slot will be discussed in more detail below.

A **relation** is a **node** with four additional fields.

```
(defstruct (relation (:include node))
  inverse-relation
  superrelation
  realization-mode
  procedure)
```

Some relations have inverse relations, and these are recorded in the *inverse-relation* field. For example, composes and comprises are inverse relations, as are has-child and has-parent. It is useful to keep track of such inverse relationships, since one might have some reason to express the inverse relation instead of the one in hand, (for instance, say "Y is X's parent" rather than "X is Y's child"). In addition, both a relation's *inverse-relation* and its *superrelation* are important for making inferences about what a hearer knows that has not been explicitly said. This inference process is described in section 3.5. The superrelation is also used by **say-elaborating-relation** (section 3.3.1).

Relations may be expressed by verbs or nouns; objects are usually expressed by nouns. Both types of node make use of the *lex-item* field, which usually contains a **lex-item**. **lex-items** and the *realization-mode* and *procedure* fields will be discussed in chapter 7 on surface structure realization.

```
(defstruct (lex-item (:include node))
  verb-form        ;; ''have'' or ''face''
  v3sg-form        ;; ''has'' or ''faces''
  noun-sing-form   ;; ''child''
  noun-plural-form ;; ''children''
  prep             ;; ''in (it)'')
```

The knowledge base itself comprises a list of **nodes**, which is the value of the global variable ***nodes***.

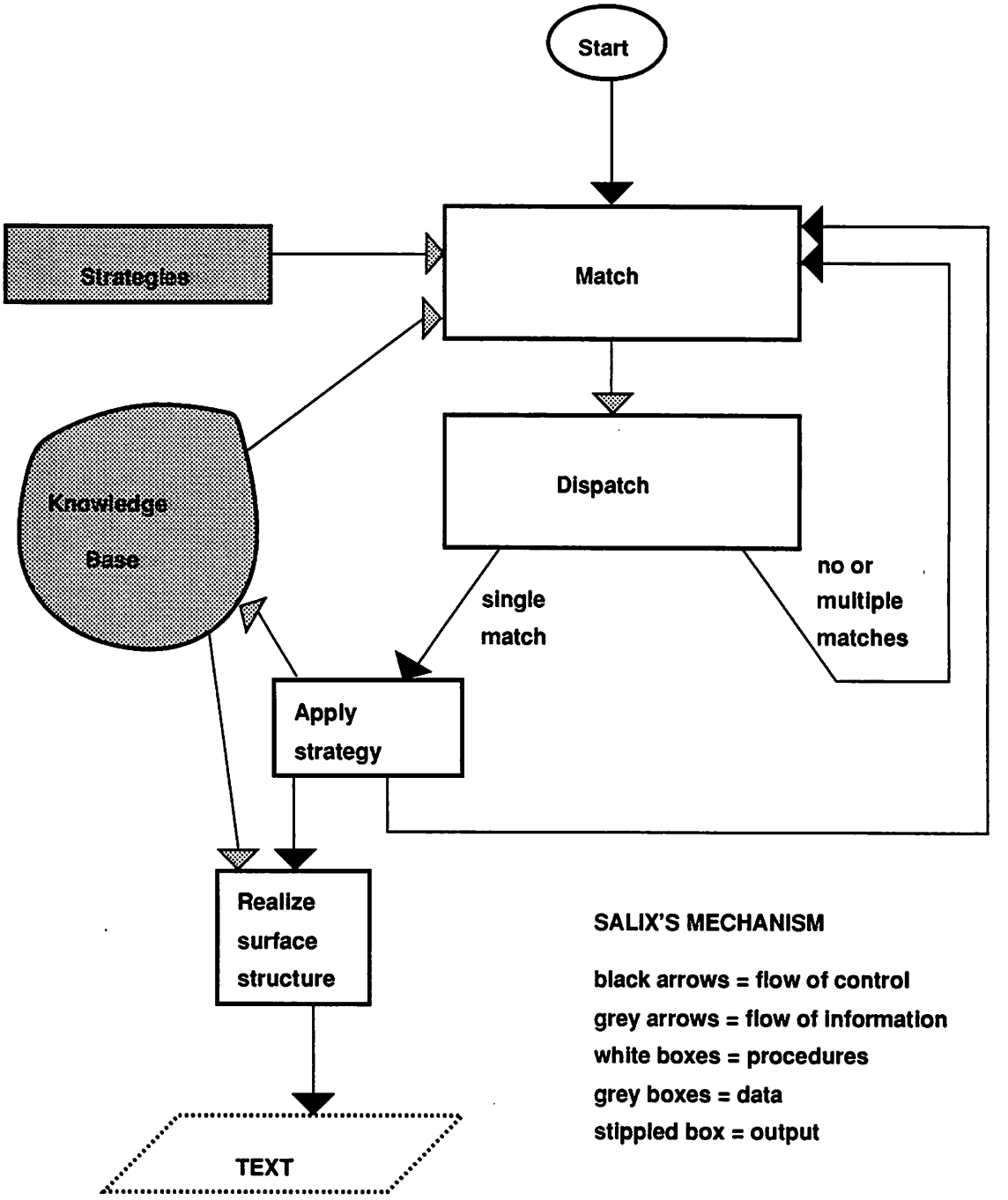

Figure 3.1: A diagram of Salix's mechanism.

3.1.2 The Links between Nodes in the Knowledge Base

The knowledge base for a domain is interconnected in terms of links between objects (and properties). Each of these links involves a relation. There are two sorts of relations: *direct relations* are explicitly represented, and links involving them are created when the KB is created; *computed relations* are relations that are computed dynamically. Computed relations are used to represent indexical or deictic knowledge, which depend on context and thus cannot be expressed at the KB creation time.

The relations field of an object contains an association list of direct relations that the object is involved in. For each relation, there is a list of *links* to objects (or properties) which are involved in this relation with this object. A link records a bit indicating whether this relationship involving this object has been said. The function

```
(related-objects-links object relation)
```

returns a list of links to objects related to this object by this relation. If the relation is a direct relation, this list is obtainable directly from the relations field of the object. If the relation is a computed relation, then its procedure is called and the list of related objects is computed dynamically. For example, in the house domain, the relation next-to is direct: when the KB is set up, all the next-to links can be directly encoded. The parts of a house do not move around, at least for the purpose of talking about it. In contrast, next-to-same-direction is a computed relation. The direction in question is the spatial direction the description in the house text has been moving in. This cannot be known *a priori*, so must be computed dynamically.

The function

```
(related-objects-to-say object relation)
```

filters the objects returned by `related-objects-links` to remove links that have been marked as already used. The list of objects *already-said?* is returned by

```
(related-objects-already-said object relation)
```

3.2 The Strategy Interpreter

Salix builds texts by means of *strategies*. The system keeps track of one node in the knowledge base at any one time: this *current node* is the node which the current increment of the text will be about and strategies are applied to this node. The system also keeps track of a dynamic *context*, which includes a variety of information such as what has just been described, whether an object has been explicitly mentioned, and what strategy has just been used. The context can also include information that is not domain independent, such as, for house texts, what spatial direction the text has been progressing in.

In this section, I present Salix's architecture.[1] A diagram of this architecture appears in figure 3.1. I first describe the dispatch cycle in which the system operates, and then show what

[1] Salix's architecture is based on a series of unpublished systems designed by Phil Agre, John Batali, and David Chapman. These systems were inspired by Doyle's SEAN (1980), which in turn was inspired by McDermott's NASL (1978) (see section 3.7.1).

```
(defun dispatch (dispatchee)
  (let ((candidates (match dispatchee)))
    (cond ((null candidates)
           (null-dispatch dispatchee))
          ((null (cdr candidates))
           (apply-strategy (first candidates)))
          (t
           (multiple-dispatch dispatchee candidates)))))


(defun apply-strategy (application)
  (apply (car application) (cdr application)))

(defun null-dispatch (dispatchee)
  (dispatch `(no-matches ,dispatchee)))

(defun multiple-dispatch (dispatchee candidates)
  (dispatch `(multiple-matches ,dispatchee ,candidates)))
```

Figure 3.2: The dispatch procedure.

```
(defun match (dispatchee)
  (let ((candidates nil))
    (maphash #'(lambda (name trigger)
                 (setf candidates
                       (nconc (funcall trigger node) candidates)))
             *strategy-table*)
    candidates))
```

Figure 3.3: The match procedure.

strategies are and how they work. I next distinguish the *saying strategies*, *finding strategies*, *metastrategies*, and *null strategies*, and describe Salix's domain independent strategies of each type. (The two spatial metastrategies are described in chapter 5.) Saying strategies realize increments of text; finding strategies find something new to say something about. Null strategies and metastrategies are triggered respectively when there have been no or multiple matches of the base-level saying and finding strategies. Null strategies are responsible for finding something to do when no base strategy applies; metastrategies are responsible for resolving conflicts among competing applicable strategies.

I give detailed examples of how Salix works in later chapters.

3.2.1 The Dispatch Process

Salix employs a simple dispatch process to decide how to continue the text from the current *dispatchee*[2] (see figure 3.2). The dispatchee can be one of two things. In the simplest case, it is just the current node in the knowledge base. Alternatively, the dispatchee may contain additional information which is necessary for deciding what to do next; in particular, the dispatchee may include a conflict set of applicable strategies.

The **dispatch** performs in two-step cycles: in the first step, all strategies that are applicable **match** the current dispatchee; then in the second step, *deliberation* is performed on the results of the match.

match returns a list of *candidates*, which are applications, that is, ways of applying strategies to the dispatchee (see figure 3.3). This function takes the dispatchee as an argument, and maps over all the strategies. **match** calls each trigger on the dispatchee, and each time there is a match, an *application* is added to the list of candidates. This list of candidates is the result of the **match**.

```
(defmacro defstrategy-internal (name var form)
  `(setf (gethash ',name *strategy-table*) #'(lambda (,var) ,form)))

(defmacro defstrategy (name var form)
  `(defstrategy-internal ,name ,var
    (if (node? ,var)
        ,form
        nil)))

(defmacro defmetastrategy (name form)
  `(defstrategy-internal ,name thing
    (if (and (listp thing) (eq (car thing) 'multiple-matches))
        (let ((node (second thing))
              (candidates (third thing)))
          ,form)
        nil)))

(defmacro defnull-strategy (name form)
  `(defstrategy-internal ,name thing
    (if (and (listp thing) (eq (car thing) 'no-matches))
        (let ((node (second thing)))
          ,form)
        nil)))
```

Figure 3.4: The strategy-creating macros.

In the deliberation step of the dispatch cycle, the result of a match falls into one of three cases: *single match*; *multiple matches*; and *no match*. If there is exactly one match, that application is applied to the dispatchee. In the cases of no and multiple matches, the system is at an *impasse*: there is nothing it can immediately do (i.e., no strategy it can apply now). In Salix, resolution of

[2] "dispatchee" seems like a clumsy name, but I have found that other plausible terms lead to confusion and misunderstanding.

these impasses is handled not by some special mechanism, but by the same dispatch procedure. In the case of no matches, a new dispatchee is created, composed of a keyword indicating that there were no matches and the old dispatchee. When there are multiple matches, there must be conflict resolution: a new dispatchee is composed of a multiple matches keyword, the old dispatchee, and all the matching candidates, and then dispatched on.

3.2.2 Strategies

Strategies have two parts: a *trigger* that is applied in the match step to see if it matches; and an *action procedure* of code for doing the strategy's job. If its trigger matches, a strategy produces a list of candidates for what to do next to continue the text with the current dispatchee; this list of *applications* is returned. For example, the strategy **say-elaborating-relation** will produce one application for each node that is involved in a elaborating relation with the current node.

The macro **defstrategy** is used to create a strategy. It it given a name and a form (the trigger), and it puts the trigger in the `*strategy-table*`, indexed by the name. The set of macros for creating all the strategies can be found in figure 3.4.

For base-level strategies, the strategy includes a test that the dispatchee is a node (since strategies only match when the dispatchee is a node), and if this is true, then the trigger is applied to the dispatchee.

If the dispatchee is simply a node, two things need to obtain for a match. First, the node must be involved in the relations which the strategy is capable of expressing. For example, one relation that the **say-elaborating-relation** strategy can express is facing (this might produce "the door faces the garage"); this strategy will match a node only if that node bears a facing relationship to another node. Second, certain aspects of the context must hold. For example, the **say-object** strategy (which produces increments like "there's a piano") will not match if the node has been marked as having already been explicitly described.

If the strategy matches on a node, each *application* returned comprises the action procedure of the strategy incorporating the node, and arguments to the action procedure, such as the relation used or the next node to be described. If an application is eventually selected, it results in the next increment of the process of producing the text.

3.3 The Strategies

The domain-independent portion of Salix involves four sorts of strategies: *finding strategies*, *saying strategies*, *metastrategies*, and *null strategies*. I will describe them in turn. The distinction between the first two types of strategies turns on that between *basic* and *elaborating* relations. The metastrategies match when there are multiple matching strategies and the null strategies when there are no matching strategies.

As far as Salix's domain independent architecture is concerned, there are two kinds of relations in the knowledge base. These two sets of relations are treated differently from each other, and all relations in each set are treated the same in deciding what sort of strategy is appropriate for expressing them. The two kinds of relations are *basic relations* and *elaborating relations*.

Basic relations provide the connectivity of a domain: the domain structure that ensures a coherent entity rather than a random collection of things. The domains I have studied can be represented in terms of either one basic relation or one basic relation that comes in slightly different

```
(defstrategy say-object object
  (if (not (object-already-said? object))
      '((say-object ,object))
      nil))

(defun say-object (object)
  (realize-object-according-to-realization-mode)
  (save-context object)
  (dispatch object))


(defstrategy say-first-object object
  (if (and (eq object *starting-point*)
           (not (object-already-said? object)))
      '((say-first-object ,object))
      nil))

(defun say-first-object (object)
  (generate-beginning)
  (generate-first-object-with-intro node)
  (save-context object)
  (dispatch object))
```

Figure 3.5: The triggers and action procedures of the strategies **say-object** and **say-first-object**.

versions. The family domain has a basic relation, related-to, which comes in several versions like has-daughter and has-father. The house domain has the basic relation next-to and its specializations which include next-to-same-direction. Basic relations are used to find the next thing to say.

Elaborating relations are relations that do not add to a text by finding something new to talk about but instead add information to, or elaborate on, the basic content of the text. In the family domain, the elaborating relations are the same as the basic relations. In the house domain, elaborating relations include has-property and contains.

What Salix needs to be told about any domain, then, is which are the elaborating relations and which are the basic relations. As we will see later in examining Salix's strategies, some of the information they use in making decisions is drawn from the knowledge base. For instance, if a pronoun is selected, it needs to have the right gender, and this information is found in the knowledge base. But this reliance on KB knowledge is not domain dependent: gender is not a domain, but a property of how language is used to refer to the world. Similarly, texts in the house domain include deictic expressions such as "left" and "right." A deictic expression is a linguistic device that is used to convey a spatial (or metaphorically spatial) relationship whenever it is appropriate, regardless of domain. Salix can construct texts without using pronouns or deictic expressions; but it is also capable of building better texts that use them.

It should be clear from the preceding discussion that Salix can construct text in any domain *providing that a basic relation or relations are available for it to exploit.* The question of what sorts of domains meet this requirement is addressed in chapter 8.

There are two types of strategies that are triggered when the dispatchee is a node: *saying* strategies and *finding* strategies; these exploit the elaborating and basic relations in the domain respectively. The saying strategies produce text about the current node and dispatch again on the current node. The finding strategies find a next node to talk about and dispatch on that node. There are five saying strategies and two finding strategies.

As will be discussed in section 4.3, the distinction between saying and finding strategies allows the system to handle phenomena usually associated with a separate focusing mechanism. These phenomena include pronominalization and cues that signal the introduction of new elements into the text.

3.3.1 Saying Strategies

say-object (figure 3.5) is the most important saying strategy. **say-object** is responsible for text that introduces the current object. It is only triggered when an object has not been marked as *already-said?*, though its action procedure is often called directly by other strategies. The general pattern of an increment produced by **say-object** is:

```
<current-object> <current-relation> <related-objects>
```

The type of increment that **say-object** produces depends on the *current relation*, which is the relation between the current object and the previous one. That is, the current relation is the link which was followed to get from the previous node to this one. The current relation may be unknown, for example, when a **jump** has happened. One might expect that, in the spirit of incrementality, the *finding* strategy that followed the relation link would realize the relation, and **say-object** would realize only the object. However, this is too restricted a view; it does not allow for the fact that not only can we say "Penni's sister is Barbara," but also "then there's Barbara, who is Penni's sister" and "Barbara is Penni's sister."

Each relation is annotated with a *realization mode* which specifies the set of surface structure patterns the text may take to express this relation. Details about how the function **realize-object-according-to-realization-mode** handles these different sets of patterns, and how context considerations select from among them, are in section 7.4.7.

say-first-object is a specialized version of **say-object**: for the first object in a text, there is no current relation; instead, the object is introduced as beginning the text.

The other two saying strategies are **say-elaborating-relation** and **say-property** (see figure 3.6. **say-elaborating-relation** is used to express the *elaborating relations* in a domain: it describes objects in relation to the current node. For example, **say-elaborating-relation** expresses the relation faces, as in "it faces the backyard."

say-property is a specialization of **say-elaborating-relation**: it expresses something related to the current node by the relation has-property. In this case, however, the something is a property, not an object, so the surface structure realization is different: it does not rely on **say-object**.

say-elaborating-relation can express a relation with a set of objects. All the family relations, for example has-father and has-daughter are expressed by **say-elaborating-relation**, as in "her parents are Kitsy and John" and "their daughter is Emily." (A set can, of course, have a single member.)

The encoding of some of the relations in the KB includes their *superrelations*. This yields a *relation hierarchy*, for example, for the family domain. When **say-elaborating-relation** expresses

```
(defstrategy say-elaborating-relation object
  ;; first of tuple is superordinate relation, rest is alist of relata and
  ;; their specific relations
  (let ((tuples nil))
    (dolist (elaborating-relation *elaborating-relations*)
      (when (or (not *inhibit-recursion*)
                (member elaborating-relation *nonrecursive-relations*))
        (let ((relative-alist nil))
          (dolist
            (relative (related-objects-to-say object elaborating-relation))
            ;; make alist
            (push (cons relative elaborating-relation) relative-alist))
          (when relative-alist
            (block maybe-merge
              (dolist (other tuples)
                (let ((lcd (relation-lcd
                            elaborating-relation (first other))))
                  (when lcd
                    ;; generalize tuple
                    (setf (first other) lcd)
                    (setf (rest other) (nconc (rest other) relative-alist))
                    (return-from maybe-merge))))
              ;; can't merge, start a new tuple
              (push (cons elaborating-relation relative-alist) tuples))))))
    (let ((forms nil))
      (dolist (tuple tuples)
        (push `(say-elaborating-relation
                ,object ,(rest tuple) ,(first tuple))
              forms))
      forms)))

(defun say-elaborating-relation (object relata-alist elaborating-relation)
  (let* ((*relata* (mapcar #'car relata-alist))
         (*inhibit-recursion* t))
    (dolist (pair relata-alist)
      (say-object (car pair))
      ;; mark the specific level relation as said, not the lcd
      (mark-relation-link-as-used node (cdr pair) (car pair))))
  (dispatch object))

(defun say-property (object property)
  (generate-property-intro)
  (generate-property property) ;; properties may be idiosyncratic
  (mark-relation-link-as-used node (get-thing has-property) property)
  (save-context object)
  (dispatch object))
```

Figure 3.6: The trigger and action procedure of the strategy **say-elaborating-relation**; also the action procedure of the strategy **say-property**.

```
(defstrategy find-next-to-say node
  (if (not *inhibit-recursion*)
      (let ((forms nil))
        (dolist (basic-relation *basic-relations*)
          (dolist (next (related-objects-to-say node basic-relation))
            (when (not (object-already-said? next))
              (push '(find-next-to-say ,node ,next ,basic-relation)
                    forms))))
        forms)
      nil))

(defun find-next-to-say (node next relation)
  (funcall *trajectory-hook* node next relation)
  (mark-relation-link-as-used node next relation)
  (save-current-node next)
  (dispatch next))
```

Figure 3.7: The trigger and the action procedure of the strategy **find-next-to-say**.

a relation between the current node and a *set* of related objects, it searches the relation hierarchy for the lowest common denominator of the relations to the related objects. So, for example, two daughters can be collectively referred to as "daughters," but a daughter and a son must be collectively referred to as "children."

The relation saying strategies are triggered only when the link from an object to another object is not marked as *already-used?*. This is independent of whether the other object is marked as *already-said?*. Thus is it important to maintain coherence by avoiding wandering off the topic along a chain of recursive elaborations. This is a danger because **say-object** is called by the say-relation strategies on the related object, and **say-object** dispatches on its argument after naming it. To eliminate this problem, the elaborating saying strategies bind a global variable, `*inhibit-recursion*`, which disallows applications that would take the text off the topic by following a recursive relation. This guarantees that only finding strategies can make a new node the current node. On the other hand, nonrecursive relations will not "go anywhere" beyond the current node, so it is always safe to say them.

The last thing that happens with any saying strategy, before the node is dispatched on again, is that context information is saved. This information includes marking the node as *already-said?* and recording that it is the *last-said node*. The strategy is saved as the *last saying strategy*; this information is used for focusing (see section 4.3). If the strategy is a relation saying strategy, the relation expressed is also marked.

3.3.2 Finding Strategies

Finding strategies find a next node to talk about. The most important finding strategy is **find-next-to-say** (see figure 3.7). **find-next-to-say** finds candidates for the next current node based on their being related to the current node by one of the basic relations of the domain. **find-next-to-say** is not triggered if recursion is inhibited. When **find-next-to-say** is applied, the new current node is set; this node will remain the current node until there is another find. In the

```
(defmetastrategy choose
    (block outer
      ;;collect the candidates in an indifference class
      (dolist (indifference-class *strategy-preference*)
        (let ((applications nil))
          (dolist (candidate candidates)
            (let ((strategy (first candidate)))
              (when (member strategy indifference-class)
                (push candidate applications))))
          (when applications
            ;; choose a strategy from the indifference class at random
            ;; filter out all applications not belonging to that strategy
            (setf applications (filter-one-strategy applications))
            ;; check to see if can order applications within the chosen
            ;; strategy
            (return-from outer
              (case (first (first applications))
                ((say-elaborating-relation
                 (argument-preference applications
                                      *elaborating-relation-preferences*))
                (find-next-to-say
                 (argument-preference applications
                                      *find-next-to-say-preferences*))
                (otherwise `((choose ,(random-elt applications)))))))))))

(defun choose (application)
  (apply-strategy application))
```

Figure 3.8: The trigger and action procedure of the **choose** metastrategy.

house domain, the basic relation is next-to (and its specializations), which represents the physical proximity of objects; for families it is the relations such as has-daughter. (**find-next-to-say** calls the *trajectory hook*, which is code to save limited history information in some domains, the exact nature of which varies with the domain. This mechanism will be discussed further in section 5.2.1.) `*trajectory-hook*` and `*basic-relations*` are all that need to be changed to make **find-next-to-say** work in any domain.

The other finding strategy is **find-salient-object**. This strategy is triggered when there is a node next-to the current node that has certain properties that make it *salient.* Something that is salient is noticeable or conspicuous in its context; a salient feature of a domain is a candidate for organizing a text or part of a text about the domain. Several researchers in generation have investigated the role of salience, notably Conklin (1983) and Pattabhiraman & Cercone (1990). However, salience is a hard thing to identify. In Salix, "salience" has a very specific sense that is grounded in the knowledge base. This sense is intended to capture that a salient object has many features (is involved in many elaborating relations) and is related to many other objects by the basic relation of the domain, against the background of which this object is salient. **find-salient-object** finds objects that are connected by the basic relation to a number of other objects, and involved in a number of relations, where these numbers are over some threshold. Clearly the value of the thresholds is arbitrary; salience is a difficult concept to operationalize.

3.3.3 Metastrategies

Metastrategies are triggered when there have been multiple matches.[3] They are responsible for resolving conflicts among competing applicable strategies. (Some of these competing strategies may be themselves metastrategies.)

As with **defstrategy**, the macro **defmetastrategy** puts triggers in the `*strategy-table*` (see figure 3.4). It includes in the trigger a test that the dispatchee is a conflict set. If the test is met, the trigger is applied to the old dispatchee (the one that resulted in multiple matches) and the candidates from the multiple match.

A dispatchee that has resulted from a multiple match will look like this:

```
(multiple-matches <old-dispatchee> <list of multiply matching candidates>)
```

The <old-dispatchee> may be a node or may itself have this form.

Two approaches to conflict resolution have been implemented in Salix. The *choose* metastrategy chooses one application from the set of candidates by finding one that is most appropriate as determined by a partial order of preference on all the strategies that may apply in this domain. For instance, it is generally appropriate to *say* things about the current node before trying to *find* the next node to talk about, so saying strategies are often ordered before finding ones. This metastrategy will be described in section 3.3.3. *Combining* metastrategies, on the other hand, are triggered when some subset of the available strategies may be felicitously combined. **say-left-right-and-center** and **say-multiple-sweep** are spatial combining metastrategies that are used in the house domain. No combining metastrategies are used in the family domain.

The dispatch cycle repeats until just one strategy is selected. This strategy then either finds the next thing to describe, says something about the current node, or, if it is a metastrategy, it may do some combination of the two. As the last step of their execution, saying strategies dispatch

[3] Davis (1980) first introduced the notion of metastrategies.

```
(defnull-strategy noop-on-inhibit-recursion
  (if *inhibit-recursion*
      '((noop-on-inhibit-recursion))
      nil))

(defun noop-on-inhibit-recursion ()
  ;; just return control to the metastrategy, return value irrelevant
  nil)

(defnull-strategy jump
  (if (null *inhibit-recursion*)
      (let ((jump-to (completeness-criterion-fulfiller)))
        (if jump-to
            '((jump ,jump-to))
            nil))
      nil))

(defun jump (to-node)
  (realize-oops)
  (save-relation (get-thing unknown))
  (save-saying-strategy 'jump)
  (setf *current-node* to-node)
  (dispatch to-node))

(defnull-strategy say-all-done
  (if (and (null *inhibit-recursion*)
           (null (completeness-criterion-fulfiller)))
      '((say-all-done))
      nil))

(defun say-all-done ()
  (realize-conclusion))
```

Figure 3.9: Salix's null strategies.

on the current node (so something else can be said about it or a next node can be found), whereas finding strategies dispatch on the new node they have found. A metastrategy may dispatch on several nodes, in an order determined by the metastrategy.

The Choose Metastrategy

There is a single **choose** metastrategy (see figure 3.8). **choose** matches any time there is a multiple match. This metastrategy uses a set of global variables to impose an ordering on the candidate applications, and returns the single application that it has selected, prefaced by the keyword choose.

strategy-preference is a partial ordering on all the strategies in the domain, including the metastrategies. In general, if there is more than one application of a strategy, one is selected

at random. There are exceptions requiring a more complex scheme: the elaborating strategy **say-elaborating-relation**, which matches on elaborating relations and **find-next-to-say**, which matches on basic relations. Since each of these strategies matches on a set of domain relations, a further ordering is imposed on them; for example a next-to-same-direction may be preferable to a next-to-opposite-direction.

The global variables `*strategy-preference*`, `*find-next-to-say-preference*`, and `*elaborating-relation-preferences*` are all that need to be changed for **choose** to work in one domain or another. Indeed, one can produce very different sorts of texts in a single domain just by reordering these preferences or removing some choices altogether.

3.3.4 Null Strategies

Null strategies are triggered when there have been no matches. Null strategies are responsible for finding something to do when no base strategy applies. As with **defstrategy**, the macro **defnull-strategy** puts triggers in the `*strategy-table*` (see figure 3.4). Included in the trigger is a test that the dispatchee is a no-matches situation. If the test is met, the trigger is applied to the old dispatchee (the one that resulted in no matches).

Salix has three null strategies, corresponding to the three possible causes for match failure (figure 3.9). First, when recursion is inhibited (section 3.3.1), most strategies will not match. In this case, the null strategy **noop-on-inhibit-recursion** simply returns control to the strategy that inhibited recursion.

Alternatively, there may genuinely be nothing left to say about the dispatchee and nothing not already said connected to it by a basic relation. As described in section 3.4, each domain has a *completeness criterion*, which specifies what must be included for a text in the domain to be complete. The null strategy **jump** uses the completeness criterion to find an object to jump to. It first produces a string such as "wait I forgot" and then dispatches on the jumped-to object.

If the completeness criterion has been satisfied, so that there is nothing more to talk about, the null strategy **say-all-done** applies. It produces a final increment of output such as "that's it."

3.4 Context

This section summarizes the *context* which comprises what Salix keeps track of during the construction of the text. Two sorts of context are maintained: annotations in the knowledge base of what has been said; and a record of what *just* happened.

Knowledge base annotations include:

1. For each node in the KB, whether it has been *already-said?*. An *already-said?* node cannot be explicitly introduced into the text again.

2. For each node in the KB, which of the relations it is involved in with other nodes have been *already-used?*. An *already-used?* relation link between two nodes cannot be used again. Relation links can become *already-used?* either by being explicitly expressed, or by being inferred (see section 3.5).

The immediate context includes:

1. The *dispatchee* (during a dispatch).
2. The *current node*, which is a single node in the knowledge base. The current node is the node which the current increment of text is about; it is the most recent node found by a finding strategy.
3. The *previous node*, which is the previous current node.
4. The *last-said node*, which is the last node expressed by **say-object**, that is, the last node that has been realized in the surface structure. It may or may not be the same as the previous node.
5. The last *saying strategy* that has been applied.
6. The *current relation*, which is the relation used to reach the current node. It may be unknown.
7. The *completeness criterion*, which is the item of domain dependent information that states what must be included for a text to be complete. The completeness criterion has a *fulfiller*, a list of nodes that must be included in the text. The completeness criterion is only checked at a *no-matches* impasse; this means that a text may continue longer than the criterion specifies.
8. The *trajectory*. In the house domain, the trajectory keeps track of what spatial direction the text moved in to get from the objects already described to the current one, and is used to calculate which of the available next nodes lie in the same direction, an orthogonal direction, or the opposite direction. (See section 5.2.1.) The trajectory need not be literally spatial: in other domains, the trajectory may be temporal, causal, or, as with a family, be up, down, or within generations.

The current node, previous node, and last-said node can best be distinguished by example.

```
(1) there's Penni
(2) her parents are Kitsy and John
(3) and then there's Barbara
```

In (1) the current node, previous node, and last-said node are all Penni. At the end of (2), Penni is still the current node and previous node, but John is the last-said node. At the end of (3), Barbara is the current node and last-said node, and Penni is the previous node.

3.5 Inferred Relations

When we say something, we expect that the hearer, in addition to understanding what has been stated explicitly, will *infer* additional information that follows from it. If we explicitly express what can have already been inferred, this is rude, and sometimes confusing, and almost certainly not very good text. So we want to dynamically keep track of what we are reasonably sure that the hearer can infer; this is a simple version of *hearer modeling*. (See (Kass & Finin 1988) for a survey of hearer, or user, modeling.)

Salix incorporates a rule system which models the inference process. The rest of this section describes the sorts of inferences that need to be made in the domains for which Salix is implemented, and gives some specific instances of rules.

Salix is very conservative about what it expects the hearer to have inferred. All the family members in the knowledge base have either the property of female or the property of male. Normally properties are available to be expressed by the strategy **say-property**, but this would result in unfortunate texts like

```
then there's John
who is male
```

We can depend on cultural knowledge of how proper names are gendered, and infer, once we've mentioned John, that he is male, and that our hearer will also be able to infer this. Thus, when Salix applies **say-object** to a person, it marks the has-property relation that involves either female or male as *already-used?*, thus suppressing the person's gender from being mentioned.

Another example of inferable information is evident in the following text:

```
then there's Penni
her parents are Kitsy and John
and then there's Barbara
who is Penni's sister
Barbara's parents are Kitsy and John
```

Once we know who Penni's parents are and that her sister is Barbara, we know who Barbara's parents are. A similar example can be drawn from the house domain. contains is an elaborating relation; it is used by **say-elaborating-relation** to elaborate on a room that it contains a set of furnishings. However, if we have already mentioned these furnishings, we probably do not want to subsequently say that the room contains them. This may not be true for all instances of contains, but in the house domain, it has proved appropriate that when an object that is contained-by another is mentioned, the link is marked as *already-used?*.

Every time Salix marks a relation link as *already-used?* it runs a procedure to *update* the relation links. The update procedure is a inference mechanism with a set of rules. The rules have a *lefthand side* and a *righthand side*. If the lefthand side matches, then the relation links described by the righthand side are marked as *already-used?*. The lefthand side of the rule has three parts: *said clauses*, *true clauses*, and *said objects*. Both types of clauses involve a special variable `?self`, which is bound to the current node. A true clause matches a specified relation link between `?self` and another object (or property). A said clause is the same except that the relation link has to be marked as *already-said?*. A said object is a specified object marked as *already-said?*. So, in general, a lefthand side matches if a certain set of relations the current node participates in have been said, another set of relations exist in the domain, and a set of objects have been explicitly expressed. If the lefthand side matches, relation links that match the righthand side are marked as *already-said?*.

Some examples make the mechanism clearer. The following rule says that if the current node is *already-said?* and it has-property male, that the relation link of having the property male is marked as *already-used?*. This rule suppresses output such as "John is male."

```
(defrule () ((?self has-property male)) (?self)
   ((?self has-property male)))
```

The next rule says that if another node has been said to be the father of the current node, and the current node is male, then it can be inferred that the current node is the son of the other node.

```
(defrule ((?self has-father ?x)) ((?self has-property male)) ()
  ((?x has-son ?self)))
```

Similarly, this rule infers a daughter relationship.

```
(defrule ((?self has-father ?x)) ((?self has-property female)) ()
  ((?x has-daughter ?self)))
```

These rules suppress output such as "Penni's father is John and John's daughter is Penni."

This next rule is slightly more complicated: it says that if the current node has one node as a daughter and another as a wife, then we can infer that the wife has the same daughter. This would suppress "John's daughter is Penni and his wife is Kitsy and Kitsy's daughter is Penni."

```
(defrule ((?self has-daughter ?x) (?self has-wife ?y)) () ()
  ((?y has-daughter ?x)))
```

When the inference mechanism is run, it iterates as many times as necessary to insure that all valid inferences are propagated.

3.6 Domain Independence

Eight global variables need to be set to reflect a particular domain. Most of these have been described in the preceding sections. `*trajectory-hook*` is used in some domains by **find-next-to-say** to save some limited history. The variables `*basic-relations*`, `*elaborating-relations*`, and `*nonrecursive-relations*` serve to identify the nature of the relations in the domain. The variables `*strategy-preference*`, `*find-next-to-say-preference*`, and `*elaborating-relation-preference*` are all used in **choose** to impose a preference ordering on candidate strategies. The `*completeness-criterion*`, discussed in section 3.4, encodes a criterion for when a text in a domain may be considered complete. As has been mentioned, two or more domains may be encoded in the same KB. The basic code for Salix is exactly the same for any domain except for the global variables in which domain dependence is encapsulated. Thus, Salix can dynamically switch from generating text in one domain to generating text in another, just by the flip of eight variables.

3.7 Related AI Architectures

Most implemented AI systems for engaging in activity, that is, for *doing something*, fall into the *planning* paradigm. An AI planner works with a description of the world which comprises a set of propositions. The world can be in one of a number of states, depending on the truth values of its propositions. The planner has at its disposal a set of operators which can change the truth values of various propositions. The planner is given a goal, say, "block A is on block B" and its job is to construct a list of operator actions which will get it from its current state to the goal state. The bulk of the work the planner does is in ensuring that the order on this list of operator actions allows the goal state to be reached.

Another, less emphasized, line of research in AI has been *problem solving*. This research started with Logic Theorist (Newell *et al.* 1958) and GPS (Newell & Simon 1963). The classic example

of a problem solver is NASL (McDermott 1978). Rather than concentrating on getting from state A to state B, when a problem solver has a task to be accomplished, it maintains a list of pending subtasks, and when it is able to accomplish one or more of them it does so.

I will review in this section two important problem solving architectures—NASL and Soar (Laird *et al.* 1987). While these two projects have different emphases from each other and from Salix, the architectures are similar to the present one in important respects, particularly in the handling of conflicts in deciding what to do next.

I also review Pengi (Agre & Chapman 1987), another AI architecture; the approach taken in Pengi to engaging in activity bears some similarity to the approach taken in Salix to building text.

As observed in the first chapter, many text planners have used some sort of hierarchical planner. Hierarchical planning (Sacerdoti 1977) incorporates the idea that goals can be decomposed into subgoals, and a plan for achieving the original goal can be constructed. This recursive goal decomposition results in a tree. Reviews of systems that use either hierarchical plans or schemas can be found in section 3.8.

more importantly, Salix is not concerned with specifying start and goal states and building a plan to get from one to the other. Instead, like a problem solver, Salix starts with something to build text about and some strategies to build text with, and it works on the task of building a text until some criterion for completion is met.

3.7.1 NASL

McDermott (1978) points out that most construals of what a problem is—such as a state of affairs to be brought about or an object to be generated—are too narrow. Many problems are not so simply described; McDermott gives as examples "Wait here for five minutes" and "Think of a fallible Irishman." Rather than trying to force these problems into state change terminology, it works better to refer to them all as actions. In McDermott's terminology, a *task* is a describable action to which the system is committed. A *problem* is an action which cannot be carried out until it is decomposed into its (primitive) subtasks (and scheduling relationships between them).

McDermott takes the view that a problem solver acts on the real world, not some data structure; so rather than doing a search for the best plan, actions are selected and executed. If the world behaves in an unexpected way or an action does not work, then it does no good to have a fully worked out plan for what to do. McDermott believes that by the time you find that your plan for action will not work, it is usually too late to do something about it: "protection violation is an execution time phenomenon."

Because search will not be performed, there must be only one way to accomplish each task. This requires that it be possible to determine the (or a) right *schema* (way to solve a task) in each task instance. Rather than try to annotate each schema with enough detail to distinguish it from any possible contender (even ones that have not yet been conceived of), schema selection for a task has two phases: retrieval of potential schemas, and choice between them. Choice is accomplished by retrieving choice rules (in the same way as schemas), and the rules choose between the schemas or select a new one. This is similar to Salix's multiple match and dispatch deliberation.

Subtask coordination is the tricky part of problem solving. There is no one way to guarantee it will work out right. Therefore, there are rules which notice which other tasks are also active, since this may have an effect on how the way to accomplish the current task is chosen.

In NASL, a task may be executed or a problem decomposed as soon as scheduling constraints

allow. If a problem cannot be decomposed, then it must be rephrased in the hope that a different statement of it may allow different, and doable, task decomposition.

The control structure is deterministic. If something goes wrong, schemas need to be found to fix it; there is no backtracking. The basic choice cycle of the rule interpreter looks like this:

1. choose task (task is chosen randomly from all that match)
2. if task is primitive, execute it else reduce it
3. repeat till done

If there is no change during the choice cycle (no task can be picked), there is a mechanism to nudge NASL to try something else. This is analogous to the use of null strategies in Salix.

Most current "reactive" planning architectures are similar in operation to NASL, for example Firby's Reactive Action Packets (1987), Georgeff & Lansky's Procedural Reasoning System (1987), and Lin *et al.*'s Task Control Architecture (1989).

Salix's architecture bears many similarities to NASL's, such as the simple control cycle, immediate execution of tasks rather than search for a plan, and mechanism to get the system out of a corner. A difference is that NASL has a notion of task decomposition whereas Salix does not. Another difference is that NASL has a more elaborate mechanism for resolving conflicts, while Salix has the single mechanism of dispatching over again an arbitrary number of times until one choice remains. This arbitration mechanism is more like that of Soar.

3.7.2 Soar

Soar (Laird *et al.* 1987; Newell 1990) has become one of the current standard AI architectures, in part because a large number of people have been working on it for nearly a decade, but also in part because its researchers have made it the basis of a psychological theory of all cognition.

The key ideas of Soar are uniform representation and uniform processing: Soar has the same components and uses the same mechanisms independent of domain and task.

Soar uses a production system to uniformly represent all long-term knowledge—search control, declarative knowledge, episodic knowledge, "everything." In a classical production system (Waterman & Hayes-Roth 1978; Newell 1973; Forgy 1981), or rule based system, when more than one rule matches a situation, and forms a conflict set, the conflict resolution mechanism simply selects one rule from the conflict set according to a hard-wired criterion. Soar, however, differs in two respects:

1. Soar productions can only add to working memory—the set of data that is currently available (deletion is a side effect of the problem solving process, when contexts are popped from the context stack, as described below).
2. Soar does not do conflict resolution but instead decides what to do by a more complex process.

For Soar, a task is an attempt to attain a goal which is formulated in terms of a desired state in a *problem space*. Given a start state, the goal state can be found by heuristic search through the problem space. Even one-step operators are represented in this scheme, in order to have uniform representation. An important part of Soar, then, is selecting the right problem space, and the start state and operators within it.

In order to achieve a goal, Soar uses a *decision cycle*, which has two phases, the *elaboration phase*, which gathers all available information about the current situation, including *preferences* for various actions; and the *decision procedure*, which uses the preferences to select the best thing to do next. The decisions can be about anything—including choosing problem spaces and operators.

For each goal, a *context* is created, consisting of the goal, a problem space, a start state, and an operator for reaching the next state. The decision procedure refers to the *context stack*, that is, earlier decisions about goals, problem spaces, states and operators. The context stack can be considered a goal-subgoal hierarchy.

Soar brings all available knowledge to bear on making the decision of what to do next to attain the current goal. If there is not enough knowledge to make the decision, the decision procedure comes to an *impasse*. Impasses may result from a multiple match of productions, no match, or a conflict between preferences.

A new subgoal is set up to deal with an impasse. Such goals may be to figure out which available operator is the best match, how to apply a selected operator, how to apply the result of a selected operator, or how to satisfy the preconditions of the selected operator. The idea here is that *anything* can be the object of deliberation, and any source of knowledge may be involved.[4]

Soar's creation of a new subgoal when an impasse is reached is just like Salix's process of redispatching on the dispatchee. In each case, the entire context, including the results of the previous dispatch, is the new dispatchee or subgoal. Soar differ from Salix in that it maintains a stack of previous contexts.

3.7.3 Pengi

Pengi (Agre & Chapman 1987) invites comparison with Salix not because of architectural similarities but because both systems take a "Simon's ant" (Simon 1970) approach to modeling the behavior of interest. Simon's ant is a metaphor for how complex activity can arise from the interaction of simple actions and a complex environment. If one watches an ant run across a beach, the path it chooses might seem complex, and one could suppose that this complexity is the result of complexity in the ant's deciding what to do. Instead, an equally plausible explanation of the ant's activity is that it makes simple choices from a small repertoire of possible actions in response to the current environment. Where the environment is complex (as is a beach for an ant), the series of choices, each of which is simple and immediate from the ant's point of view, may look as though they compose some larger more complex structure from an observer's point of view.

Pengi plays a fast-paced video game, Pengo, by constantly redeciding what to do, based on what is happening in the game right now and a small set of actions the player can take. Pengi tries to avoid having the penguin icon get killed by ice blocks that are kicked by killer bees while it collects magic blocks in order to win the game. For example, Pengi makes decisions concerning which way to run or whether to kick a block at a threatening bee, or to search for a magic block. At any point in its execution, Pengi is operating on the current environment with only a set of goals and a set of skills.

Patterns of interaction between Pengi and the game are called *routines*, and arise from the agent's application of rules to situations. Though these routines are discernible to an observer,

[4]This process has been referred to as "universal subgoaling," conveying the idea that whatever impasse Soar is facing, it uses the same technique of creating a new subgoal. I find this terminology to the point when comparing computational architectures; however, it has fallen into disfavor in the Soar community which has been emphasizing Soar's appropriateness as a cognitive model (Allen Newell, personal communication).

they are not given an explicit representation. The agent has no preconceived notion of the effects of its actions, and at any time the actions may change to cope with a change in the world, resulting in a different routine.

Pengi's world is represented not by a comprehensive set of propositions, but by *deictic entities* and *aspects* which are the relevant properties of the immediate situation. The aspects are *indexical* because they depend on the circumstances (they comprise the part of the world which is relevant to the agent); entities and aspects are *functional* because they are relative to the agent's goals. The set of entities to which Pengi pays attention at any one time is limited: Pengi has a set of visual markers which it can put on things in the game which it wants to keep track of.

The architecture employed by Pengi consists of a central system for cognition, and peripheral systems for perception and effector control (the peripheral systems perceive and interact with the game screen). Action arbitration is used to select from the set of locally plausible actions at any time, and there are levels of arbitration which decide among the conflicting suggestions.

With this brief description of Pengi, we can draw comparisons between Pengi and Salix. Pengi's task is to play a video game, Salix's is to generate text. The environment that Pengi operates in is the game of Pengo, while Salix's environment is the knowledge base from which it is generating. In each case, everything the system needs to know about the domain is encoded; the encoding is in terms of the important aspects of the domain. For Pengi, these are what can be seen on the game screen—what the entities look like, where they are, and which direction they are moving. For Salix, the important aspects of the domain are what objects and properties there are and what relations hold between them. Pengi pays attention to the functional aspects of entities, while Salix is sensitive to whether relations are basic or elaborating; each mechanism serves to highlight the most crucial things to the system.

Salix uses a local context of what has just been said and what is available to say (what is related to what has just been talked about) to constrain choices of what to say next. Pengi uses a limited set of visual markers to constrain the entities that it keeps track of and thus to constrain the set of actions it will consider taking.

The complex activity in which Pengi is engaged is competent playing of the video game. The interaction between Pengi's simple strategies and the complex environment of the game results in emergent routines. To an observer, a particular routine serves to cope with a particular game situation, but the routine has no status as far as Pengi's actions are concerned. Similarly, in Salix, the complex "activity" is the construction of coherent text. The interaction between Salix's strategies and the KB representation of the domain produces what to an observer can be global text structure; this global structure, however, is not represented in Salix and does not figure into the choices Salix makes in the process of generation.

As indicated by the scare quotes around Salix's "activity," there is a point at which the analogy between Salix and Pengi breaks down. Salix is not really engaging in an activity or interacting with a world, which might be fast paced and unpredictable. Salix would be participating in activity of this sort if it engaged in conversation and making its decisions in a context that included other conversational participants and the things they said. As it is, Salix operates in an environment where it can move at its own pace and where any changes effected are those brought about by Salix itself. However, in Salix, as in Pengi, complex results, be they global text structure or routines of activity, arise from the interaction of simple strategies with a complex environment, without the observable structure being represented by the system itself.

3.8 Text Planning Architectures

In this section I briefly review important and representative generation systems. The reviews here emphasize the strategic component of these systems; more particularly tactical systems are discussed in chapter 7.

3.8.1 KAMP

Appelt (1985) does not distinguish language generation from any other action, and uses the same planner, KAMP, that would plan physical actions to plan output sentences, including planning what is included in particular noun phrases. The justification is that everything an agent does is goal-based and language is no different. Appelt is particularly interested in accounting for how noun phrases refer.

Appelt did his work within a tradition that considers language to be an effector that operates on the world like any other (Cohen & Perrault 1979). In such a goal-based theory, each utterance typically satisfies multiple goals, such as communicating information, communicating affect, and obeying social conventions such as being polite. A speaker plans to satisfy all these goals using a hierarchical planner with nonlinear plans (Sacerdoti 1977), using information modeling the state of the world, interlocuters' knowledge and beliefs, and the agent's own mental state and goals.

A *referring expression* is one means by which a concept is activated, that is, brought to the attention of both the speaker and the hearer. A surface speech act (a text) has two interacting components, for *intention communication* and *linguistic realization.* As the plan for the text is expanded, the intention communication component constrains and is constrained by the syntactic structure built by the linguistic realization component. The higher-level illocutionary acts such as **inform** and **request** can map onto many different surface speech acts (e.g., "Get the door" and "Would you close the door, please?") and more than one illocutionary act can be realized in a surface speech act (e.g., both politeness and request in the second example). Concept activation occurs during the building of a surface speech act; a basic descriptor is selected, and descriptors are added to uniquely identify the concept.

For Appelt, the ability to add large amounts of information locally to a noun phrase exemplifies the planning technique of *action subsumption* to achieve multiple goals. Generally speaking, a planner's *critics* can detect, when the planner is satisfying the goal of activating a concept, that it has another goal of mentioning something else to do with that concept, and the planner can attempt to satisfy the second goal while satisfying the first.

While Appelt is insistent that the planning view of generation obviates the strategic/tactical distinction, he relies on the hierarchical nature of his planner to separate the planning of domain-level actions and low-level linguistic actions, which seems like a similar distinction in a different guise. On the whole, Appelt's work is constrained by his determination to apply a particular planning formalism to natural language generation, rather than looking for how text can best be generated.

3.8.2 TEXT

McKeown, with her TEXT system (1985), accomplished some of the earliest work in planning the generation of text. Working from a knowledge base of naval ships and weapons, TEXT answers questions like "What is a ship?" with paragraph-long answers. In this case, the paragraph begins

with the sentence, "A ship is a water-going vehicle that travels on the surface." and continues with several simple sentences describing database attributes such as displacement, fuel capacity, and range.

Built to be capable of achieving three communicative goals—*definition*, *description*, and *comparison*—TEXT is divided into two components, the *strategic* and the *tactical*, which in McKeown's terminology decide, respectively, "what to say" and "how to say it."

McKeown notes that while selecting information is an important task of the planner, ordering of the information also has a great effect on meaning and clarity. The relations that are available to TEXT are database relations such as those that attribute properties to objects. McKeown believes, however, that textual structure should not be constrained by the structure of the knowledge base; instead, there are certain text patterns that may be used over and over again, and several of these may be used to present the same information in different ways for different purposes. The tasks of the strategic component of TEXT are: to construct a *relevant knowledge pool* that is a subset of the knowledge base; to select a strategy which encodes *rhetorical techniques*; and to implement an *immediate focus* method, based on Sidner's model of focus (Sidner 1979).

McKeown's analysis of paragraph-length texts written by people indicated to her that rhetorical techniques, like *specification*, *evidence*, and *analogy*, used to achieve a particular communicative goal (such as *definition* or *comparison*), tended to cluster in certain combinations. She reasoned that an effective method for having a generator produce coherent text would be to aggregate these techniques into *schemata*: *identification*, *constituency*, *attributive*, and *contrastive*. Schemata contain many alternatives and are, according to McKeown, relatively unrestrictive because of this. They are implemented as augmented transition networks, in which taking an arc involves selecting a proposition for the schema. The ATN is modified so as not to allow backtracking, since this would interfere with focus constraints.

Selecting a strategy for answering a question becomes a matter of selecting an appropriate schema and instantiating it by traversing it. A traversal begins at the start state, and arcs are selected depending on whether associated predicates match propositions in the relevant knowledge pool. A schema is selected by virtue of its association with a discourse goal (in TEXT's implementation, the type of question to be answered) and on the basis of available information; for example, *constituency* can be used for definition when there is a lot of information about the object; *identification* is used when there is not.

Schemata interact with the focusing mechanism, which keeps one or more objects in focus and applies *immediate focus* constraints to order propositions, by considering which proposition should follow the current one depending on the objects in focus that they have in common.

Schemata work well in constrained generation tasks. However, they are not very generalizable: the methods for organizing the underlying representation are to a large extent precompiled and prepackaged. In a very real sense, there is only one choice that can be made, selecting a schema, and after that there is no room for changing course or going on a tangent or dropping the subject altogether. It is likely that for general tasks of description, and certainly for more varied tasks, sets of schemata simply will not have sufficient versatility to do the job.

3.8.3 PAULINE

Hovy (1988) argues that the tasks of a program that plans for generation fall into two categories: *prescriptive* and *restrictive*. He proposes that a planner contain two types of planning to accommodate this situation, and he calls this combination *limited-commitment planning*. His solution also

requires that planning and realization be interleaved and able to affect each other.

Hovy categorizes previous planning systems into two sorts. In the *integrated approach*, the *planner-realizer* treats all constraints, be they syntactic or pragmatic, the same way; they are all incorporated in a traditional hierarchical planner. The grammar is distributed, which is unattractive, and all the types of planning tasks must be represented homogeneously, which would be difficult.

The *separated approach*, as in TEXT (McKeown 1985), plans an entire text completely, and then hands it off to the realizer. This kind of planner cannot take advantage of serendipitous syntactic contexts; it is likely to have only an imperfect notion of syntax. Hovy claims that if the realizer could fulfill zero or several planning instructions with one construction, this would necessitate replanning, which is impossible in this model.

Thus, Hovy argues for two separable planning tasks. *Prescriptive* planning determines what should be done for a section of text, but not, under Hovy's assumption, down to syntactic choices, which tend to be simple and local. When the realizer needs to make a choice, it calls on the planner, accepts the choice, and continues to the next choice point. Hovy calls this *in-line* or *restrictive* planning, and conceives of it as different from prescriptive planning.

The author builds an odd straw man in support of his argument that restrictive, but not prescriptive, planning can achieve pragmatic goals. He claims that a plan constructed beforehand, which he calls a top-down plan, could not possibly prescribe ways to express goals such as impressing the hearer or being friendly, since these goals usually find expression in individual word or phrase choice. He argues that a top-down plan for the goal of impressing could only accomplish the sentence "I want to impress you," which is obviously counterproductive. Another unusual argument he makes is that top-down planning is not good for pragmatic goals: goals in a top-down plan may be achieved and flushed, but pragmatic goals usually want to obtain throughout the realization of the text.

In Hovy's design, implemented as part of PAULINE (see section 7.2.3), a limited-commitment approach is taken—most planning is deferred until the realizer needs it, so that the planner and the realizer communicate at the realizer's choice points. This allows the planner to take full opportunity of all the choices available, including those contingent on the local syntactic context.

Hovy is correct in saying that much of the interesting generation work is accomplished by low-level syntactic and lexical decisions, and, of course, he is right that these must be controlled to achieve the desired effects. But his arguments to support his answer to this problem are weak in two ways. First, his claims as to the constraints on and capabilities of a top-down planner are unsubstantiated; he also adds unnecessary constraints that other generation systems do not find problematic, particularly, the lack of explicit syntactic knowledge available to the planner. Second, and more important, it is quite unclear why what Hovy calls in-line planning needs to be considered planning at all, and as such fit into the workings of the planner. Instead, the single-choice plans—selected on the basis of goals—seem exactly like Mumble's realization class choices—selected by characteristics (see section 7.2.2), or Nigel's system features—selected through inquiries (see section 7.2.1). Instead of solving an existing problem, Hovy has managed simply to relabel as a type of text planning a process that is generally considered to be surface structure realization.

Chapter 4

Application I: Families

One domain on which Salix has been tested is that of families. This chapter describes that application, including how the domain is represented, what the basic and elaborating relations are, how the strategies exploit them, and what sorts of text are realized. Salix uses only domain independent strategies to build family texts. This chapter also examines *focus* as it is handled in the family domain in particular and by Salix in general.

4.1 The Family Domain Implementation

To represent the family domain, the knowledge base requires **objects** of one type, *family-member*, the **propertys** female and male, and the **relations** has-mother, has-father, has-daughter, has-son, has-wife, has-husband, has-sister, and has-brother.

These relations are the *basic relations* of the family domain, the ones exploited by **find-next-to-say**. When you are done talking about someone, it makes sense to next talk about someone who is closely related to them. These relations are *also* the *elaborating relations*: they can be exploited by the saying strategies. When you are talking about someone's position in the family, talking about their closest relatives is a way to add information about them.

All of these relations are *recursive* elaborating relations. Therefore, Salix will inhibit recursion when applying them: once one of these relation links has been followed to another object, following further links is disallowed, or the text would never return to the object being elaborated on.

Relations	Generalization
has-mother, has-father	has-parent
has-daughter, has-son	has-child
has-wife, has-husband	has-spouse
has-sister, has-brother	has-sibling

Figure 4.1: The relation generalization hierarchy.

All these relations are also *set relations*; that is, a family member can be elaborated on by talking about her parents, siblings, and children collectively, for example by using a single phrase

to say "her parents are Kitsy and John." Of course, a set can have a single member, as the relation has-husband typically does.

These relations belong to a simple *generalization hierarchy*, as shown in figure 4.1. This generalization hierarchy encodes the knowledge that, for example, a son and a daughter are both children; this knowledge allows the surface structure realization component to refer to a son and a daughter collectively as "children." The generalized relations are not explicitly represented by links.

The generalization hierarchy allows the collection of, for example, both daughters and sons under the has-child superrelation, by one application of **say-elaborating-relation**. In the general case, the realized relation is the lowest common denominator of all the relations collected by **say-elaborating-relation**. For example, in the family domain, if someone's children are both female, we can say "her daughters are Lara and Samara," but if one is female and one male, we have to say "her children are Christopher and Sarah."

The family domain requires just one metastrategy, **choose**, for building texts. This is the partial ordering on strategy applications that **choose** uses to resolve multiple matches:

```
(say-first-object)
(say-object)
(say-elaborating-relation)
(find-next-to-say)
(choose)
```

The lists indicate that these are *indifference* classes, that is, that any member of any type within the class has the same status as any other. In this case, the order is total, so the indifference classes are all singleton.

Typically, there will be many applications of **say-elaborating-relation** and **find-next-to-say** available at any match. Therefore, another partial ordering aids in selecting a single application. In the family domain, the partial ordering (also in terms of indifference classes) among the relations is as follows. This ordering approximates that in my data.

```
(has-wife has-husband has-daughter has-son)
(has-mother has-father has-sister has-brother)
```

The *completeness criterion* for a family text is that everyone in the family should be mentioned. The *fulfiller* for this criterion is a list of all the family members; this implicitly encodes that they are all related to each other. When the null strategy **jump** is applied, during construction of the text, an unsaid family member is picked off this list, thus ensuring that everyone in the family will get mentioned.

I have encoded in the KB my extended family, as described in section 2.2. All 51 family members are defined as objects of type *family-member*, and all have proper names. For example (the *t* indicates that a proper name is being supplied):

```
(defobject garafelia family-member "Garafelia" t)
```

All the relation links each family member is involved in are created, and each family member is given either the property female or the property male.

A representative part of the family KB is included in appendix C.

4.2 Example of Text in the Family Domain

In this section, we will walk through Salix's generation of a family text (see figure 4.2). The details of how text is realized by the strategy **say-object** will be discussed in section 4.3.

The text starts with a dispatch on Penni, which is made the current node. Several strategies match and return applications. **say-elaborating-relation** returns two applications: one with the relation has-parent and the objects Kitsy and John; and one with the relation has-sister and the object Barbara. **find-next-to-say** returns three applications: with has-father and John; with has-mother and Kitsy; and with has-sister and Barbara. **say-object** matches with Penni, and **say-first-object** matches with Penni. Since **choose** prefers **say-first-object** above all other strategies, it selects this last application.

say-first-object first realizes the introductory cue phrase "all right" (1), and realizes an increment that indicates that this is the first object in the text (2). Penni is marked as *already-said?* and **say-first-object** dispatches again on Penni.

This time the same applications of **say-elaborating-relation** and **find-next-to-say** are returned by **match** as before; both **say-first-object** and **say-object** are disabled because Penni is marked as *already-said?*. **choose** prefers the **say-elaborating-relation** applications. Since these two applications belong to the same indifference class, **choose** selects one randomly: **say-elaborating-relation** with the relation has-sister and the object Barbara (3). **say-elaborating-relation** calls **say-object** on Barbara. **say-elaborating-relation** inhibits recursion; when, after marking Barbara as *already-said?*, **say-object** dispatches on Barbara, no saying or finding strategies match because all involve recursive relations. Therefore, there is a no-match impasse, which the null strategy **noop-on-inhibit-recursion** matches. It returns control to the application of **say-elaborating-relation**, which again dispatches on Penni.

On the next dispatch, the available applications are: **say-elaborating-relation** with has-parent and John and Kitsy; **find-next-to-say** with has-father and John; and **find-next-to-say** with has-mother and Kitsy. The **say-elaborating-relation** application is chosen (4), and applied in a manner similar to that in the previous increment. Once again, Penni is dispatched on.

This time, there are no matching strategies because all the relation links from Penni have been realized. The completeness criterion has not been fulfilled, so the null strategy **jump** matches. This strategy randomly selects a family member who has not been marked as *already-said?* and dispatches on, in this case, Ann. Ann becomes the new current node.

On this dispatch, **say-object** and various **say-elaborating-relation** and **find-next-to-say** applications are returned by match, and **say-object** is selected and applied to Ann, resulting in increment (6). At the end of **say-object**, Ann is dispatched on again. Recursion has *not* been inhibited, so finding and saying strategies match, and the **say-elaborating-relation** resulting in increment (7) is selected.

This process continues in similar vein for the rest of the text, until increment (42). This increment has been produced by a **say-elaborating-relation** application on Helen, at the end of which Helen is dispatched on. No saying or finding strategies match, recursion is not inhibited, and the completeness criterion has been fulfilled, so the null strategy **say-all-done** matches. This strategy realizes the conclusion cue phrase in (43), and does not dispatch. The generation of the text is all done.

An execution trace of part of a text in the family domain appears in appendix D.

```
(describe-family penni)

        (1)  all right
        (2)  first there's Penni
        (3)  her sister is Barbara
        (4)  and her parents are Kitsy and John
        (5)  and then there's Ann
        (6)  her siblings are Billy Diana Martha and Becky
        (7)  Ann's parents are Margaret and Bill
        (8)  Ann's children are Rebecca and Nathan
        (9)  Ann's husband is Bennett
       (10)  then there's Andrew
       (11)  Andrew's siblings are Jeffrey and Alison
       (12)  Andrew's parents are Vi and Jeff
       (13)  then there's Kathy
       (14)  her parents are Diana and Paul
       (15)  Kathy's brother is David
       (16)  and then there's Ruth
       (17)  her daughters are Margaret and Alice
       (18)  Ruth's husband is Charles
       (19)  then there's Andrew
       (20)  his parents are Dorothy and Jim
       (21)  Andrew's siblings are Jonathan Benjamin and Martha
       (22)  then there's Pat
       (23)  his wife is Martha
       (24)  their daughters are Lara and Samara
       (25)  then there's Carol
       (26)  Carol's children are Sarah and Chris
       (27)  and her husband is Carleton
       (28)  and then there's Betty
       (29)  her parents are Ann and Eugene
       (30)  and her sister is Kitsy
       (31)  then there's Emily
       (32)  Emily's parents are Jane and David
       (33)  then there's Arthur
       (34)  Arthur's daughters are Eleanor and Elizabeth
       (35)  his brother is Charles
       (36)  their mother is Garafelia
       (37)  then there's George
       (38)  George's wife is Alice
       (39)  George's children are Dorothy and Jeff and Carleton and David
       (40)  and then there's Helen
       (41)  her parents are Ellen and Charles
       (42)  Helen's siblings are Eugene and Margaret and Alice
       (43)  that's all
```

Figure 4.2: A family text generated by Salix.

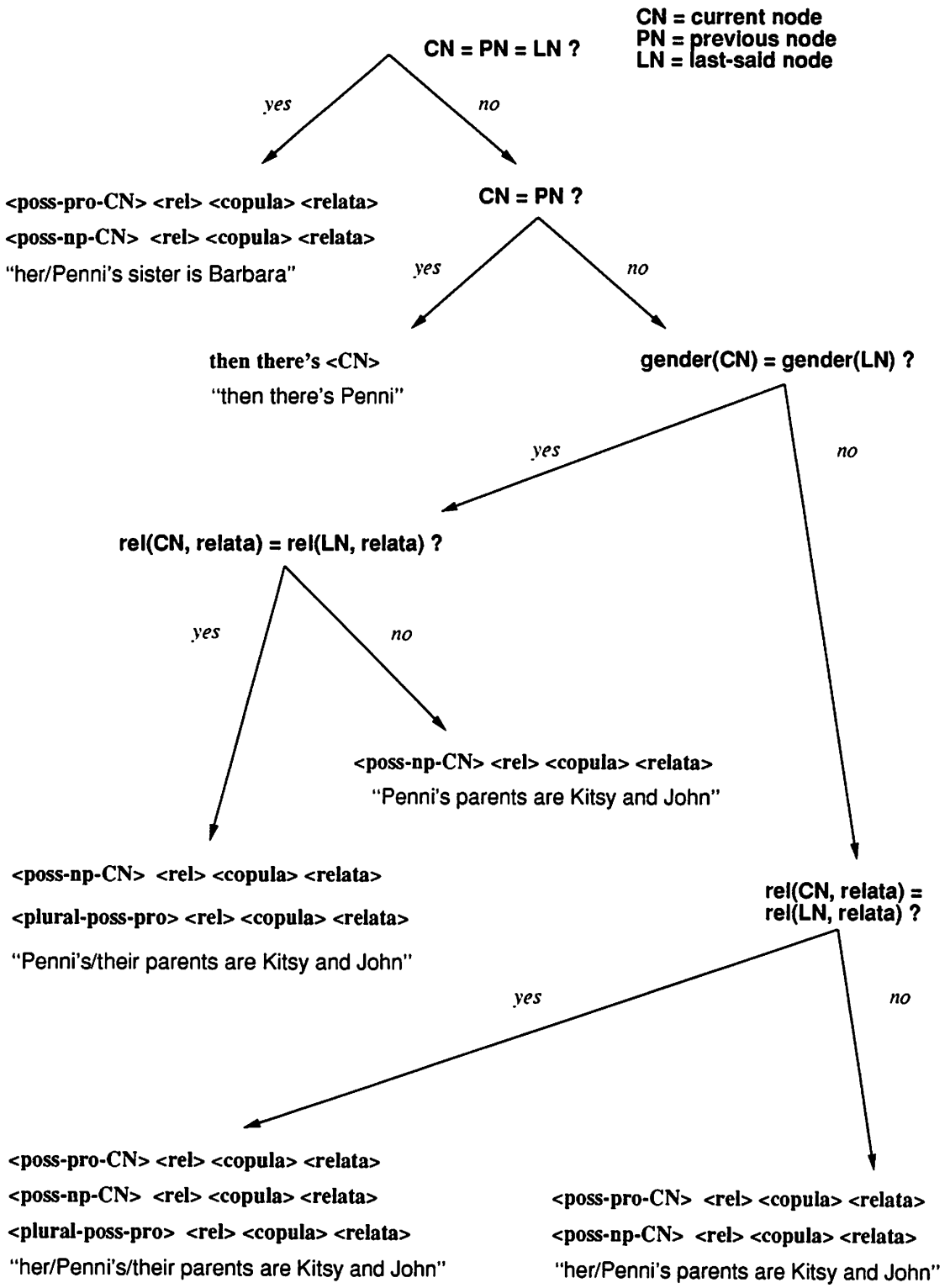

Figure 4.3: Decision tree for anaphora algorithm.

4.3 Focus

Focus is concerned with constraining what can be said next and about deciding when anaphora may and may not be used. Most of Salix's architecture is geared toward half this problem, that is, deciding what to say next. There is no special purpose focusing mechanism in Salix. In this section I discuss how Salix handles the particular issue of *anaphora*. Generally speaking, an *anaphor* is a word or phrase which is understood only in terms of its *coreferent*, which is another word or phrase in the text (the coreferent usually precedes the anaphor). Anaphora is particularly interesting in the family domain, where pronouns may be masculine or feminine, singular or plural.

In the domains in which Salix has been applied there is no need to look beyond the local context to decide what to do about anaphora. In particular, the information needed is contained in four variables: the *current node*; the *previous node*; the *last-said node*; and the *current relation*.[1] The algorithm described here is schematized in figure 4.3.

The current node is the node the current increment is about. Either the increment is realizing the current node explicitly with **say-object** or **say-first-object**, or the increment is an elaboration on the current node. For example, Penni is the current node in increments (1-4), and Ann is the current node in increments (5-9). The use of anaphora (either possessive pronouns or possessive noun phrases) in increments (6-9) makes explicit that these increments elaborate on the object mentioned in increment (5).

The previous node is the node the last increment was about, i.e., the current node of the last increment. In increments (6-9), the previous node is Ann, but in (5) the previous node is Penni.

The last-said node is whatever node was last expressed by **say-object**. At the beginning of increment (6), the last-said node is Ann; at the beginning of increment (7), it is Becky.

The current relation, the relation expressed in the current increment, is either one of the basic relations of the domain, or it is the relation unknown. In the family domain, the realization mode of all the basic relations is *say-object-noun+be*. So if the current relation, which is the relation link that was followed from the current node to the related objects, is one of the basic relations, the general form of the increment produced by **say-object** is

```
<current-node-possessive> <relation-noun> <copula> <related-objects>
```

When the **jump** null strategy is applied, the current relation is set to unknown: no relation link was followed to get to the current object, so the relationship between the previous node and the current node is unknown. In this case, the realization-mode is *say-object-there-insertion*, and the increment has the form

```
then there's <current-node>
```

Examples of text produced in this mode are increments (5) and (10). Such increments do not involve anaphora, but instead indicate, by both the lack of anaphora and the use of *there-insertion*, a focus shift. Because no anaphora is used to relate this increment to a previous one, the hearer can assume that the text has stopped talking about whatever object it was talking about before and is now talking about some other one.

[1] This set of four variables that Salix uses is similar to Epicure's (Dale 1988) *immediate context* used for *immediate pronominalization*. The immediate context is the part of the discourse model corresponding to to current and previous clauses of text.

When the current relation is one of the domain relations, then examination of the current, previous, and last-said nodes determines what anaphors are appropriate.

At increment (3), the current node, previous node, and last-said node are all Penni. The current relation is has-sister, and the related object is Barbara. To express that it is the current node, Penni, who bears this relation to Barbara, we have the following two options:

```
her sister is Barbara
Penni's sister is Barbara
```

Salix chooses one of these forms nondeterministically.

At increment (4), the current node and previous node are still both Penni, but the last-said node is Barbara. If we said in this case

```
her parents are Kitsy and John
```

this would imply, in the absence of prosodics, that Barbara's parents are Kitsy and John, and in turn imply that Penni's parents were not. This latter is certainly not the case, since this increment is an elaboration on Penni. So to ensure that the hearer knows that increment (4) is about Penni's parents, Salix can choose one of the following increments

```
Penni's parents are Kitsy and John
and her parents are Kitsy and John
```

In increment (4), Salix chooses the latter. In increment (18), Salix makes the former choice (in this case, the current node is Ruth).

Notice that there is not a conflict between whether the anaphor corefers with the last-said node or with the current node in the similar case in increments (34-35).

```
Arthur's daughters are Eleanor and Elizabeth
his brother is Charles
```

In this case, the last-said node and the current node have different genders, and there can be no misunderstanding.

There is a third option for realizing increment (4), and that is

```
their parents are Kitsy and John
```

In this case, the plural possessive pronoun is possible, because the last-said node bears the same relation to the related objects that is being expressed with respect to the current node in the current increment.

In realizing increment (4), the choice among

```
Penni's parents are Kitsy and John
and her parents are Kitsy and John
their parents are Kitsy and John
```

is made nondeterministically.

The focusing algorithm for anaphora described here will not work for deciding anaphors that require less local context. Where a larger context is required, the stack of McKeown (1985) and the trees of Hovy & McCoy (1989) may prove appropriate. Dale (1989), who uses hierarchical recipe plans, has addressed some of the issues of where anaphora are appropriate. As his work points out, anaphors are not only used to refer to a single definite entity in the text (such as the current node), but may refer to a collection of things, or something in a different form. As research in the comprehension of anaphora has shown (e.g., Webber 1983, Webber 1988, Anderson *et al.* 1989), the coreferent of an anaphor may be very difficult to pick out of a text.

Chapter 5

Application II: Houses

Another domain to which Salix has been applied is that of houses. This chapter describes that domain, including how the domain is represented, what the basic and elaborating relations are, how the strategies exploit them, and what sorts of text are realized. Salix can build house texts using only domain independent strategies; there are also some spatial metastrategies that Salix can use for spatial domains like houses. The spatial metastrategies incorporate deixis, which is discussed in section 5.2.1.

5.1 The House Domain Implementation

To represent the house domain, the knowledge base requires **objects** of types *room*, *path*, *non-path-room*, *furnishing*, *structure*, and *doorway*. Some objects have just one type: a window is a *structure*, and a piece of furniture or a large kitchen appliance is a *furnishing*. Other objects in the house domain have more than one type. A room like the kitchen is a *room* and a *non-path-room*; a hallway is a *path* as well as a *room*, because it serves as a way to travel from one place to another. A walkway or a stairway is also a *path*. A door is a *doorway*, which is something that gives access from one room to another; it is also, like a window, a *structure*, which is something that is an integral part of a house.

The representation in the KB of the particular house includes about twenty-five objects of the types listed above, as well as several properties. The objects, properties, and relations for the house are included in the KB in appendix C.

The *elaborating* **relations** are faces, comprises and its inverse composes, contains and its inverse contained-by, and has-property. The *basic* **relations** are next-to, next-to-same-direction, next-to-opposite-direction, next-to-orthogonal-direction, next-to-doors, next-to-doors-same-direction, next-to-doors-opposite-direction, and next-to-doors-orthogonal-direction. There are also four relations used by the spatial metastrategies: east-of; south-of; west-of; and north-of.

The next-to family of relations makes up the set of *basic relations* that are exploited by **find-next-to-say**. There are two basic types: next-to and next-to-doors. This bifurcation captures the difference between objects such as rooms and furnishing being next to each other by physical proximity, and rooms being next to rooms and next to doorways because you can get from one to the other through a door. For example, in the house that is encoded in the KB, the livingroom is next-to ann-room, but they are not next-to-doors; however the livingroom is next-to-doors the long-hallway and the short-hallway.

next-to and next-to-doors are *direct relations* (as are the elaborating relations), that is, they are asserted at the time the KB is created. The other next-to relations, such as next-to-same-direction, are *computed relations.* The direction in question is the spatial direction the description in the house text has been moving in. This cannot be known *a priori*, so it must be computed dynamically.

has-property, faces, and comprises are nonrecursive relations; these relations need not be inhibited during the construction of a text, because they will not lead off to a chain of nodes away from the current node.

This is the partial ordering on applications that the metastrategy **choose** uses to resolve multiple matches in the house domain:

```
(say-first-object)
(say-object)
(say-property)
(say-elaborating-relation)
(find-salient-object)
(find-next-to-say)
(say-multiple-sweep say-left-right-and-center)
(choose)
```

say-first-object is ordered before **say-object** because both match on the first object, and we want to be sure that **choose** selects the right one. **say-property** and **say-elaborating-relation** follow **say-object** because it makes sense to elaborate on an object after you have realized it. The saying strategies are ordered before the finding strategies because it makes sense to say what you can about the current node before finding a next one to talk about. **find-salient-object** is ordered before **find-next-to-say** because if there is in fact a salient object, we want to make sure that it is the next object selected to be the current node. The three metastrategies come last: their ordering does not in fact interact with the ordering on the other strategies, since metastrategies are only triggered when there are multiple matches. The two spatial metastrategies, **say-multiple-sweep** and **say-left-right-and-center**, discussed in section 5.4, are ordered before the default metastrategy **choose**, because they offer the possibility of more interesting text.

The ordering on indifference classes for the elaborating relation preferences is

```
(contains)
(comprises faces)
```

While has-property is an elaborating relation, it is used by the saying strategy **say-property**, not by **say-elaborating-relation.**

The preference ordering on basic relations used by **find-next-to-say** is

```
(next-to-doors-same-direction)
(next-to-doors-orthogonal-direction)
(next-to-doors-opposite-direction)
(next-to-doors)
(next-to-same-direction)
(next-to-orthogonal-direction)
(next-to-opposite-direction)
(next-to)
```

The next-to-doors relations are ordered before the next-to relations because if the current node is a room, a room reachable by a door is preferable to a room that is not for the next current node. (Non-room objects do not bear next-to-doors relations to anything.) A text is more coherent if it follows a course through the domain that continues in the same direction than if it reverses course and goes in the opposite one (and the orthogonal direction is of intermediate desirability). Sometimes no choices with directionality are available, in which case the unmarked choices of next-to-doors or next-to may be taken.

The completeness criterion for the house is fulfilled when all the non-path rooms (excluding the bathroom) have been mentioned. Thus, when the null strategy **jump** is applied, if one of these rooms is not marked as *already-said?*, it is selected to continue the text.

5.2 Deixis

5.2.1 Salix's Deictic Mechanism

This section describes the model implemented in Salix for selecting appropriate *deictic* spatial expressions during the construction of texts in the house domain.[1] Deictic expressions must be computed dynamically during the construction of a text; they cannot be explicitly encoded in the KB. In contrast to deictic expressions, intrinsic expressions refer to stable properties of objects; the end of this section discusses how Salix can be extended to handle intrinsic usage as well.

Most common spatial expressions, such as "left," are used only deictically.[2] Salix computes these dynamically; spatial relationships in the knowledge base are encoded only in terms of the cardinal compass directions. Some spatial expressions, notably "opposite," are used intrinsically as well as deictically, and this section concludes with a discussion of how Salix's mechanism can be extended to handle this usage.

Approximately half the clauses in the house and apartment corpora involve the location of objects in space, nearly always with respect to some reference object. The relationship between object and reference object is most frequently given in terms of the *secondary deixis* system (Ullmer-Ehrich 1982): **front**, **back**, **left**, and **right**. (The *primary deixis* system consists of **here** and **there**.) In principle, all of these spatial terms may be considered ambiguous between a deictic and an intrinsic meaning. Usage varies, but I reserve the term *intrinsic* for cases in which the reference object has an identifiable front and back (or distinguished left and right sides) with respect to which some other object is located. A spatial term used intrinsically makes no reference to the speaker's position (or to that of any observer), but only to the surfaces of the reference object (e.g., we can tell someone that she left her book **in front of** the television in the other room, because a television has an identifiable intrinsic **front**). *Deictic* spatial terms, on the other hand, take their meaning from the observer's perspective on the reference object, ignoring any asymmetries of the latter. Thus, deictic **in front of** designates the space between the observer and the reference object, and deictic **in back of** (**behind**) means the space on the far side of the reference object from the observer's point of view, regardless of the reference object's orientation.

The terms **front** and **back** in my corpora are unambiguously intrinsic only in fixed phrases such as "the front door" and "the backyard." In most cases, the deictic and intrinsic uses of

[1] This model was developed in conjunction with Alison Huettner. Parts of this section are adapted from (Sibun & Huettner 1989).

[2] For thorough overviews of deixis, see Miller & Johnson-Laird 1976, chapter 6, and Anderson & Keenan 1985. For a thorough treatment of spatial prepositions, see Herskovits 1986.

front and **back** are indistinguishable, because the observer is facing the reference object and the reference object is generally thought of as facing the observer. Deictic and intrinsic **left** (or **right**), on the other hand, are distinct under these circumstances, and speakers invariably use these terms deictically. For uniformity, I have treated the many ambiguous uses of **front** and **back** as deictic as well. Since deictic orientations must be dynamically determined during construction of a text, they are at once more complicated and more interesting than hard-wired intrinsic orientations.

The KB includes the four compass relations east-of, south-of, west-of, and north-of. Nodes representing the physical objects in the house domain bear one or more of these relations to each other. For example, if an object is northeast of another, this is encoded by both a north-of and an east-of relationship from one to the other, and by a south-of and a west-of relationship in the reverse direction.

All four directions in the KB record their opposite and orthogonal directions. For example, **north**'s opposite direction is **south**, and its orthogonal directions are **east** and **west**. This information is used in computing the *trajectory*, as well as by the house-domain metastrategies.

All uses of spatial expressions in Salix's house domain texts are deictic and dynamically determined. Generation of these deictic expressions involve a set of variables, collectively referred to as the *frame of reference.*

The variables involved are the *current direction*, the *previous direction*, the *trajectory* (which is a function of the preceding two), and the *speaker position.* The trajectory is used by the *trajectory hook*, which is called by **find-next-to-say**; it also plays a role in triggering **say-left-right-and-center**. The trajectory and the speaker position together are used to compute the deictic terms that are used in the scope of the spatial metastrategies.

Each application of **find-next-to-say** follows a next-to link. Usually, the new current node bear some direction relation, e.g., east-of the previous node. This direction *east* is recorded as the current direction (and whatever used to be the current direction is recorded as the previous direction). If the current direction is the same as the trajectory (e.g., they are both *east*) or if the current direction is orthogonal to the trajectory (e.g., the trajectory is *east* and the current direction is *north* or *south*), the trajectory is not changed. If the current direction is opposite the trajectory, the trajectory is set to *nil.* If the trajectory is *nil*, then it is set to the current direction.

The computed relations, such as **next-to-same-direction** are computed with respect to this trajectory. For example, if the trajectory is *east*, and an object next-to the current node is also east-of it, this object will be "next-to-same-direction" of the the current node, and will trigger **find-next-to-same-direction**.

The spatial metastrategy **say-left-right-and-center** is triggered by a **find-next-to-same-direction** application and two **find-next-to-orthogonal-direction** application. This will be described in more detail in section 5.4.

The *realization mode* of all the next-to relations is *spatial-deixis-as-pp+be*; this realization mode is used by **say-object** to decide what sort of text to realize when the current relation is one of the next-to relations (see section 7.4.7 for more details). In the default case, the realized text does *not* include a deictic expression: a spatial relation can be expressed deictically only in reference to some point of view, e.g., that of the speaker who is located (either literally or figuratively) at some particular *speaker position.* The two spatial metastrategies set a speaker position with respect to which deictic terms can be computed, and set a *speaker mode* which enables the realization of deictic expressions. Examples are described in detail in section 5.4.

5.2.2 Extensions to the Spatial Model

A spatial expression that does not immediately fit into the framework as outlined above is "opposite," and its variants "diagonally opposite" and "kitty-corner from." While **left**, **right**, **front**, and **back** are best computed dynamically, the situation with **opposite** is more complex. There are two cases we must consider: that in which something is opposite the speaker (deictic **opposite**); and that in which two objects are opposite each other (intrinsic **opposite**).

An object is a candidate for being described as deictically **opposite** if it is removed from the speaker-position in the direction of the trajectory. For example, if the external-reference-point is east, speaker-position is **livingroom-door**, and these relations hold in the knowledge base

(coffee-table east-of livingroom-door)
(couch east-of coffee-table)

then **coffee-table** and **couch** are candidates for being **opposite** the speaker. There is a further defining characteristic: objects that are **opposite** have either another object or empty space between them. Salix's representation captures this relationship neatly: objects related by **next-to** cannot be considered **opposite** each other. I believe therefore that I can model the use of deictic **opposite** by allowing its usage for objects that are situated in the direction of the trajectory from speaker-position and that are not next-to speaker-position.

A speaker's ability to say that two objects are **opposite** each other does not vary with her own position; this, then, constitutes an intrinsic use of a spatial expression. In my corpora, all intrinsic **opposites** involve walls, doors, and windows (which are always embedded in walls), and corners, which are defined as the meeting of two walls. The knowledge base records the relationships between the walls of a room, so it is a straightforward computation to determine whether walls, doors, or windows are **opposite**. "Diagonally opposite" or "kitty-corner from" can be used to describe corners that are **opposite** (because their walls are pairwise **opposite**), as in "The door is kitty-corner from where the windows are."

With the exception of "opposite," there are no instances in which an intrinsic perspective need be taken to select an appropriate spatial expression. However, there are certainly times when a speaker does in fact take an intrinsic perspective. To handle this, Salix would need a mechanism to allow intrinsic use, but the KB should not be burdened with intrinsic sides for each object. Not only will this information usually not be needed, but it may sometimes be inappropriate, or need to be computed (for instance, intrinsic sides are inappropriate for a dining table, but would need to be computed if that table were pushed against the wall). One solution would be what might be termed a *two-mode* system. The first mode reflects the situation in which the objects are considered physical items (with labels such as **stove**) and nothing more. This is the mode that is already implemented. The second mode comes into force when the object is considered in terms of some of its inherent properties, such as those involved in functionality or animacy (Miller & Johnson-Laird 1976, p 400). These properties may include or entail intrinsic **fronts** or even **lefts** and **rights** (consider, for example, a toy kitten or a newspaper sheet).

The question remains of what would trigger a shift between modes. The first mode, objects *qua* objects, will be the default in giving a description of a living space layout. The second mode will become appropriate in a description in which such things as the functionality of the object play a role. This may occur, for example, in reply to questions like " How do you make supper in your kitchen?" and "What do you like and dislike about the way your house is arranged?" This is a topic for future research.

5.2.3 Another System Using Deixis

Retz-Schmidt (1986, 1988) describes a computer program, CITYTOUR, that is capable of generating both intrinsic and deictic expressions. CITYTOUR is part of the VITRA project, which is developing interfaces between image understanding and natural language systems. CITYTOUR consists in a *city map* of buildings and streets, as viewed from above, and a *tour bus* which can be moved about the streets, indicating the position of the observer simulated by the program when it answers questions about relative positions of the buildings. The important components of a building representation are a *delineative rectangle*, which regularizes the *polygon* outline of the building, its *center of gravity*, and the building's *prominent front* (it is prominent because it is the intrinsic front). It is important to note that the prominent front is user-defined and is not computed dynamically. The delineative rectangle determines four half-planes around the object, each beginning at the line coincident with one of the rectangle's sides. If an intrinsic description is being constructed, the half-plane corresponding to the prominent front is labeled **front**, and the others are labeled clockwise as **right**, **back**, and **left**. For a deictic description, a *line-of-sight* is drawn from the observer's position to the delineated rectangle, and the nearest half-plane is labeled **front**, and the others **left**, **back**, and **right**.

CITYTOUR uses a three-argument representation for spatial expressions: the *subject*, the *reference object* (in relation to which the subject is located), and the *point of view*. The *observer's position* may or may not be one of these. The use of an expression is intrinsic when the reference object and the point of view are the same, and deictic when the point of view and the observer are the same. CITYTOUR operates in only two dimensions, so vertical displacement is not a factor.

Intrinsic use of expressions in CITYTOUR is the *unmarked* (essentially, default) case. If intrinsic use is impossible (because of no prominent front) the deictic system is used. If the deictic is used but the intrinsic is possible, the deictic use is marked by 'from here.' This avoids ambiguity by explicating the point of view.

Retz-Schmidt examines an additional problem: the degree of applicability of prepositions, which depends on the size of the object and its distance from the reference object, and that can be expressed by such terms as 'directly' or 'almost' (contrast 'directly behind' and 'almost behind'). However, the model described in this work for choosing between deictic and intrinsic descriptions where both are possible is not sufficiently complex to account for all the ways people handle the situation.

5.3 Example House Text without Spatial Metastrategies

This section discusses an example of the house strategies at work, when the only metastrategy available is **choose**. In section 5.4, we will see that adding metastrategies specific to spatial domains can improve the text.

The sample outputs in figure 5.1 both begin at the kitchen window in the house being described (we will assume that this fragment is not from the beginning of a text). We will walk through the production of the first one, and note where the second one differs.

In the initial call on **dispatch** for this fragment, the dispatchee is the node large-kitchen-window. A number of strategies match, so there is a dispatch on the resulting conflict set. On this dispatch, the metastrategy **choose** is triggered.

In this domain, **choose** prefers all the saying strategies to all the finding ones, since in general it makes sense to describe an object before moving on to the next one. **choose** also orders **say-**

```
(A1) and then there's the window
(A2) which is a picture window
(A3) and is large
(A4) it has two flanking windows
(A5) and faces the backyard
(A6) and then there's the sliding glass door
(A7) then the window
(A8) which is small
(A9) then there's the closet

(B1) then there's the window
(B2) which is large
(B3) and is a picture window
(B4) it has two flanking windows
(B5) and faces the backyard
(B6) and then there's the sink
(B7) and the stove
(B8) then the refrigerator
```

Figure 5.1: Two versions of part of one of Salix's house texts.

object before **say-property** and **say-elaborating-relation**, since it is important to say what an object is before saying things about it. Thus **choose** selects the strategy **say-object** (see line A1 in figure 5.1).

say-object has several realizations, which vary with context. The most important aspect of the context is whether the **say-object** is within a series of **say-objects** or follows some other sort of strategy. If the last increment of text was not produced by **say-object**, then it is necessary to mark the introduction of one (or more) new objects with an existential "there," and the choices are:

```
then there's the <object>
and then there's the <object>
```

If, instead, the most recent increment of text was produced by say-object (see lines A7 and B7), the choices are:

```
and the <object>
then the <object>
```

At the end of **say-object**, the current node is marked as *already-said?*, which prevents it from being introduced again into the text (for example, by another application of **say-object**). Since at the end of every saying strategy, the current node is dispatched on again, after increment (A1) is realized, **dispatch** is called with large-kitchen-window. Many strategies are triggered, but since **say-object** no longer applies, **say-property** is at the top of **choose**'s preference list. There are in fact two **say-property** applications: large-kitchen-window is linked to the properties of being large and being a picture window. Either application may be chosen by **choose**, the appropriate text generated, and the property marked as already said. (**choose** will pick the other **say-property** application the next time.) **say-property**'s realizations also vary with context; see lines (A2-A3)

and (B2-B3). If the property is the first one mentioned of the object of which it is a property, then it is expressed with a full clause, one of:

```
it is large
which is large
```

If the increment described a property that is one of a series of properties, then it can be expressed in abbreviated form:

```
and is large
```

In this text, dispatching continues on large-kitchen-window until all the **say-property** and **say-elaborating-relation** applications have been used. Then in the next dispatch cycle, there is nothing more to say about large-kitchen-window; only finding strategies are available. (They have, of course, been available all along.) There are several next nodes from large-kitchen-window to describe: because no trajectory has been established, any of these available next nodes can be chosen. Indeed, different choices, sliding-glass-door and sink, were taken in the two samples in figure 5.1.

The rest of both samples are constructed by following the trajectory established in the context. That is, for each node, several finding strategies are applicable, and **choose** selects the one that continues the text in the same general spatial direction.

5.4 Example House Text with Two Spatial Metastrategies

In this section, we will walk through in greater detail the house text presented in chapter 1 and repeated here (figure 5.2). The discussion will include the introduction of the two spatial metastrategies **say-left-right-and-center** and **say-multiple-sweep**.

The text starts with a dispatch on side-door, which is made the current node. Several strategies match and return applications. **find-next-to-say** returns two applications: one with the relation next-to and the object entrance-hallway; and one with the relation **next-to-doors** and the object entrance-hallway. **say-object** and **say-first-object** both match with side-door. **say-first-object** has the highest preference in **choose**'s ordering, so this application is selected, resulting in the introductory cue phrase in (1) and the introduction of the first object in (2). side-door is marked as *already-said?* and **say-first-object** dispatches on side-door again.

The same two finding strategy applications are returned, and the application of **find-next-to-say** with next-to-doors and entrance-hallway is selected. entrance-hallway becomes the new current node and is dispatched on, and **choose** selects the **say-object** application from the available candidates, which results in increment (3). entrance-hallway is marked as *already-said?* and dispatched on again.

Now that there has been a finding strategy previously applied, the *trajectory* has been established; it is set to *south*. Four **find-next-to-say** applications are returned. Two are with the relation next-to-orthogonal-direction: one with the object kitchen and one with the object livingroom. Two are with the relation next-to and the same two objects. The next-to-orthogonal-direction applications are preferred, and one is selected nondeterministically, in this case, the one with the kitchen. The kitchen is dispatched upon.

```
(describe-house side-door)

     (1)   OK
     (2)   we can start at the side door
     (3)   and then there's a entrance hallway
     (4)   then
     (5)   in the kitchen
     (6)   there is a window
     (7)   which is large
     (8)   and is a picture window
     (9)   it has two flanking windows
    (10)   and faces the backyard
    (11)   and if we're facing that window
    (12)   on the right is the sliding glass door
    (13)   and a window
    (14)   which is small
    (15)   if we're facing the backyard
    (16)   on the left is the stove
    (17)   and a refrigerator
    (18)   and if we're facing that window
    (19)   underneath is the sink
    (20)   then a dishwasher
    (21)   then there's a living room
    (22)   it has
    (23)   a window
    (24)   which is large
    (25)   in it
    (26)   and then there's a short hallway
    (27)   and
    (28)   Ann's bedroom on the left
    (29)   Claire's bedroom on the right
    (30)   and
    (31)   a bathroom in the middle
    (33)   oh yeah
    (34)   and then there's Penni's bedroom
    (35)   and a long hallway
    (36)   that's it
```

Figure 5.2: House description generated by Salix.

Many applications are returned by **match**: there is an application of **say-elaborat-ing-relation** with the relation contains and all the furnishings in the kitchen; there are seven applications of **find-next-to-say** for various rooms that have various next-to relations to the kitchen; there is an application of say-object; and there is a application of find-salient-object with the object large-kitchen-window.

All these strategies constitute a multiple match, and they are bundled up with the node kitchen and dispatched on again. This time, two metastrategies match. **choose** matches because it is always triggered by multiple matches, and it returns the application of **say-object**, which ranks the highest on its preference list. **say-multiple-sweep** also matches, and returns an application. So there is again a multiple match, which is again dispatched on. This time, **choose** is the only metastrategy to match, and it selects the application of **say-multiple-sweep**.

Let us take a moment to examine the trigger of the spatial metastrategy **say-multiple-sweep** (see figure 5.3). This metastrategy is triggered by a multiple match, and by the presence of an application of **find-salient-object** among the candidates. A *salient object* is an object which is related to (by a relation such as contained-by) the current one, and is involved in many relations with other objects and properties. **say-multiple-sweep** gathers up all the objects related to the salient object by a next-to relation and bearing the same contained-by relation to the current object. It next takes each of these next-to objects and calculates a *sweep* that begins with the object and continues along the trajectory from the salient object to the next-to object; each object in the sweep must also bear the same contained-by relation to the current object. An application is returned including the current object, the salient object, and the list of sweeps.

The application of **say-multiple-sweep** with the current node kitchen has been selected. This strategy inhibits recursions, so that strategies involving recursive relations, which would take the text off the subject of the current node, will not match. The *speaker mode* is set and the *speaker position* is set to be the salient object.[3] To indicate that the focus of the text is shifting to what will be produced by **say-multiple-sweep**, an introductory cue is realized (4). The multiple sweep is then *situated* (5) in the kitchen: the multiple sweep is built around the salient object, but it is necessary to indicate that the salient object is salient with respect to the current object. The salient object, the large-kitchen-window, is then realized (6). **say-salient-object**, like **say-object**, marks the object as *already-said?* and dispatches again on the same node. Thus, the large-kitchen-window is elaborated on (7-10), as described in section 5.3. Recursion is inhibited, but all the elaborating relations the large-kitchen-window is involved in are nonrecursive.

Next the sweeps must be realized. For each sweep, there is an *orientation* to the salient object (11,15,18). Because in this case the salient object is a window, the orientation can be expressed in terms of facing the window (11,18), or in terms of facing what the window is facing (15), in this case, the backyard. After the orientation, **say-object** is called on each object of the sweep in turn. Before each call on **say-object**, the trajectory is updated, just as though the object has been selected by an application of **find-next-to-say**. The objects in the sweep may be elaborated on if they are involved in nonrecursive elaborating relations (13-14). Because the speaker mode is set, the appropriate deictic expressions may be realized in **say-object**, and they are realized for the first object in each sweep (12,16,19). However, within a sweep, the deictic expressions would be redundant, so they are suppressed (13-14,17,20).

Just as with any saying strategy, at the end of **say-multiple-sweep**, the context is saved, and the current node, the kitchen, is dispatched upon. Now, only the seven **find-next-to-say** applications are returned by **match**. **say-object** does not match, because the kitchen has been

[3]To set the speaker position on an item of furnishing or structure such as the large-kitchen-window does not mean to suggest that the speaker is on or in such an object, but merely near it.

```
(defmetastrategy say-multiple-sweep
    (let ((salient-object (third (assoc 'find-salient-object candidates))))
      (when salient-object
        (let* ((so-cb (first (contained-by-objects salient-object)))
               (next-tos
                (remove-if-not
                 #'(lambda (next-to)
                     (and (contained-by-same-thing?
                             so-cb
                            (first (contained-by-objects next-to)))
                          (can-continue-sweep? next-to salient-object)))
                  (next-to-objects salient-object)))
               (sweeps nil))
               (dolist (next-to next-tos)
                 (let ((next-to-chain
                        (find-next-to-chain salient-object so-cb next-to)))
                   (when next-to-chain
                     (push (cons next-to next-to-chain) sweeps))))
               `((say-multiple-sweep ,object ,salient-object ,sweeps))))))

(defun say-multiple-sweep (object salient-object sweeps)
  (let ((*inhibit-recursion* t)
        (*speaker-mode* t)
        (*speaker-position* salient-object))
    (realize-intro-cue)
    (say-situate object)
    (say-salient-object salient-object)
    (dolist (sweep sweeps)
      (orient salient-object)
      (funcall *trajectory-hook* salient-object (car sweep) nil)
      (say-object (car sweep))
      (let ((*inside-sweep* t))
        (dolist (s (cdr sweep))
          (funcall *trajectory-hook* (last-said) s nil)
          (say-object s))))
    (save-context))
  (dispatch object))
```

Figure 5.3: The trigger and action procedure of the metastrategy **say-multiple-sweep**.

```
(defmetastrategy say-left-right-and-center
    (let ((same-dirs (get-find-type candidates (get-thing next-to-same-dir)))
          (orthos (get-find-type
                    candidates (get-thing next-to-ortho-dir)))
          (lrcs nil))
      (when same-dirs
        (let ((same-dir (random-elt same-dirs)))
          (do ((first-ortho-tail orthos (cdr first-ortho-tail)))
              ((null first-ortho-tail))
            (let ((first-ortho (first first-ortho-tail)))
              (dolist (second-ortho (rest first-ortho-tail))
                (when (opposite-dirs?
                       (figure-out-dir
                        (third first-ortho) (third second-ortho))
                       (figure-out-dir
                        (third second-ortho) (third first-ortho)))
                  (push `(say-left-right-and-center
                             object ,first-ortho ,same-dir ,second-ortho)
                        lrcs)))))))
      lrcs))

(defun say-left-right-and-center (object one-hand center other-hand)
  (let ((*inhibit-recursion* t)
        (*speaker-mode* t))
    (set-speaker-position object)
    (realize-and)
    (apply-strategy one-hand)
    (apply-strategy other-hand)
    (realize-and)
    (apply-strategy center)
    (save-context))
  (dispatch node))
```

Figure 5.4: The trigger and action procedure of the metastrategy **say- left-right-and-center**.

marked as *already-said?* by **say-salient-object**. **say-elaborating-relation** with the relation contains does not match, because all the kitchen's contents have been realized and marked as *already-said?* in the course of **say-multiple-sweep**, and the inference rules have marked the contains links as *already-used?* (see section 3.5). The application of **find-next-to-say** with the relation next-to-same-direction and the object livingroom is selected by **choose**, and the livingroom becomes the current node.

When the livingroom is dispatched on, **match** returns an application of **say-object**, eight applications of **find-next-to-say**, and an application of **say-elaborating-relation** with the relation contains and the singleton set of objects large-livingroom-window. After the **say-object** application is applied (21), on the next dispatch **choose** selects the application of **say-elaborating-relation**. The relation contains is realized in increments (22) and (25), and **say-object** is called on each of the related objects, in this case, large-livingroom-window (23). **say-elaborating-relation** inhibits recursion, but has-property is a nonrecursive relation, so when **say-object** dispatches on large-livingroom-window, **match** returns an application of say-property with the property largeness, and **choose** selects the application (24).

say-property dispatches again on the current node, large-livingroom-window. This time, however, no applications are returned by **match**; there are no more nonrecursive elaborating relations on the large-livingroom-window. This no matches impasse is dispatched on, and the null strategy **noop-on-inhibit-recursion** matches.[4] It returns control to **say-elaborating-relation** which again dispatches on the livingroom.

The next object found and said is the short-hallway (26). When the short-hallway is dispatched on again by **say-object**, ten **find-next-to-say** applications are returned by **match**, including: one involving the relation next-to-same-direction and the object bathroom; one involving the relation next-to-orthogonal-direction and the object ann-room; and one involving the relation next-to-orthogonal-direction and the object claire-room. When the multiple match is dispatched on, the presence of these three applications triggers the spatial metastrategy **say-left-right-and-center** (see figure 5.4). The metastrategy checks that the orthogonal directions are opposite each other, and returns an application including the current node and these three **find-next-to-say** applications.

choose also matches, and when this multiple match is dispatched on, **choose** selects the application of **say-left-right-and-center** (27-31). **say-left-right-and-center** inhibits recursion, sets the *speaker mode* to enable deixis, and sets the *speaker position* to the short-hallway. The applications of the two orthogonal finding strategies are applied first, one after the other (the order does not matter), followed by "and" and the application involving the object that is in the same direction as the trajectory. In each case, the application of **find-next-to-say** is applied, and the next node, for example ann-room is dispatched on. Because recursion is inhibited, the only application available for ann-room is **say-object**, which is applied. Because speaker mode is set, deictic terms are enabled; this results in the "left, right, and center" realization of the metastrategy. Then no strategies match, and control is returned to **say-left-right-and-center** by the null strategy **noop-on-inhibit-recursion**.

At the end of **say-left-right-and-center**, the short-hallway is dispatched on again, and there are no matches.[5] The *completeness criterion*, that all the major rooms be mentioned, has not been fulfilled, so the null strategy **jump** matches. In the house domain, **jump** cues that there will be a jump (and thus probably a spatial discontinuity) in the text (33). penni-room has not been marked

[4] A similar situation obtained for all the objects mentioned in the course of the multiple sweep; the discussion was deleted there for clarity.

[5] There *should* be matches, e.g., a find on long-hallway, but there are not, due to a subtle bug in the implementation. I have left the bug in, because it allows me to illustrate Salix's behavior in a "forgetting" situation.

as *already-said?*, so it is dispatched on by **jump**, and realized (34). When penni-room is dispatched on again by **say-object**, a **find-next-to-say** application with the object long-hallway is selected and the long-hallway is realized (35). When long-hallway is dispatched on, there are no matches, and the completeness criterion has been fulfilled, so the null strategy **say-all-done** matches, signals the end (36), and the text is done.

An execution trace of part of a text in the house domain appears in appendix D.

Chapter 6

Application III: Physical Processes

In this chapter, I discuss the extension of Salix's text generation capabilities to a new domain. I show what is interesting and different about this domain, physical processes (Forbus 1985), and describe a proposed implementation of this application in Salix.

6.1 Talking about Processes

As emphasized in this thesis, Salix structures text by domain structure. In order to do this, Salix needs to know what are the *basic relations* of a domain, that is, what relations it can exploit to find the next thing to add to the text. In spatial domains, such as that of a house, the basic relations are spatial ones. We might hypothesize that in a temporal domain, temporal relations could serve as basic relations.

Little work has been done in text generation on generating temporally structured texts (though the issue has been addressed, e.g., Maybury 1990). Perhaps the most impressive temporally structured generation system is Meehan's TALE-SPIN (1977). This Conceptual Dependency-based system told Aesop's fables-like stories about animal characters interacting and having simple adventures.

Building texts in domains that are temporally structured is in principle no different from building texts in domains that are spatially structured. However, because language itself has temporal extent, it is sometimes hard to tease apart the temporal structure of the subject matter and that of the text. For example, temporal structure is inherent in all interactive systems and all generation systems with a user model, since the (temporal) order in which things appear in the text has an impact on what a hearer might ask or what a speaker can infer that the hearer already knows. However, this is distinct from the temporal structure in the domain—the order in which things happen.

A difficulty in generating texts about domains with temporal structure is that the representation of states, processes, events, and time in general is very hard, though the problem has certainly been addressed (e.g., Allen 1984). For example, events can be instantaneous or extended, can overlap or be coterminous, and can bear all sorts of relations to each other which may be reflected in the text, particularly in the tense and aspect of the verbs and the temporal deixis expressions (such as "now" and "before"). In order to circumscribe the problem while still demonstrating that temporal relations can be exploited as basic relations, I selected the domain of *physical processes* as represented in Forbus's Qualitative Process (QP) Theory (1985).

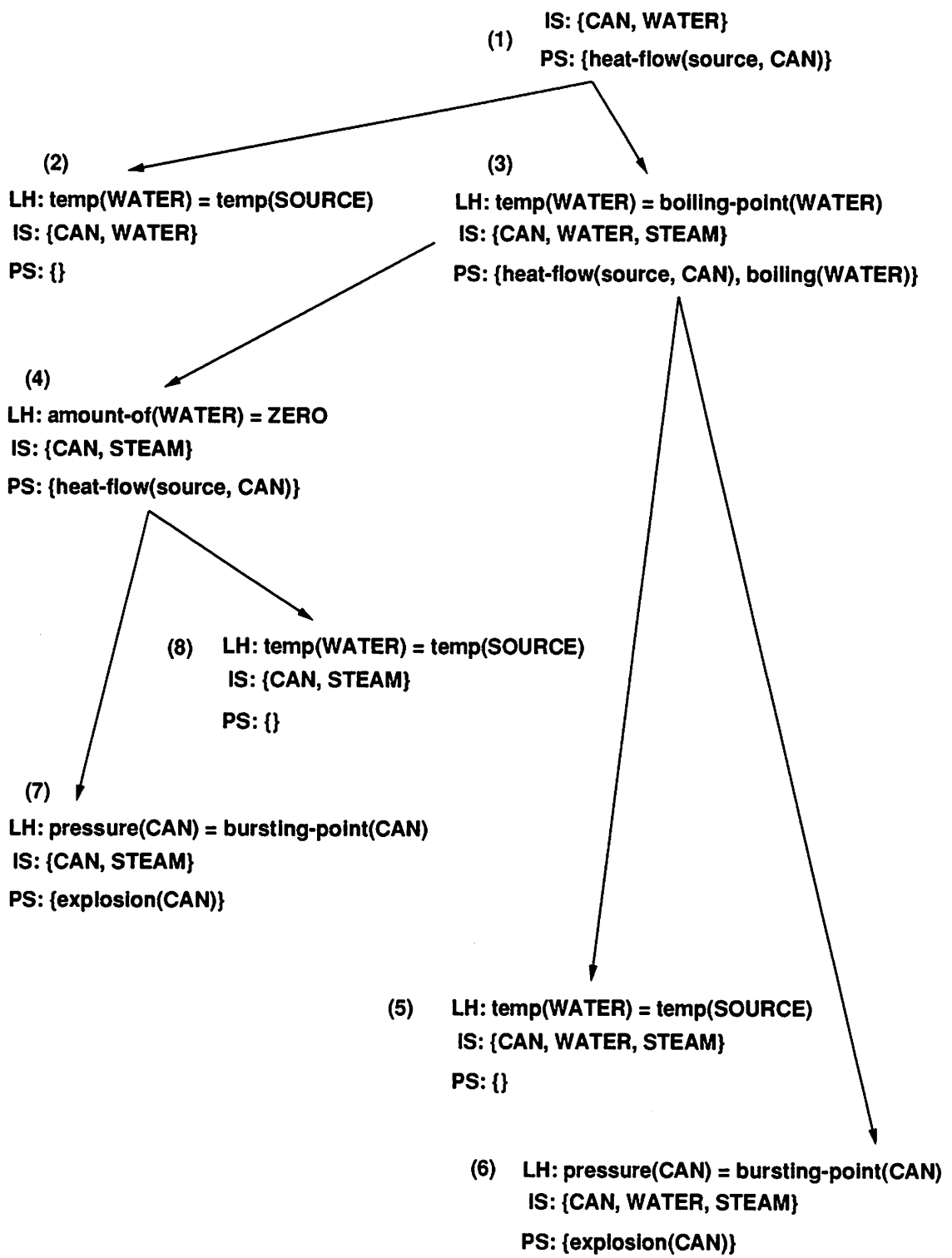

Figure 6.1: Graph representing the process structures for water being heated in a sealed container. Adapted from (Forbus 1985, p. 131).

initially
there is a heat flow from the source into the container
the temperature of the water may reach the temperature of the source
alternatively
the temperature of the water may reach the boiling point
in this case
boiling produces steam
then the amount of water may go to zero
if the pressure in the can reaches the bursting point
there is an explosion
alternatively
the temperature of the water may reach the temperature of the source
in which case equilibrium is reached

in a different situation
the temperature of the water may reach the temperature of the source

alternatively
the pressure in the can may reach the bursting point
in which case there is an explosion

Figure 6.2: Text corresponding to representation in figure 6.1.

Forbus characterizes a *process* as a change in objects over time. For example, if the objects are a sealed container and water inside that container, and if heat is applied from some source, changes will occur in the water and the container. People can reason qualitatively about what these changes might be, for example, by saying "the temperature of the water may reach the boiling point, thereby producing steam." This reasoning can be represented as a space of possible outcomes extended over time. Forbus developed a way to represent such process structures; his graph representing what can happen when water is heated in a sealed container is reproduced in figure 6.1. The nodes in the graph correspond to situations; the arcs correspond to time. Each situation is characterized by a *limit hypothesis*. A limit hypothesis describes a qualitative change in a process; this occurs when an equality or inequality between quantities is reached. For example, the temperature of the water reaching the boiling point is a limit hypothesis.

A text, generated by a colleague of mine, describing Forbus's representation, can be found in figure 6.2. (One state, that in which the temperature of the water drops below the boiling point has been deleted because it is an artifact due to "stutter." See Forbus 1985, p 152.)

In the next section, I outline proposed extensions to Salix to enable it to generate text in a temporally structured domain, as represented in Forbus's QP theory.

6.2 Proposed Physical Process Domain Implementation

The most important type of object in the physical process domain is the situation. The situations are represented by the nodes in figure 6.1. The *basic relations* in this domain are previous-situation and next-situation. These relations hold, respectively, between a situation and one that immediately

precedes it in time, and between the situation and one that immediately follows it in time. These two relations are *inverse relations*, and capture the asymmetry of physical processes, which change, from one situation to a next, over time.

The *elaborating relations* are those that relate aspects of a situation to the situation itself. These aspects of the situation, which are encoded in the knowledge base, include *processes*, such as heat-flow and explosion, *individuals*, such as container, and *constants*, such as boiling-point-of-water. The *completeness criterion* for a text about a physical process is that all the situations be mentioned.

As can be seen in figure 6.1, the structure in the physical process domain is slightly more constrained than in the domains examined so far in this thesis. Forbus chose to use tree structures to represent physical processes; this has two consequences. First, the next-situation and previous-situation links form two distinct sets: all the "down" arcs in the tree and all the "up" arcs (these latter arcs are not shown in figure 6.1). Because these two sets of links express complementary information, it is better to stick to one set as much as possible, usually the "down" links, which represent forward progress in time. Therefore, there needs to be a mechanism for systematically ensuring that this is the case. Second, part of the point of a tree structure is that there is a particular relationship between a node and the set of its children. In other words, part of the information available in the tree structure is whether other children of a node, as well as a node itself, have been *already-said?*. For example, if **find-next-to-say** finds a situation that is a child of the situation that is the current node, and another child of the current node is *already-said?*, then Salix can indicate that it is taking an alternative path descending from the situation that is the current node.

These differences in domain structure can be handled by augmentation to Salix's current model. I will consider first the issues involved in traveling up and down a tree structure, and then the issues involved in selecting alternative children of a node.

```
(defstrategy find-unsaid-to-say node
  (if (not *inhibit-recursion*)
      (let ((forms nil))
        (dolist (basic-relation *basic-relations*)
          (dolist (next (related-objects-to-say node basic-relation))
            (when (object-already-said? next) ;; cf. find-next-to-say
              (push '(find-unsaid-to-say ,node ,next ,basic-relation)
                    forms))))
        forms)
      nil))
```

Figure 6.3: The action procedure of the strategy **find-unsaid-to-say**.

Salix's current finding strategy, **find-next-to-say**, will only match with knowledge base nodes related to the current node by a basic relation that are not marked as *already-said?*.[1] If no applications of **find-next-to-say** are returned, but the completeness criterion has not been satisfied, the null strategy **jump** applies and nondeterministically selects an unsaid node from the completeness criterion fulfiller to be the new current node. Jumps can be marked (e.g., by "wait I forgot"), but the resulting text may be suboptimally choppy. In order to retain the information encoded in the tree structure of the knowledge base in producing the text, Salix must be able to backtrack.[2] So

[1] **find-salient-object** behaves in this way also, but it is not an important strategy in this domain.

[2] Salix need not literally backtrack; it could build a stack of choice points visited. However, maintaining a stack is

Salix needs a finding strategy that will allow it to find a new current node that has been *already-said?*. This strategy is almost exactly like **find-next-to-say**, with one crucial difference: instead of returning applications for nodes that are not *already-said?*, it returns them for nodes that are. This strategy is called **find-unsaid-to-say** (see figure 6.3), because, as we shall see, its effect, in repeated applications, is to retrace the path through the KB until an unsaid node is found.

Suppose Salix is asked to produce a text about the process of heating water in a sealed container, starting with the node for situation-1 (labeled (1) in figure 6.1). When Salix is done elaborating on that node, it will nondeterministically choose between an application of **find-next-to-say** with situation-2 and an application of **find-next-to-say** with situation-3. Suppose situation-3 becomes the new current node and is elaborated on, and Salix then nondeterministically chooses situation-4, from the available applications for situation-4, situation-5, and situation-6. Situation-4 is elaborated upon, and situation-7 is selected next. When situation-7 has elaborated upon, and is dispatched on, no applications of **find-next-to-say** are returned, since there is no node related to situation-7 by a basic relation that is not *already-said?*. Rather than jump to some unsaid node, it would be better to backtrack, and find the closest unsaid node, in this case, situation-8.

If **find-unsaid-to-say** is an available strategy, then on many dispatches it will return applications; in the current instance, it will return an application with the node situation-4. If **find-unsaid-to-say** is ordered last in the `*strategy-preference*` of this domain, then this point will be the first opportunity for a **find-unsaid-to-say** application to be applied, since at every previous dispatch an application of either an elaborating strategy or **find-next-to-say** will have been selected. When the application of **find-unsaid-to-say** is applied, situation-4 becomes the current node. When situation-4 is dispatched on, an application of **find-next-to-say** with situation-8 is returned and applied. Similarly, when situation-8 has been elaborated upon, successive applications of **find-unsaid-to-say** with situation-4 and situation-3 will accomplish backtracking to the next point where there is something unsaid (situation-5 and situation-6) available to say.

In this way, by applying **find-unsaid-to-say** as a strategy of last resort, Salix can ensure that it recapitulates the important aspects of the tree structure in the knowledge base in the structure of the text.

The second import of a tree-structured KB is the special status of children of a particular node with respect to each other. In the physical process domain, the children of a node represent the possibilities of what can happen next, as expressed in this fragment from the text in figure 6.2; the corresponding situations from figure 6.1 are indicated.

then the amount of water may go to zero (situation-4)
if the pressure in the can reaches the bursting point
there is an explosion (situation-7)
alternatively
the temperature of the water may reach the temperature of the source
in which case equilibrium is reached (situation-8)

In order to generate informative text, Salix needs to be able to indicate alternative children of a node, with connectives like "alternatively" and "in another case." It can determine this locally in a straightforward manner. The previous node is always part of the local context. If the previous node bears a previous-situation relation to the current node, and if it bears a previous-situation relation to any other node which has been marked as *already-said?*, then Salix knows to use an "alternative" connective, rather than the default "then" to introduce the new current node.

In this chapter, I have indicated the major challenges to Salix in generating texts in the domain

not necessary and would require an extra, nonlocal, data structure.

of physical processes. To make a fully complete implementation requires a certain amount of detailed work as well: encoding the knowledge into the knowledge base; ensuring that all the *lex-items* are available for expressing what needs to be expressed; and making sure that there are *realization functions* suited to the applications of the strategies in the domain.

Chapter 7

Surface Structure Realization

This chapter begins with an overview of the role of surface structure realization in Salix's process of generation. I then discuss several state-of-the-art sentence based realization components. The chapter concludes with a detailed examination of the surface structure realization capability in Salix, both the procedures that actually realize the text and some of the organizing templates provided by the generation modes on relations.

7.1 The Function of the Surface Structure Realization

The surface structure realization functionality in Salix is not confined to a separate realization component. Instead, this functionality resides in a collection of small procedures that are called in the contexts for which they are appropriate. The text generated by each of these procedures may be a connective ("and"), a cue phrase ("OK"), an introductory phrase ("and then there's"), a noun phrase ("Penni's bedroom"), or a verb ("faces").

Coordination between the realization procedures, such as number agreement between a noun phrase and a verb ("the window faces" rather than "the window face") is handled by whatever procedures call the realization procedures. If for some reason this coordination does not happen, a mistake will appear in the output; there is no grammatical checking. No part of the system builds syntax trees or constructs sentences that will then be realized all at once. Each realization procedure produces the text for which it is responsible immediately upon being called.

So far, the description of text construction in Salix has concentrated on the strategic aspects of generation and not the tactical nitty-gritty of syntax. The construction of a text is driven by domain structure, and increments in the text result from the application of strategies which find something next to say and say something about it. More specifically, increments of text have the following sources:

Saying strategies
The bulk of the text is generated by the application of the saying strategies. For example, **say-object** is responsible for texts such as "and then there's the living room" and **say-elaborating-relation** is responsible for texts such as "it faces the backyard."

Metastrategies
Metastrategies such as **say-left-right-and-center** collect and order applicable strategies and dispatch on them; these strategies eventually result in text. In addition, such metastrategies produce cue phrases, connectives, and other text directly. For example, **say-left-right-and-center** directly

generates "and" in texts with the form

```
<X> on the left
<Y> on the right
and
<Z> in the center
```

Null strategies

Some null strategies realize cues when they are applied. For example, the null strategy **jump** may signal a discontinuity in the text by a cue like "wait I forgot."

Salix's model of surface structure realization does not have much precedence in the literature. The closest in spirit is Becker's paper on the Phrasal Lexicon (1974). Becker observed that an English speaker probably knows as many stock phrases as he does individual words. Such phrases range from *polywords* (e.g., "for good," "not as such") through *phrasal constraints* (e.g., "by .... coincidence"), to largely invariant *situational utterances* (e.g., "No, thanks; I'm trying to cut down."). The lexicon can contain *frozen* (invariant) phrases and *semi-frozen* phrases (with some variables, like "the best <noun> I ever <verb + -ed> in my life"). Becker suggests that in fact all use of language is an appropriate combination of such phrases, often tempered by modification to fit new circumstances. Since Becker's Phrases run the gamut of size from a single word to entire passages, such as nursery rhymes, the sentence has no special status in his theory. While there are rules for combining phrases, the rules are more in terms of constraints on what phrases fit with other phrases than on how the pieces contribute to a syntactic structure.

Salix's surface structure realization capability can be thought of as a phrasal lexicon that is comprehensive enough to produce texts in three domains. While this capability does not by any means cover the range of English expression, it is not particular to the domains for which Salix has generated text. Most surface structure realization components, such as those described in section 7.2, strive for generality by being capable of producing as many sentence patterns as possible. The selection of the best sentence pattern depends on information provided by the text planner. Salix's realization capability has a similar set of text components (e.g., connectives and noun phrases) to those that make up other realization components; the way these texts are put together to produce the output text is determined by the strategies using considerations similar to those to those supplied by a text planner. The difference is in large part that Salix makes these choices explicitly without their passing through a layer of abstraction from strategic to tactical component.

7.2 Sentence-based Surface Structure Realization

Most surface structure realization components ("realizers" for short) operate hierarchically because grammatical information for any piece of text (typically a sentence) can be represented as an augmented tree structure. This structure is an explication of syntactic knowledge about clauses and phrases and types of words, and the feature augmentation handles issues like subject-verb agreement, correct ordering of complex verbs, and dropping the subject and object on the proper sides of the verb. In the usual model, a text planner organizes a body of information to be conveyed and the realizer has the capability to produce language from these concepts. The realizer's task, then, is to select the linguistic structures that best convey the desired information.

This section considers programs that assume that most of the interesting strategic decisions have already taken place. The three generation systems presented here are most of the generators

that are currently operational. Interestingly, the systems in this section bear striking similarity. All of them are informed by phrase structure grammar—the idea that language has constituent structure—which reflects the notion that the generation process must pay close attention to grammatical rules because generation, after all, needs to produce grammatically correct output.

The second similarity among these systems is that they are all shaped, to a greater or lesser degree, by *systemic grammar* (Halliday 1976). Essentially, systemic grammar is defined in terms of features rather than syntactic categories. The features of a unit reflect its functions, from the grammatical ones to the rhetorical or pragmatic ones. Systemic grammar's emphasis on the functions of constructions and justified criteria for selecting from among choices make it conducive to natural language generation.

The Penman system (Mann 1983a, Mann 1983b, The Penman Project 1989), most closely follows the systemic paradigm, and is reviewed first. Next is a review of Mumble (McDonald & Pustejovsky 1985, Meteer *et al.* 1987), whose formalisms are also rooted in systemic grammar. Hovy's PAULINE (1987), though developed in a different milieu, bears striking similarity to Mumble. Its distinguishing characteristic is its *phrasal lexicon* that explicitly combines lexical and grammatical information in one representation.

All designers of surface structure realization components have two goals: constructing fast, efficient systems, with no exponential algorithms; and implementing a complete grammar. However, none has demonstrated, either through theoretical proof or implementation, that he has reached these goals. Though Penman has a much bigger grammar and lexicon than the others, it is still far from being able to generate the full range of expression of the English language.

From the point of view of a strategic component, there is special interest in the interfaces these generators make available to a text planner. Both Mumble and PAULINE provide a clean interface, though Mumble appears better able to support a system. Penman, with its emphasis on querying the underlying knowledge base, does not appear easy to communicate with, though some work on interfacing it to different knowledge systems has been done (for example, Sondheimer & Nebel 1986).

7.2.1 Penman

Nigel, the generation component of the Penman system (Mann 1983a, Mann 1983b, The Penman Project, 1989), is an implementation of systemic grammar which communicates with the knowledge base by querying it. Nigel's grammar is believed to be the largest and most robust of any generator.

The systemic framework for representing language emphasizes the functional aspects of grammatical phenomena, that is, how a grammatical entity can serve the goals of the speaker. There are three sorts of entities in this grammar: systems; realizations; and lexical items. Mann points out that systemic grammar is different from structural grammar (such at that provided by ATNs or context-free grammars) in that the systems do not specify the ordering of constituents in the output, but rather they specify *features* of the structures. By means of these features systemic grammar can accommodate and combine constraints of a variety of types, derived from linguistic functions concerned with such aspects as logical content, interpersonal attitudes and status, and emphasis and coherence of the text. The different sorts of information are represented uniformly by the features.

A *system* contains a set of alternative *grammatical features.* A system is *entered* under conditions specified in its *input expression* and exactly one feature is chosen. This feature is added to the *selection expression*, which is the set of all features chosen so far. Based on this expression,

realization operators that are associated with the features are invoked to build structure, to constrain order, and to associate features with functions (such as those resulting in the selection of lexical items) affected by the other two operations. Order constraints not specifically addressed by the realization operators are taken care of by defaults.

The grammar is organized into a network of systems, with a single entry point for the generation of any unit. When a system is entered, its *chooser* selects the appropriate choice based on the state of the system. However, not all the information that the chooser needs is directly available. The chooser must be able to query the *environment* outside of Nigel. The environment consists in the static knowledge base, the current text plan, and the *text services*, which are provided by the *function association table*. This table associates *hubs*—symbols recognizable to the choosers that represent entities in the environment—and grammatical functions. The choosers ask questions about, for example, the multiplicity and gender of objects so that these can be represented in the output. A chooser generally has the form of a tree; most of its questions have a predetermined set of answers so that a path may be traced to the appropriate choice. The *inquiry* the chooser presents to the environment combines an operator and parameters that are grammatical functions that are matched through the table with entities in the environment. The dialog of inquiries and environment responses explicates the grammar's assumptions and dependencies. Mann feels that the inquiry structure in the choosers allows for better handling of exceptional cases than in most generators.

Mann claims that Nigel makes no presumptions on the form of the underlying knowledge representation because Nigel is only interested in communicating through the arbitrary symbols in the function association table. Of course, *somebody* has to make sure the environment can also communicate through this table. Because of this interface, Nigel can generate from any representation that can cope with the table, and thus, Mann claims, Nigel is unlikely to reflect any hidden assumptions of the representation.

Nigel is embedded in Penman, a system that, based on communication goals, acquires information in a knowledge base and constructs a hierarchical text plan for organizing it. Nigel generates text according to this text plan; the text is then evaluated and revised, by altering the plan and regenerating one or more times, until no more need for improvement is detected.

Penman pays for its size in lack of speed; further all the systems and choosers must be hand-coded, making Nigel unwieldy at best to work with. Mann is well aware of the need for a robust and versatile interface between a generator and the system that uses it, but an architecture in which choosers make queries of the knowledge base seems to make the communication awkward. Nigel can generate from any underlying system because it forces the underlying system to conform to its representations. In theory, a text planner interposed between Nigel and the system could mediate the communication (for instance, by precompiling some of the most requested sorts of information, such as gender and number) so that the knowledge base need not worry about what Nigel wants. But this set-up makes the generator less of a useful tool and more of a system that needs to be catered to, which is a significant drawback.

7.2.2 Mumble

One of the oldest extant natural language generators, Mumble has gone through considerable evolution. McDonald and Pustejovsky (1985), describe the philosophy behind Mumble's most recent incarnation, Mumble-86 (Meteer *et al.* 1987). Of major significance is the streamlining and uniformity of Mumble's input specifications, designed to make interface to an arbitrary text planner feasible. There is also an ongoing concern in the research to reflect in the computational model a

plausible psycholinguistic one for how people produce language.

McDonald and Pustejovsky point out that generation is a process of decision-making, and the decisions are made on several levels. Two levels are domain-dependent, and are thus considered to lie in the province of the planner—choosing goals for expression, and then choosing information and rhetorical effects to achieve those goals. The third level belongs to the *linguistic component*, the generator proper: realizing those specifications in well-formed, integrated text.

Within Mumble itself, there are three levels of processing, *Attachment*, *Realization*, and *Phrase Structure Execution*, though it should be understood that processing at all the levels is *interleaved*—that is, Mumble's execution essentially consists in cycling through these processes. Each level has its own rules, representation, and reasoning processes, and its own restrictions on information and structures available. The lack of representational homogeneity is argued to contribute to the efficiency and flexibility of the design.

Mumble expects its input to be in the form of *realization specifications* (*rspecs*) that correspond to conceptual units in the underlying knowledge representation; these specifications typically result in clauses, but may also result in verb phrases, noun phrases, adjectives, etc. The major representation within Mumble is the *surface structure tree*, which is an abstract syntactic representation for phrase structure grammar; this linguistic structure has a controlling influence on the generator, because all of the processes of the generator operate on it. As Mumble builds a surface-structure tree it builds a node for each rspec and evaluates its arguments in left-to-right order, building a child node for each. (Note that the children in the tree may not have the same order as the arguments of the rspec: the tree order is determined by the choice selected from the rclass.) Because arguments may be rspecs, this process is recursive, stopping each time it reaches a word, which forms a leaf of the tree.

Attachment finds places (*attachment points*) in the surface structure tree for the incoming units. *Realization* of a unit happens after it has been attached. Conceptual units are associated with *realization classes* (*rclasses*), sets of parameterized and annotated *choices*. A choice is selected based on grammatical constraints, rhetorical goals, and the current state of the surface structure, and the arguments of the specification (for example, the arguments of a verb) are mapped to the pattern of the choice. If the arguments are themselves specifications, the process will be applied recursively. A choice is a tree fragment, and it is knit into the surface structure tree at the point chosen by Attachment. *Phrase Structure Execution* conducts a depth-first traversal of this tree. As PSE traverses the tree, it encounters attachment points, and calls on Attachment to see if they should be filled; it encounters units, and calls on Realization to realize them; and it encounters words, and calls on routines that output the morphologically specialized forms of the words. As PSE walks the tree, it visits each node both before and after visiting its children (in this way, constraints are propagated). In the current version of Mumble, the root of the tree is expected to be a clause, and the resulting text a sentence of English.

The current version of Mumble, Mumble-86, embodying as it does only the surface structure realization component of the generation process, is by design a rather sophisticated programming language. (Though one may be tempted to ask the utility of using a language that is an often unwieldy syntactic sugaring of Lisp.) Mumble has successfully been used as a surface structure realization component with other systems, including Gnomon (Forster 1989) and TEXT (Rubinoff 1986).

7.2.3 PAULINE

In his work on surface structure realization (Hovy 1987), Hovy's position is that language is composed of phrases, not easily divisible into words and syntax rules; given this, a realization program should not separate the grammar rules that a machine applies from the lexicon of words it uses to produce text. This does not mean that he eschews syntax, but instead he encodes syntax along with words in the *phrasal lexicon.* (Hovy cites Becker's Phrasal Lexicon (1974) as inspiration.)

Hovy's model of generation is embodied in a program called PAULINE (Planning And Uttering Language In Natural Environments) which plans different versions of the same text based on different pragmatic goals. According to Hovy, the generation task is composed of three parts: deciding what to *include* of the available information (much of this is actually done by the planner, which is assumed to have access to the speaker's goals); *ordering* the selected information, usually within a paragraph-sized piece of text; and *casting* it into the appropriate form for the output text.

The input to the generator is a list of language-independent structures collected by some sort of text planner. These units are either syntax goals, which create a syntactic environment corresponding to a phrase, or actual words. The generator will consider these structures in a left-to-right order, recursively replacing the goals with words and goals until the implicit tree being built by this process bottoms out into just words.

Hovy's implementation is guided by the general division of grammatical rules in English into two sorts: *formative*, which affect the order of constituents, that is, the form of the output; and *non-formative.* Formative rules specify the ordering of *environments* within environments; non-formative rules specify how relationships between environments are signalled. An environment corresponds to a phrase; thus as an example of the first rule type would be that the **article** precedes the **head noun** in a **noun group**. An example of the second rule type is number agreement between subject and verb.

In order to accomplish the three tasks of generation listed above PAULINE incorporates *syntax specialists*, each of which performs these tasks on a particular syntactic goal in the input and produces an ordered list of words and syntactic goals. A specialist takes into account three types of criteria: *syntactic*, which must be adhered to for grammatical results; *semantic*, which match features between input elements (for example, if the verbal relation is **INGEST** and the object has the feature **liquid**, then the verb should probably be 'drink'); and *pragmatic*, which often help the semantic criteria in the selection of appropriate words.

Non-formative rules operate across specialists and require the containing phrasal environment to keep track of information between phrases; for example, the sentence environment handles subject-verb number agreement between the noun phrase and the verb phrase. Specialists that take care of the general formative rules that create generic phrasal environments, rather than being attached to a particular word or phrase, work the same but are accessed indirectly. They too reside in the lexicon, and roughly correspond to phrase structure rules; examples are **say-relative-clause**, **say-subject**, **say-prep-group**. Thus, the entire grammar is contained in the lexicon.

The realizer, then, makes its inclusion decisions by choosing a goal's topic, its casting decisions by choosing specialists, and its ordering decisions by ordering goals.

Hovy argues that because a person's linguistic competence includes many phrases, and choices for expressing similar semantic content can have different syntactic properties (consider, for example, 'die,' 'kick the bucket,' and 'halt'), there is not a clear distinction between lexical and grammatical information. Further, for efficiency in the generation system, all this information should have a uniform representation, and should reside in the lexicon. However, Hovy has not

made the efficiency of his design clear. Further, the strength of his generator is handicapped by the confused design of his text planning component (see section 3.8.3).

7.3 Incremental Sentence Realization

A small group of researchers in Europe has taken a somewhat different approach to the problem of generating sentences. This research more heavily relies on results from psycholinguists than that described in the previous section. Levelt dubs this model of language production "from intention to articulation" in the subtitle to his book (Levelt 1989). This book provides the most extensive review of this many-layered model, in which the processes at all the levels work in parallel, with their results cascading, in *increments* to the next lower level for processing there. Part of this approach has focused on the level of text generation comparable to that addressed by the systems described in this chapter; this approach is summarized in (De Smedt 1990), and includes work by Kempen and Hoenkamp (1987). The title of De Smedt's thesis, *Incremental Sentence Generation* illustrates both its strength and its weakness. The strength lies in the realization that people produce text in increments, not full sentences, and the model accounts for some of the mistakes people make as a result of constructing sentences incrementally, as well as capturing much of the grammar of well-formed sentences. (The system generates Dutch.) The weakness of the work lies in not modeling where the choices that specify the form of the generated increments are made, that is, in simply assuming a strategic component. This results in a system that is capable of handling the surface structure realization of increments that it receives out of order or that do not fit together. But without any strategic component making demands on the incremental sentence generator that might result in increments out of order, this capability is unmotivated. However, addressing the problem of incremental generation, even if only at the sentence level represents a real advance in the field.

The process of surface structure realization in De Smedt's system starts with a syntax tree. Three types of operation may be performed on this tree, as new increments come in and need to be integrated: *expansion*, *coordination*, and *correction*. Expansion is the process that grows the syntax tree. Expansion may be *upward*, in which case the original root node is replaced by a new root that dominates the old root and the new material. *Downward* expansion simply adds a new node below an existing node. *Insertion* adds nodes between existing ones. Insertion may or may not result in self-correction; an example of where it does is the following:

"John wants an apple...wants to eat an apple."

Coordination is the process of iterating over several increments, of any size, that form a conjunction (or disjunction), and relating them together with commas and conjuncts such as "and." *Self correction* is a process that decides how much of the sentence needs to be regenerated after insertion into a part that has already been produced.

This theory has been implemented twice, once using parallel processing and once using an object-oriented system. (This parallel approach bears some similarities to Word Expert Parsing (Small & Rieger 1982).) The basic underlying mechanism is functional unification. Roughly speaking, unification is of two functional descriptions, one of which is of semantic content and the other of which is of a fragment of the grammar for the language. If the two descriptions can be unified, then that content can be expressed via that grammar fragment. Functional Unification Grammar (FUG) was developed by Kay (1979), and is similar to Lexical Functional Grammar (Kaplan & Bresnan 1982). FUG has been used as the grammatical formalism for a number of other surface structure components, such as TEXT (McKeown 1985) and Epicure (Dale 1988).

7.4 Salix's Surface Structure Realization

This section describes the surface structure realization capability of Salix.

There are about 25 realization functions, each responsible for handling a particular sort of text increment. These functions are generally passed a node, and perhaps some parameters, and they take care of the "nitty gritty" of choosing the correct surface form. Choices include selecting the appropriate gender and number of pronouns and the appropriate number of verbs. Some realization functions regularly make random choices. In many cases, different realizations of an expression are available and regularly used (e.g., "and" and "then" when introducing the next item in a series). Since on the one hand text is better when the full variety of expression is used and on the other hand there is no principled way to choose between the alternatives each time a selection is needed, nondeterministic choice is the most effective solution. Occasionally, it does not matter which choice is made, but it is then advisable to ensure that subsequent choices are made the same way (see section 7.4.5).

7.4.1 Primitives

The most primitive function is **realize**, which is a string output function. All the other **realize-** functions call it. There are two auxiliary functions, **increment-start** and **increment-end**, which ensure that an increment of text occupies its own line of output. This typographic convention serves to inform the reader of the text as to how the increments are delimited.

7.4.2 Cue Phrases

Some phrases are used to signal discourse moves (Reichman 1985, Grosz & Sidner 1986). Such signals include ones that mark when a speaker starts her turn, when she finishes, when she changes the topic, and when she makes a metacomment about what she has or has not said.

realize-beginning, which is called by **say-first-object**, produces an increment such one as the following:

```
OK
all right
```

realize-conclusion is called by the null metastrategy **say-all-done** when there is nothing left to say, and produces something like:

```
OK
that's it
```

realize-oops is called by the null metastrategy **jump** when no strategies have matched but the completeness criterion has not been met. It produces a member of the `*jump-strings*` for the domain. The family domain does not have any jump strings; the house domain has ones that include:

```
wait I forgot
oh yeah
```

realize-intro-cue is called to signal some change in the flow of the text. For example, it is called at the beginning of **say-multiple-sweep** to indicate that a new topic is coming up and that there will be several increments of text generated about it. **realize-intro-cue** produces the text:

```
then
```

7.4.3 Introductions

The default text for **say-object** (e.g., "there's a window") has the form

```
<intro> <object>
```

and for **say-property** (e.g., "it is large")

```
<intro> <property>
```

say-object calls the function **realize-intro** to produce the appropriate introduction for the object. (See section 7.4.4 for how text for the object is realized.) What is appropriate depends on context: whether the last increment of text was produced by **say-object** or by some other strategy. If the last increment was also produced by **say-object**, then a brief introduction is all that is necessary, since the context for the current increment is simply one of mentioning objects, and this is just another mention like those before it. If, however, the last increment of text was of some other type, a full introduction is needed.

Therefore, in the fragment

```
(1) then there's a window
(2) which is large
(3) then there's a sliding glass door
(4) and a window
```

the intro of increment (3) must be a full intro, since increment (2) was not produced by **say-object**, whereas the intro of increment (4) is a brief intro, since it follows a **say-object** increment.

The choices for full intros include

```
and then there's
then there's
```

and for brief intros include

```
then
and
```

Any strategy can specify which intro it requires by using a keyword.

The case for **realize-property-intro** is similar. In Salix, properties are always expressed clausally, following the noun they modify (e.g., "the window which is large").[1] If the preceding increment was produced by **say-property**, the brief intro

[1] Prenominal adjectives were not implemented in Salix. It would have been perfectly easy to do so. Phrases such as "sliding glass door" were judged to be noncompositional names and were recorded as such in the knowledge base.

```
and is
```

is used, otherwise the full intro

```
which is
```

is used. Use of the full intro can be seen in increment (2) and the brief intro in increment (3).

```
(1) there is a window
(2) which is large
(3) and is a picture window
```

realize-first-object-intro is used in conjunction with **realize-object-name** by **say-first-object** to mention the first object in a text, which is often described differently from the rest. If the first object has an associated preposition, then the intro will use that. For example, if the first object is side-door, its associated preposition is "at," and the first object can be introduced by

```
we can start at the side door
```

(The choice of pronoun here is as described in section 7.4.5.)

Otherwise, the first object can be introduced with something like

```
first there's
```

7.4.4 Noun Phrases

There are several functions that produce noun phrases for objects. All of them take a KB node as an argument. Usually the *noun-sing-form* of the *lex-item* of the node is used; in some cases, the node's *proper-name* may be used.

The distinctions among the realize-object functions determine what sort of determiner, if any, is included in the noun phrase.

realize-proper-name produces a proper name with no determiner, like

```
Penni
```

realize-object-plain produces the name of a node with no determiner. This function is used when the node is marked as having special (possibly idiosyncratic) properties, which indicates using the text included with the node without modifying it. This function is used, for example, to realize the orientation in the house domain which is expressed by

```
and if we're facing the backyard
```

The form of "the backyard" in this increment is that of a definite noun phrase. However, "the backyard" is actually being used as a reference point for orientation, exactly as "north" or "back the way that we came" would be. In this context "the backyard" is functioning *in toto* as an adverbial phrase, and is not treated like noun phrases that refer to objects.

realize-object-name produces a noun phrase with a definite article, such as

```
the window
```

realize-object-type produces a noun phrase with an indefinite article, such as

```
a window
```

Both functions check to see whether the object in question participates in a has-owner relationship. If so, the possessive form of the owner's name is used instead of an article, as in

```
Penni's bedroom
```

As usual, either of these functions may be called directly. There is also a general function, **realize-object**, which nondeterministically chooses between **realize-object-type** and **realize-object-name**. **realize-object** is called by **say-object** when the *realization mode* (section 7.4.7) is *default*; this has the result that unless there are overriding reasons in particular cases, about half of all objects are mentioned in definite noun phrases and half in indefinite noun phrases. While this may seem counterintuitive, since much emphasis has traditionally been placed on the implications of using these two different types of noun phrases, it reflects the usage in my data, as discussed in section 2.1.5.

realize-object-name-again is used to indicate that an object has already been said and is being referred to again; the determiner "that" is used. **realize-object-name-again** is called by **say-multiple-sweep** to indicate that the same object is being oriented toward.

```
in the kitchen
there is a window
....
and if we're facing that window
```

7.4.5 Pronouns and Possessives

The pronoun capabilities of Salix cover the full range of third person pronouns. First and second person pronouns are not implemented as such because there is not a general theory in Salix for their usage. (There is one specific usage of a first or second person pronoun, discussed in section 7.4.5.) However, the pronoun realization functions could trivially be extended to choose the correct form of first and second pronouns as well.

realize-pronoun takes a node as an argument (from which it can determine gender) and a set of keywords which indicate whether or not the pronoun should be plural, relative, objective, or possessive. It makes this selection according to the criteria laid out in figure 7.1.

In addition to being able to produce possessive pronouns, Salix can produce possessive noun phrases like the following, in which "Penni's" and "the room's" are the definite determiners.

```
Penni's bedroom
the room's closet
```

realize-possessive-expression is a general function which takes a node as an argument and can call **realize-proper-possessive** if the node has a proper name or **realize-definite-possessive**

Criteria		Pronoun
relative		which
	animate	who
	possessive	whose
plural		they
	objective	them
	possessive	their
female		she
	objective	her
	possessive	her
male		he
	objective	him
	possessive	his
	possessive	it
		its

Figure 7.1: Pronoun choice criteria.

if it does not. As a default, **realize-possessive-expression** nondeterministically selects between a noun phrase possessive and a pronoun possessive, though it can be directed to select either.

There are two other functions in Salix which handle the production of pronouns. **realize-anaphor-for-current-node** calls **realize-pronoun** with the current node as argument. The number of the pronoun is determined by context. The current node is the knowledge base node which the current increment is about, and this increment normally expresses a relation between the current node and another node or nodes. If the last said node (the node that most recently appeared in the text) bears the same relation to these other nodes, then a plural pronoun is selected. This is illustrated in the following examples; the first does not use the plural pronoun and the second does.

```
first there's Penni
her sister is Barbara
her parents are Kitsy and John

first there's Penni
her sister is Barbara
their parents are Kitsy and John
```

The final sort of pronoun is determined by the function **realize-non-referential-pronoun**. This is the pronoun used in expressions like

```
if you're facing the backyard
we can start at the side door
```

In this usage, the pronoun of choice is either "you" or "we." Such a pronoun is not referential in the way implicit in the pronoun capability described above. That is, when we say "her sister

is Barbara" or "it faces the backyard," "her" and "it" refer to something in particular, which is determined by the context. However, when we say "if we're facing the backyard," we are not referring to anything in particular by "we," but instead using a familiar locution which happens to have the pronoun "we" embedded in it.[2] That Salix can produce nonreferential "we" and "you" does not interfere with its being able to produce referential "we" and "you." There is simply more than one source for such pronouns.

realize-nonreferential-pronoun is a "sticky" function. Whether to use "you" or "we" is selected nondeterministically, but once the choice is made it is saved for the duration of the text, becoming part of the context.

7.4.6 Deictic Expressions

Salix is capable of producing spatial deictic expressions, such as "on the left" and "underneath." (For a fuller discussion of spatial deixis see section 5.2.1). Deictic expressions, whether they are spatial, temporal, or of any other sort, are contextual: they are always used in reference to some reference frame, which is dynamically determined. This reference frame is usually relative to the speaker or the hearer or both, and can be thought of as an imaginary person. In a spatial domain, such as that of a house, the reference frame has speaker position, which corresponds to some part of the domain, and an orientation (trajectory) which indicates which spatial direction is forwards from the reference frame. This information is used to compute the appropriate spatial deictic terms. The realization mode *spatial-deixis-as-pp+be* requires using a deictic term if possible; otherwise the *default* is used (see section 7.4.7).

The following two fragments are about the same part of the house; the first uses the deictic term "on the left," the second does not.

```
and then there's a short hallway
and
Lisa's bedroom on the left

and then there's a short hallway
then Lisa's bedroom
```

realize-deictic is called when a deictic expression is required. It is passed a node and it compares the node to the reference frame. If, as shown in figure 7.2, it can determine the position of the node relative to the reference frame, it produces the corresponding deictic term. If the relationship cannot be determined, nothing is produced.

Notice that if the deictic term is being produced within the scope of the metastrategy **say-left-right-and-center**, and if the term is to express a forwards relationship, the specialized term "in the middle" or "in the center" is used.

7.4.7 Realizing Objects According to Relation Modes

As described in section 3.3.1, the strategy **say-object** is responsible for increments of text whose general pattern is:

[2]Probably scores of philosophers of language will disagree with my analysis here. I do not wish to argue the point, but merely present a mode of usage within the implementation that is consistent with the data I have studied.

Relationship	Deictic term
to the left	`on the left`
to the right	`on the right`
forwards	`in front`
(left, right, and center)	`in the middle`
below	`underneath`

Figure 7.2: Deictic choice criteria.

```
<current-object> <current-relation> <related-objects>
```

The particular form of the increment depends on the *realization-mode* of the current relation. The function **realize-object-according-to-realization-mode** selects the appropriate realization.

A relation can usually be expressed by a verb, as with face in (1); this is the realization mode *verb*. Sometimes there may be text in addition to the verb, such as the prepositional phrase in (2) and (3); this is the realization mode *verb+pp*. The placement of the prepositional phrase is selected nondeterministically, unless there are more than three related objects, in which case it must be directly following the verb. A relation can also be expressed by a noun followed by a copula (form of "be") (4); this is the realization mode *noun+be*. See figure 7.3.

```
    <current-object> <relation>   <related-objects>
    ---------------- -----------  -----------------
(1) the window       faces        the backyard
(2) the livingroom   has          a window           in it
(3) the livingroom   has in it    a window
(4) her              brothers are Jeffrey and Andrew
```

Figure 7.3: Relation modes.

If the relation is unknown, then the realization mode is *there-insertion* (5). Finally a relation can be expressible by spatial deixis (see section 5.2.1); this realization mode is *spatial-deixis-as-pp+be*, for which there are several variant realizations (6) and (7). If spatial deixis is not enabled, this mode defaults to the *default* realization mode (7).

```
(5) then there's the livingroom
(6) Ann's room is on the left
(7) on the left is Ann's room
(8) then there's Ann's room
```

When there is more than one related object, there is an option to insert "and" (using **realize-and**), between the realizations of all the objects. This choice is made nondeterministically; there is always a **realize-and** before the last object.

The realization of the current object is subject to *focus* conditions (see section 4.3); **realize-anaphor-for-current-node** is the function called.

To realize the relation, the functions **realize-relation-noun**, **realize-relation-verb**, and **realize-relation-pp** take a relation node as argument and produce the required form, from the lex-item associated with the realization. Keywords may specify whether the form should be plural; the default is for it to be singular.

7.4.8 Other Functionality

realize-copula is called to produce the appropriate form of the verb "to be." By default, it produces "is." If the information is available that the copula will be followed by more than one application of **say-object**, for example, within the scope of **say-set-relation**, then **realize-copula** will generate "are."

realize-and is completely self explanatory.

realize-orientation produces the orientation phrase

```
if we're facing
if you're facing
```

The pronoun is determined by **realize-nonreferential-pronoun** (see section 7.4.5).

Chapter 8

Evaluation

In this chapter, I describe a methodology for evaluating text that is generated by Salix, and present preliminary findings of a study using this methodology. I also discuss "structure" in the context of generating text. Finally, I describe some future directions of the work presented in this thesis, particularly in regards to extending the current implementation to more fully account for the data in the house and family domains.

8.1 Evaluation

As pointed out in chapter 1, there is no "gold standard" for evaluating text generation systems. That is, there is no suite of tests or set of benchmarks for generation systems, and little basis for direct comparison between systems. This is in part because the problem is still being defined; this thesis is part of that process of definition. Another reason is that different domains inspire different approaches; if researchers were restricted to generating a small fixed set of texts, their ability and incentive to explore broader issues in generation would be curtailed.

I cannot compare Salix's size and speed with that of other systems, but I can give some statistics about Salix. The entire system is about 3700 lines of code. It is implemented in Common Lisp. It has been tested mostly on a terminal remotely logged in to a Sun workstation running Unix; when Salix is compiled and run in this environment, it generates output text at faster than reading speed.

The base system of Salix comprises three finding strategies, four saying strategies, one metastrategy, and three null strategies. The finding strategies are: **find-next-to-say**; **find-salient-object**; and **find-unsaid-to-say** (currently under implementation). The saying strategies are: **say-object**; **say-first-object**; **say-elaborating-relation**; and **say-property**. The metastrategy is **choose** and the null strategies are **noop-on-inhibit-recursion**, **jump**, and **say-all-done**. For spatial domains, there are an additional two metastrategies: **say-left-right-and-center** and **say-multiple-sweep**. There are about twenty-five **realize** functions that accomplish the surface structure realization of the text.

While I cannot compare Salix's output with the output of other text generation systems, I can compare different instances of Salix's output with each other and with the human generated text that inspired it. To assess the contributions of different components of Salix's process, these components were selectively disabled before texts were generated. The rest of this section describes the results of asking people to evaluate sets of Salix's texts.

8.1.1 The Evaluation Task

In order to evaluate Salix's output, I created an evaluation packet containing: five texts in the house domain; five or six texts in the family domain; the sketch of the house (figure 1.2); the sketch of the family (figure B.1); and a sheet for recording evaluations. This sheet included the following instructions:

> **Directions:** This packet contains two sets of texts, **A** and **B**. Each comes with an accompanying diagram, which may or may not be useful in understanding the texts. I'd like you to read each set of texts, and rate each text on *how good it is*. That is, does it make sense? Does it sound stupid? Is it tedious? Is it badly organized? Evaluate the text by whatever criteria you think appropriate. Then mark on the scale how you rate the text. If you can say (briefly) why, write that down too. Finally, for each set of texts, pick the one that you think is best.

For each text, there was a scale like this one:

```
1----------2----------3----------4----------5

OK                                          not OK
```

I distributed this packet to seven people I know at Xerox Palo Alto Research Center. The familiarity of these evaluators with my research ranged from a fair amount (my officemate) to none (my bridge partner).

The texts generated by Salix included in the packet were either produced by Salix with its full capabilities, or with a single component disabled. The components disabled included the inference rules, the elaborating strategies, and the spatial metastrategies (in the house domain).

As is always the case with eliciting data from people, the data are both fascinating and unexpected. I will examine the evaluation of the various texts that is relevant to the distinctions among them; and then summarize briefly the sorts of feedback I got that I was not looking for.

8.1.2 Testing Components in the House Domain

Evaluators were given five texts that were generated by Salix in the house domain. One text was one of Salix's best efforts, similar to the text in figure 1.1. Another text was an example of when Salix gets confused and jumps around a lot. For the generation of each of the other three texts, a single component was disabled. These components were the inference mechanism, the spatial metastrategies, and the elaborating relations.

Interestingly, when evaluators were asked to pick what they thought was the best text, there was no consensus, and several of the evaluators could not decide between two or more texts.

The Vagaries of Local Organization

Recall that there is an element of nondeterminism in all of Salix's texts. Some nondeterministic choices, such as that between "the" and "a," do not have any impact on subsequent text construction. Other nondeterministic choices, however, such as selecting a **find-next-to-say** application,

can affect how the rest of the text is structured. In particular, some choices can lead Salix into a dead end, in which it has to apply **jump** to find the next thing to say, signaling this by "oh yeah" or "wait I forgot." A string of bad choices can result in text replete with **jumps** and as a result very disorganized, since the domain structure is no longer structuring the text. Salix rarely produces texts like this.

I simulated such excessively forgetful behavior in Salix by making it dead end and need to jump almost immediately after starting the text. As one would expect, this text was flagged by the evaluators as being uncoordinated and rather stupid sounding. As one evaluator put it, "How can you forget most of the house?"

Because Salix's text generation is locally organized, Salix cannot "look ahead" and see that it will paint itself into a corner. Salix does have a way, jumping, to get itself out of such a corner, but there might be instances where anticipation of a dead end could lead to better text, either by avoiding that dead end or by signaling it beforehand as well as after the fact. Such a look ahead capability might be incorporated into Salix to insure against such problems. However, it is not clear how useful such machinery would be, since when the KB is properly encoded the problem rarely occurs.

Spatial Metastrategies

Disabling the spatial metastrategies results in texts that lack both the more elaborate organization afforded by the metastrategies and the use of deictic terms that results from the fact that the metastrategies are spatial. Instead the text consists of mentioning objects, the fact that they are next to each other, and elaborations on them. One thing shows up in the text that usually does not when **say-multiple-sweep** is available: **say-elaborating-relation** on the kitchen with the relation contains. This results in the inclusion of the following text for the kitchen:

```
and a kitchen
it has
in it
the window
which is a picture window
and is large
and has two flanking windows
and faces the backyard
and a refrigerator and a stove and a sink and a dishwasher and a window
which is small
and a sliding glass door
```

Certainly, enumerating the contents of a room like this is less informative than the text provided by **say-multiple-sweep**, and this was noted by some of the evaluators.

Inference Rules

The only inference rule currently implemented that has an effect in the house domain is the one that says that if an object is realized by **say-object** and marked as *already-said?*, and that object is contained by another object, the contains relation is marked as *already-used?* (see section 3.5). When a **say-multiple-sweep** applies to the kitchen, all the objects in the kitchen are marked as

already-said?, and that they are contained by the kitchen is marked as *already-used?*; this prevents a **say-elaborating-relation** from matching on the kitchen with the relation contains and these objects, as happened in the preceding section. When the inference mechanism is disabled, however, the resulting text contains a multiple sweep and an application of **say-elaborating-relation** with contains, effectively describing the kitchen twice. This repetitiveness was noted as a misfeature by most of the evaluators.

Elaborating Strategies

One of the house texts was generated with the elaborating strategies disabled, which was accomplished simply by removing **say-elaborating-relation** and **say-property** from **choose**'s strategy preference list. This resulted in a text which only included how things were spatially related, including by use of the spatial metastrategies.

None of the evaluators seemed to notice that elaborations in general were missing. However, one elaboration in particular turned out to crucially interact with the spatial metastrategy **say-multiple-sweep**. The offending bit of text is reproduced below:

```
then
in the kitchen
there is a window
if we're facing the backyard
on the right is the sliding glass door
```

Because **say-elaborating-relation** was not available to realize "[the window] faces the backyard," the metastrategy's use of the orienting phrase "if we're facing the backyard" is confusing.

This interaction suggests perhaps that either the strategies should be redesigned to avoid the dependency of **say-multiple-sweep** on **say-elaborating-relation** or that spatial metastrategies should not be enabled unless elaborating ones are. A more satisfactory solution, however, that would remain consistent with the data, might be to expand Salix's context to allow **say-multiple-sweep** to check whether the objects to which it will refer for orientation have been recently mentioned.

8.1.3 Testing Components in the Family Domain

The set of family texts to be evaluated included: a text produced by Salix's full capabilities (figure 4.2); a text with all elaborating relations removed; a text with half the elaborating relations removed; a text for which the gender inference rules were disabled; a text for which the relation inference rules were disabled; and a slightly edited version of one of the person-generated texts from my original corpus (figure 2.7). Some evaluators did not receive this last text.

Inference Rules

I disabled the inference rules (section 3.5) in two sets, under the assumption that text without the gender inference rules is so horrible that no one would notice what else is wrong with it. This is an example of text in the family domain with the following rule disabled: if an object is realized by its proper name, the link to its gender property should be marked as *already-used?*.

```
first there's Penni
```

```
Penni's parents are Kitsy
who is female
and John
who is male
Penni's sister is Barbara
who is female
and is female
```

The last "and is female" is an elaboration on Penni. Not surprisingly, evaluators found the mention of gender superfluous, if not downright funny.

The other set of inference rules relevant to the family domain are those which mark inverse relations as *already-used?*, suppressing texts of the form "Penni's father is John and John's daughter is Penni." When these inference rules are disabled, the effects may be more subtle than in this example: several increments may appear between "Penni's father is John" and "John's daughter is Penni," and among the welter of relationships being mentioned, such redundancy may be missed. However, in an entire text, enough of this redundancy was evident that over half the evaluators noted the repetition.

Elaborating Strategies

I took two tacks with removing elaborating relations. If they are disabled across the board (by removing **say-elaborating-relation** from **choose**'s strategy preference), the text looks like this:

```
then there's Jeff
then there's Vi
and then there's Alison
then there's Andrew
then there's Jeffrey
```

Evaluators found this text "tedious" and "BORING."

To generate a different text, I removed half the elaborating relations from the list of elaborating relations. This resulted in a text with a somewhat disjointed mixture of says and finds:

```
then there's Jonathan
Jonathan's brothers are Andrew and Benjamin
then there's Martha
her parents are Dorothy and Jim
then there's Eleanor
```

While the evaluators were not particularly excited by this text, none of them indicated dissatisfaction that was particular to the effects of making the text more disjointed in this way.

The Real Thing

For the texts in the family domain, there was much more agreement than in the house domain on which text was the best, and the consensus was for the text generated by Salix with nothing disabled.

Some of the evaluators were also given the text in figure 2.7, edited to remove text that was not directly relevant to family relations and thus not comparable to that produced by Salix. When this text was included, it was chosen as the best text, because it made better use of pronouns and had more surface structure variety. One evaluator, who guessed that this text was generated by a person, decided that Salix's best text was generated by a person too.

8.1.4 Other Feedback

Some of the feedback included criticisms that may be legitimate but were not to the point. For example, one evaluator complained that a description was a "run-on sentence," presumably because of the lack of punctuation. Several evaluators did not like "a window which is large." They are quite right that "a large window" is better; the reason Salix realizes adjectives this way is discussed in section 7.4.3. There were several complaints about the increments "if we're facing the backyard" and "if we're facing that window." Evaluators seemed to feel that the texts were better *without* these phrases. This surprised me, since those locutions, produced by **say-multiple-sweep**, directly reflect the text given to me by one of my speakers (see A.1.2). Finally, some evaluators mistook the purpose of the included sketches, and thought the texts were meant to be descriptions of the sketches. This resulted in comments like, "main problem is that it doesn't mention Bub's Bar-B-Q and the party house."

8.2 Structure

Because Salix exploits domain structure to build text structure, it is legitimate to ask if there are characteristics of the structure of a domain that make it more or less amenable to the approach in Salix. This question can be asked in general—what is it that Salix can or cannot do, and by comparison with other text generation systems, particular ones that use global hierarchical structure to structure text. The answer is that it is not characteristics of a domain but characteristics of a knowledge base representing the domain that determine whether Salix's local organization approach can build reasonably structured texts.

To see that the structure of the domain is not what is crucial, consider the following thought experiment. Suppose that we want to generate a text about any domain at all, say, a description of the political events leading up to the war in the Persian Gulf. We need to encode this knowledge in a KB. We could encode the KB as an unbranching chain, in which the nodes are in exactly the order they need to be in so that generating text while traversing this chain would produce a perfect text of the domain. This would not be a very interesting or versatile KB, but it could be written for any domain. If there were a domain that such a data structure could not be written for, then no text generator, regardless of its methodology, could generate text for that domain.

Since it is the structure of the KB that is at issue, we need to make clear what it is about the structure that is important. Salix depends on being able to distinguish the basic relations in the KB; the basic relations are what allow Salix to find a next thing to talk about. It is finding each next thing that structures the text. Therefore, the KB needs to meet these criteria:

1. The *basic relations* of the domain need to be *explicit.* This can be accomplished by directly encoding them in the KB, or by providing a mechanism for inferring them. An appropriate inference mechanism may explicate basic relations that were not originally encoded in the KB. The basic relations make "local" in the KB things that need to be local in the text.

That is, the basic relations connect things that should appear next to each other in the text; for them to appear next to each other in the text, they must be next to each other in the KB.

2. The local *context* in the KB must be sufficient to allow a choice, from all the things available to say, a most appropriate next thing to say.

Some KBs may not meet these criteria. To generate text from such KBs might require a mechanism like a hierarchical planner or a schema. Such mechanisms can maintain state and can search the KB arbitrarily far, in effect making local to each other things which are not local in the KB, and thus allowing them to be expressed together. This thesis has not addressed the issue of generating text from KBs with which such mechanisms may be required; instead, the work presented here has focused on generating text from KBs that meet the two criteria listed above.

8.3 Structure and Domain

As I have emphasized, most of Salix's strategies are domain independent. Two metastrategies used in the house domain, **say-multiple-sweep** and **say-left-right-and-center**, are not domain independent. This does not imply, however, that they are domain *dependent.* That is, it is not the case that these strategies are only appropriate for building text in the house domain. Instead, these two metastrategies are appropriate to any *spatial* domain. Other examples of spatial domains include circuit layouts, towns, and maps. Not only can the domain independent portions of Salix's architecture be easily imported to such domains, but the spatial metastrategies can be as well, supplying an even more comprehensive text generating capability.

8.4 Future Directions

In this section I will outline some phenomena in the house and family domains that the current implementation of Salix does not account for, and in some cases sketch extensions to Salix's capabilities. These extensions would allow Salix to produce better texts, and ones that are truer to the data. They would also in general increase the capability and flexibility of Salix's text generation capacity. These extensions are in the areas of: realizing properties as adjectives; modeling *paths* in the house domain; assigning labels to nondiscreet features of a knowledge base; expressing arbitrary family relationships; .... and referring to text that has already been produced.

8.4.1 Realizing Properties

Salix realizes properties via applications of the strategy **say-property**. **say-property** is an elaborating strategy, which means that in general **say-property** is only applied after **say-object** (or **say-first-object**). This results in text like this:

```
there's a window
which is large
and is a picture window
```

However, the actual text produced by a person, and which surely sounds more natural is:[1]

[1]Let us assume for the sake of argument that "big" and "large" are interchangeable.

well there's....a squared-off area where there's one thin hallway
leads from the kitchen into the living room
then there's a wider one
that leads from....the kitchen into the living room
....about three times as wide

Figure 8.1: Distinguishing halls with descriptive labels.

"there's a big picture window"

In this text, the property of largeness is expressed as a prenominal adjective, that is, an adjective that comes before the noun, "window," that is used to express the object large-kitchen-window. The property of picture-window-ness is expressed by the prenominal adjective "picture."[2] For Salix to be able to express properties as prenominal adjectives, it needs to be able to decide to do so *before* applying **say-object**. This can be achieved by a domain independent metastrategy, **say-object-with-properties**. This metastrategy would be triggered when a **say-object** and one or more **say-property** applications are returned. In the current example, **say-object-with-properties** would be triggered by the set of applications comprising **say-object** with large-kitchen-window, **say-property** with large-kitchen-window and largeness, and **say-property** with large-kitchen-window and picture-window-ness. If applied, the application of **say-object-with-properties** would order the realization of the appropriate adjectives between the introduction ("there's a") and the noun for the object ("window").

The one difficulty in this approach is actually ordering the prenominal adjectives: "picture large window" just would not do.[3] This may be an extreme case, but in general there are conventions for the ordering of multiple prenominal adjectives. For instance, "two small green chairs" sounds better than "small two green chairs" or "two green small chairs." While on the one hand we do not want Salix to randomly order prenominal adjectives that realize properties that to Salix are indistinguishable, on the other hand we cannot just declare that properties realized as prenominal adjectives must always be ordered according to an arbitrary criterion. There are certainly contexts in which "two green small chairs" is the appropriate thing to say. However, I do not have a theory for getting around this problem in Salix.

8.4.2 Assigning Labels

The current implementation of Salix has finessed the issue of lexical choice. *Lexical choice* is the process of choosing the appropriate lexical item (word or phrase). This process is difficult along several dimensions. For instance, degree of *specificity* may matter, e.g., between "parent" and "mother" in the family domain. In the house domain, specificity is rarely an issue: speakers very reliably call a stove a "stove" and not an "appliance." Another dimension of lexical choice is the affect that may be encoded. For example, "cat" encodes little affect, "magnificent beast" (arguably) encodes positive affect, and "unmitigated monster" encodes negative affect. There were few examples of such affect encoding in either the house or the family corpus. In general, lexical choice issues of specificity and affect were infrequent in the corpora I studied.

[2] Or perhaps "window" is replaced by "picture window."

[3] This observation argues that *replacing* "window" with "picture window" is the appropriate way to realize the combination of **say-object** with large-kitchen-window and **say-property** with large-kitchen-window and picture-window-ness.

how should we describe this
....the hallway to the bedrooms
....in relationship to that wide hallway between the living room and kitchen
is like a T
there's a short bit
....and then there's a top of the T

um going from the living room toward the bathroom
there's a T-shaped hallway

Figure 8.2: Two speakers construct a label for the "T-shaped" hallway.

Two other lexical choice issues do show up in the house domain: distinguishing objects with the same or similar labels, and constructing labels for objects that are not already individuated and labeled in the KB. Hallways are a good example in the house domain of objects which have the same default labels (e.g., "hall" or "hallway") but that speakers typically try to distinguish with more descriptive labels, as can be seen in figure 8.1. As the example shows, in the house domain, where most objects are physical objects, distinguishing characteristics are often ones of relative size. In the current implementation of Salix, these relative sizes are "hardwired" in, such that the lex-item associated with each object carries distinguishing information, such as in "short hallway." A more sophisticated version of Salix might try to compute these labels dynamically, for instance by searching the KB for other instances of this type of object in order to construct an appropriate distinguishing label. However, the data do not suggest a good approach to this problem; it will be left to future research.

Hallways also provide good examples of constructed descriptions in the house data. For example, at least two speakers constructed "T-shaped hallway" labels in their house descriptions (see figure 8.2, and see figure 1.2 for the sketch of the house). Not all the speakers referred to this configuration of two hallways as "T-shaped"; instead, the two hallways are often referred to individually (including by the speakers who refer to them collectively as a "T-shaped hallway"). This indicated that the hallways must be encoded as separate objects, and some other mechanism must be responsible for recognizing whatever it is about them that allows the collective label. Incorporating such a mechanism in Salix is a desideratum, but I believe more data need to be examined to try to understand how people make such labeling decisions.

8.4.3 Paths

Probably the most obvious gap in Salix's model of generation in the house domain is its inability to construct tour-like texts. Both Linde (1974) and I found frequent examples of tour-like descriptions in our data. An example from my data can be seen in figure 8.3. I did not model this type of text in Salix because I could get complete texts without modeling tour-like behavior. This functionality can be added to Salix, however, by introducing the notion of *paths*.

In a house, a path can be one of two things. Some objects in a house are naturally paths because one of their functions typically is to enable people to get from one place to another place or places; all hallways are of both type room and type path. Paths can also be constructed by means of an imaginary tour that goes from room to room. The first type of path is more common (it is what is being used by the speaker in figure 8.3, although she does not mention the hallways

....and then diagonally up to your right
....as you go past the kitchen
is one of the bedrooms
....ah you go around the chimney to your left and
....in the front corner of the house is the living room
....and if you keep going to your—as you go around to the left
if you go right there's a closet to your left

Figure 8.3: Example fragment of tour-like house description.

explicitly), and can be modeled in Salix with the addition of a **find-path** strategy. This strategy will match when an object is next-to the current object, is not *already-said?*, and is of type *path*; further, this object must be next-to another object that is not *already-said?* (this object is the destination of the path). All three objects are included in the application. The path itself may be realized explicitly, as in

"if you came in through the garage and walked across the walkway
you would go in the side door"

or it may be realized inexplicitly as in

"if you go right there's a closet to your left"

Because the path specifies the *trajectory* between the first object and the second object, deictic expressions can be used (see section 5.2.1). The second object becomes the new current object.

8.4.4 Arbitrary Family Relationships

This section discusses how Salix can be enhanced to express more family relationships, and gives two reasons—making the variety in Salix's texts closer to that in people's and making Salix's texts easier to follow—for doing so.

In the family domain, Salix currently expresses only *direct* relations, e.g., "Penni's father is John." However, people are certainly capable of talking about grandparents and cousins and other relationships that are not direct. On the other hand, people are not good at coming up with *arbitrary* relationships, but not very quickly (see figure 8.4). It is usually not the case that people do not have enough information available for them to figure out a relationship (this can be seen from examining their entire family descriptions), but rather that they do not have the relationship readily available. There is a middle ground, and Salix can be modified to occupy it. This can be achieved by giving Salix a *limited* ability to compute relations of up to three degrees.[4]

Salix has all the information it needs to compute relationships, and it can easily keep track of the degree of a relationship it computes. Given two family member nodes in the KB, fm-1 and fm-2, Salix can conduct a breadth-first search, to the depth of three, from fm-1. If it encounters fm-2, the path between the two nodes encodes the relationship between them; this path is guaranteed to be a member of the set of shortest possible paths between the two nodes. A simple set of rules allows Salix to select the appropriate expression for the relationship. For example, if the path from

[4]The degree of a relation is how many one-step links need to be traversed to get from one person to the other. Thus, *mother* and *sister* are first degree relations, *grandmother* and *aunt* are second degree relations, and *great-aunt* is a third degree relation. *First cousin once removed* is a fifth degree relation.

um the people whose whose house we go to are Margaret and Bill
who are Mommy's
....um
sh—Margaret's my great-aunt
so it must be Mommy's aunt

you are my niece
no you're my grand-niece
uh your mother is my niece

and also um Eleanor and Elizabeth come
who are....cousins of....all of us
um I don't know what generation cousins they are

Figure 8.4: Expressing relations in family descriptions.

fm-1 to fm-2 is mother-of sister-of daughter-of, the relation between the two nodes can be expressed as "(first) cousin." If there is not a three degree or less relationship between fm-1 and fm-2, then Salix will not attempt to express the relationship.

The ability to compute some family relationships serves two purposes. Not only does it make the variety of Salix's texts more closely approximate that found in people's, but it serves the important function of making Salix's family domain texts more connected, and thus more coherent. Consider the example text in figure 8.5. Lines (1-4) are generated in the usual way. At line (1), a new current object, Alice, is selected and said. At lines (2-3), the current object is elaborated on. Next, no applications of saying strategies are available, since they have all just been applied, and no applications of finding strategies are available, since all of Alice's first degree relatives (the ones who would match with **find-next-to-say**) are *already-said?*. The completeness criterion, mentioning all the family members, has not been satisfied, so an application of **jump** with Elizabeth is selected. Elizabeth becomes the current node, Alice becomes the previous node, and an application of **say-object** with Elizabeth produces line (4). Now Salix can see if it can *relate back* Elizabeth, the current node, to Alice, the previous node. Search is performed and determines that Elizabeth is Alice's cousin, and this is generated (line 5).

```
(1)  then there's Alice
(2)  Alice's husband is George
(3)  their children are Dorothy and Jeff and Carleton and David
(4)  and then there's Elizabeth
(5)  who is Alice's cousin
```

Figure 8.5: A sample text demonstrating the notion of *relating back*.

and on our righthand side would be a door
which leads to Penni's room
and you walk in there
....and there are two windows
I think
....in the....opposite corner from the one in which you enter
where the door is
....and they are at right angles to each other
one's on the lefthand wall
and one's on the wall that you would be facing
then on the righthand side....of her room
is the closet
or two closets
I'm not sure which
...
on the righthand side we go into Claire's room
and we walk in there
and the door again is in the same relationship to the windows as it is in Penni's room
it's kitty—corner from....where the windows are
and the windows are on the same walls
only this room is smaller
and the closet is also in the same place

Figure 8.6: A speaker repeating part of her description.

8.4.5 Reference to Previous Text

A final desideratum for Salix's text generation performance is for Salix to be able to indicate that some part of its text is similar to a previous section. This is something that people do, as can be seen in figure 8.6. In this text, the speaker constructs one part of her house description, that of Claire's room, in reference to another part, that of Penni's room. In order to be able to do this, the speaker has to remember what she said about Penni's room.[5] The speaker is using what she already said about the house as a *resource* in the process of constructing her description. For Salix, a previous-text resource could be added to the collection of resources that Salix already has, which includes the knowledge about the objects, properties, and relations in the domain that are encoded in the KB, the knowledge about the appropriate lex-items, and the knowledge about how text is put together that is encoded in the strategies.

What remains an open issue is *how* the previous-text resource would be encoded and accessed. For instance, the generated text itself could be the resource, but then there would have to be some way to interpret it. Alternatively, there could be some intermediate representation in terms of the strategy applications chosen to build the text that would be easier for Salix to interpret. Or, the record of what has been said can be made as an annotation on the KB: this annotation would be a more elaborate form of the *already-said?* and *already-used?* markings that would include information indicating the order of the KB items' appearance in the text. However, it remains unclear how the previous-text information could be used by Salix. For instance, what

[5]One could argue that all she need do is remember *that* she already described Penni's room, since she is disposed to describe it only one way. This is not contradicted by the data, but it seems a less plausible hypothesis.

would prompt the system to try to compare Claire's room to Penni's? We do not want Salix to try to compare everything it is talking about to everything it has already talked about: this is inefficient and unrealistic. To address this issue requires more research into how people use previous text as a resource.

Chapter 9

Conclusion

In this thesis I have presented an architecture for generating extended text, implemented in a system called Salix. Salix incrementally generates natural language texts whose structure is derived from the domain structure of the subject matter. The architecture is composed of data driven domain independent strategies for producing increments of text and metastrategies that combine or choose among all strategies that are applicable at each increment or decide what to do if no strategy applies. Salix's capabilities have been demonstrated in generating texts, in the domains of houses and families, that are comparable to descriptions elicited from human speakers. A total of eleven domain independent strategies have been implemented; two additional spatial metastrategies are used in the house domain. Salix has also been partially implemented to generate texts describing physical processes.

In this final chapter, I review the five themes of this thesis—the unified process of generation, local organization, coherence, focus, and domain independence—that I consider contributions to the field of natural language generation. For each of these themes, I contrast the position taken in this thesis with others in the field.

9.1 Unified Process of Generation

Most generation systems are divided into *strategic* and *tactical* components. The strategic component is usually conceived of as deciding what to say, and, perhaps, making decisions about how to say it, but *not* making decisions about the subsentential structure of the text, which is handled by a separate tactical component.

In Salix, the strategic and the tactical are tightly interleaved. As soon as Salix, by selecting a strategy application, decides on what to say, the surface structure realization functions are called to say it. The increments that are realized vary in size from a word to a clause. While virtually all surface structure realization systems take the sentence to be the basic unit of generation, Salix has no notion of sentences or sentence boundaries.

9.2 Local Organization

Salix builds text using only a local context. This context is limited to:

- A single *current node* in the knowledge base.

- The *previous node* (the last current node) and the *last-said node* (the last node realized in the surface structure.
- The last *saying strategy* that has been applied.
- The *current relation*, which was used to reach the current node.
- Annotations in the knowledge base of which nodes have been *already-said?* and which relations in which a node is involved have been *already-used?*.
- In some domains, the *trajectory*, which is a generalization over the last two choices made.

The final overall structure of the resultant text is not a concern of the process that builds it.

Any text, whether produced by Salix, another generation system, or a person, can be examined and a global structure for it found. Finding a global structure for a text is a common practice in text analysis; Hobbs (1985) and Mann & Thompson (1987) explain methodologies for doing this. For most analysts, including these authors, a tree is the global structure used to describe texts. It is probable that most well-formed and many ill-formed texts can be given a tree structure using these methods.

Because a global tree structure seems to be a characteristic of well-formed text, many generation systems explicitly build that tree for their texts, either by starting with a built or partially built tree and filling it in (Dale 1990, Cawsey 1990) or by using a mechanism like a hierarchical planner to build a tree (Moore & Swartout 1989, Hovy 1990). Similarly, schema-based systems (McKeown 1985, Paris 1988) stipulate a global structure in the schema that is selected.

There are certainly global structures in texts, such as the stylized format of a letter, or the ordering of sections and subsections in a technical paper. And there are characteristics of some texts that require a global perspective, such as making sure to order the examples before the conclusions and introduce concepts before referring to them. But in the construction of text, these considerations are more a resource to refer to during the process than a prescription for how any stretch of text is to be built. As I outlined in the introduction to this thesis, there are many sources of constraints on different sorts of text, and not all of them are global in nature.

One might argue that the domains themselves have a global tree structure, and this structure has some special impact on the organization of texts in these domains (Linde (1974), for instance, hypothesizes that speakers have tree structured mental models of their houses). My work with the data suggests that this is another case of tree structure inferred by some *post hoc* analysis. People's texts do not contain indications that they rely on a tree structure in deciding what to say next. In fact, if one superimposes on a sketch of the house a graph whose nodes are objects mentioned and whose arcs connect the objects in order, the graph looks like a plate of spaghetti, not a tree (see figure 2.3). And it is worth noting that "family trees" are not actually trees—each child has more than one parent.

9.3 Coherence

Just as with sentence grammaticality, people tend to agree that some texts are perfectly coherent and some perfectly incoherent, but, just as with grammaticality, there is a gray area in which fall texts on which people's judgements differ, and not always for articulable reasons. Coherence is elusive: we want our texts to be coherent, but it is hard to say exactly what coherence *is*.

The well-formedness of global tree structure discussed in the previous section is a candidate for an operationalization of coherence: coherence seems to be in part a function of structure—you can tell how things relate if you can see their relations as part of a tree. Indeed coherence seems to be the primary justification in the literature for tree structure (Mann & Thompson 1987, Hobbs 1985). In summing up a review of text generation research, Hovy (1990) articulates this position most clearly: "[I]t seems fairly likely that all approaches to the problem of dynamically constructing coherent texts will use a tree to capture the dependencies among and order of the clauses." Salix's increments, which are the interdependent and ordered units of the texts it dynamically constructs, are not all clauses, but most of them are and the resulting texts have clauses much like any other. Either the texts Salix generates are not coherent, or Salix is an instance proof that systems can generate coherent text without trees.

The literature offers a more specific reason for relying on tree structures for texts: such a structure allows a tutoring or advising system constructing an explanation to more easily locate the source of a listener's misunderstanding (as expressed by his response or question) and effect a repair of the explanation (Moore & Swartout 1989, Cawsey 1990). However, as Cawsey points out, the issues involved in an explanation-giving situation are more of understandability than of coherence. Whether or not a tree structure is needed to keep track of the development of an explanation, such a structure cannot be the only source of constraints on the text, because of the interactive nature of the situation. The usual solution in this case is to build a structured text, and then augment or modify it to make it more understandable in response to the needs of the listener or user. This augmentation to the text generation process is a form of hearer or user modeling (see Kass & Finin, 1988, for an overview). In many systems, hearer models are predefined data structures (or models induced from types of questions asked) that include, for example, facts that distinguish an explanation for an expert from one for a novice. These models are consulted in the process of building texts, which are still tree structured. But Cawsey (1991) observes that in practice user models do not work very well and that more emphasis should be put on actually responding to the user.

The coherence in the texts Salix generates derives from domain structure. The domain structure not only obviates the need for building a separate structure for coherence, it also provides a natural way for keeping track of one's place in a text. Since queries are usually about subject matter, rather than the way in which it is presented, a system like Salix could use domain structure to locate the source of a misunderstanding and negotiate a repair with the hearer.

9.4 Focus

The local organization of text successfully addresses the phenomenon of *focus* that has been an important concern in text generation since McKeown's work (1985). Focusing is represented as a constraint on what can be said next, and how it can be said (e.g., the surface order of noun phrases and whether a noun phrase can be pronominalized); the focus is the item or items which are currently what the text is about. "What can be said next" has two parts: which items that have not been mentioned yet are appropriate candidates for mentioning now; and which of those appropriate candidates have already been said. McKeown's system used a stack mechanism based on Sidner's focusing algorithm (1979); more recently, focus trees have been used to provide a more complete record of what has been mentioned and in what context (Hovy & McCoy 1989).

The local organization approach is precisely about what can be said next; Salix knows directly what is available to be mentioned. In addition, Salix records in the knowledge base when an object, or a relationship between an object and another object or property, has been mentioned in the text,

or can be inferred. This annotation suppresses these objects and relations from being candidates for what to say next, though a strategy may include an already-said node in the description of the current node.

Salix's architecture naturally has the capability of using pronouns appropriately. The distinction between expressing relations by **say-elaborating-relation** and using relations to guide the finding strategies accounts for how focus changes throughout a text. For saying strategies, the current node is the node about which something is said; thus this current node is "in focus" and can be pronominalized appropriately. Finding strategies find another node to be the current node, thus accomplishing, *de facto*, a "focus shift."

Particularly complex example of anaphora are found in the family domain. As well as needing to know the current node and the previous node, Salix needs to keep track of the *last-said* node. For instance, if the last-said family member is of the same gender, pronominalization must be blocked. (See section 4.3 for a complete discussion.)

The focusing issue of what to say (or not say) next is handled by Salix as a matter of course. Additional issues of when pronominalization is allowed or proscribed are handled in a straightforward fashion without needing to augment Salix's local context. It is certainly true that there are some instances of focusing, such as returning to a node that was mentioned farther back than the reach of Salix's memory, that Salix would not handle gracefully. (The node would be annotated as having been said, but the circumstances under which that had happened would be forgotten.) While such a capability is important in the long run, in the short term, Salix has all the focusing capabilities it needs to produce coherent text.

9.5 Domain Independence

A primary concern in natural language generation is domain independence: we want our approaches to be usable in domains that we might not have thought of when doing our own research, and we want our theories to be relevant to others' work. So a text generation system must be portable between domains.

A candidate for a domain independent theory of text structure is rhetoric, a discipline that has been vigorous from ancient Greece (Aristotle 1926) through this century (Perelman & Olbrechts-Tyteca 1969). Rhetoric is the art of persuading an audience of a proposition, and a rhetorician can be persuasive about anything. This attractive property of rhetoric—that it can be used in any domain—has encouraged the notion that "rhetorical relations" are the appropriate source of flexible, domain independent text structure. This notion is reflected in the name of Rhetorical Structure Theory (RST) (Mann & Thompson 1987), a comprehensive theory of text structure based on the study of a variety of texts. RST comprises 23 relations between clauses, seven of which, such as **motivation** and **justify**, are classified as *presentational*. That is, in the spirit of rhetoric, they are used to "increase some inclination" of the audience. The other 16, like **sequence**, **elaboration**, and **non-volitional cause**, are listed as *subject matter* relations, relations which, in effect, convey information about relations in the domain. RST is a descriptive theory, but there have been many generation systems that have incorporated all or part of it in their text structuring component (for example, Hovy 1990, Moore & Swartout 1989, Cawsey 1990, Wahlster *et al.* 1991; McKeown's (1985) and Paris's (1988) systems incorporate sets of rhetorical relations similar to RST's).

These systems are domain independent in virtue of their use of rhetorical relations. But of course rhetorical relations are not completely divorced from the subject matter of a text; in fact, Mann & Thompson are quite clear that at least two-thirds of their proposed relations are subject

matter ones. Additionally, some of the subject matter relations are not relevant in all domains, or they express very different things in different domains. For example, **non-volitional cause** is not the same relation in the domain of exploding bombs as it is in the domain of international affairs. In fact, the ability to exploit a causal relation requires more domain dependent knowledge than the ability to exploit a spatial one. Whereas it is not clear how to transfer a theory of causality from bomb detonations to diplomatic relations, the spatial relations exploitable in a house domain are exploitable in the same way in the domains of circuit layouts and the structure of mechanical devices.

As Rambow *et al.* (1991) point out, using subject matter RST relations requires knowing what the relations in the domain are—knowing, for example, what domain relations are appropriate for a **sequence** relation, and further, knowing how to choose among them. In other words, text structuring approaches based on rhetorical relation theories need to know exactly the same sort of domain dependent information about domain relations as does the approach taken in Salix. Accordingly, the domain dependence is encapsulated in a similar way: a well-designed RST-based generation system can implement **sequence** and other subject matter relations in a way as domain independent as Salix's **find-next-to-say** strategy.

Recall that in Salix sensitivity to knowledge that is not domain independent shows up not only in the ability to exploit domain relations but in the spatial metastrategies, like **say-left-right-and-center**, which improve the quality of the text. This domain sensitivity exemplifies Rambow's notion of *domain communication knowledge* (1990). Rambow distinguishes three types of knowledge: communication knowledge (like rhetorical relations); domain knowledge (specifically about the domain, independent of communication); and domain communication knowledge—domain sensitive ways of *talking* about things in a domain. As Rambow observes, no communication is possible without domain communication knowledge; the challenge, which Salix takes up, is to encode and exploit knowledge about the domain in a principled fashion.

9.6 Generating Text without Trees

I have presented an architecture for incrementally generating text. The architecture is embodied in a program, Salix, that produces texts in the domains of houses and of families. Texts are produced in increments by strategies that decide what to say next. What to say next is determined by the local structure of the domain and by a context of information about what has already been said.

The work described here has emphasized modeling and evaluating texts whose structures reflect the structure of the subject matter. The ability to exploit domain structure to produce text that is *about* something is one of several skills that our text generation systems must have if they are ever to be capable of using language in all the ways that people do.

Text generating systems need to choose and order what to say, produce text that is coherent, use anaphora and other textual cues appropriately, and be domain independent. Salix exploits domain structure by means of domain independent strategies, builds coherent text incrementally by making local choices without constructing or relying on a tree, and accounts for focusing phenomena without resorting to focus trees and other special mechanisms. Salix is a system that generates text without trees.

Appendix A

Transcribed Data for House and Apartment

The transcription conventions I have used are the following:

1. Each line of text consists of one segment.
2. No sentence punctuation is used. The characters "!" and "?" indicate intonation only.
3. An ellipsis, "...", indicates a discernible pause.
4. A dash, "—", indicates a restart, self-interruption, or other-interruption. This may either occur within a word or the interruption may be followed by something completely different.
5. When two speakers talk at once, overlap is not indicated but the speakers' talk is interleaved as accurately as possible.
6. Question marks in brackets, "[??]", indicate an unintelligible portion of the tape.

A.1 Descriptions of House Layout

5 October 1986

These interviews were conducted by me.

A.1.1 Claire

This interview was conducted in Claire's room. Claire is a native German speaker, but has lived in the US for two years, and is generally considered quite fluent.

Penni
this is Claire on the 28th of September and would you please describe for me the layout of our house

Claire
oh, shit
I've tried that in Germany
I tried to—to draw floor maps and I couldn't

Penni
but this is a verbal description

Claire
I—I couldn't come up with a floor map—ok
um....well, let me start here
this is my room—what do you want? which detail?
just just what—which comes where or descriptions of what's inside?

Penni
the layout of the house
not so much the different furniture and decorations and stuff

Claire
ok
my room is....a little longer than wider
but it looks pretty square
it has two windows....sort of at this one corner
that is pretty much diagonally opposed to the door
which goes outside to the little hallroom before the bathroom
and Ann's room
the bathroom is fairly uh....not square
I don't know, does this—the bathroom stick out?

Penni
no!

Claire
it doesn't?

Penni
I don't think you're supposed—

Claire
it doesn't?

Penni
—to ask me though!

Claire
well, I don't—ok—anyway
so there's the bathroom that has one adjacent wall to mine
the other one is adjacent to Ann's room
which is a—a corner room in the house

so she has—I think—two windows
sortof very opposite to mine
aren't they?
one facing out to 116
the other facing out to the—to the side yard
and....her room is about my size
about the same shape I would say
....not my size, the size of my room
so if you would come back out
see I'm sortof walking you through the house
as you sortof would—would come out of her room again
um....of course you would face my door
and on the righthand side you have this—this funny little
....uh, leftover from the closet that
....ought to be her closet
but isn't—isn't reachable from the room
so sortof go around and there's this closet door
....my shoes
and her linen
and....then you go around this same thing and from the other side
it has a door which is sortof her—her....part of her wardrobe
which leaves me with trying to describe this funny front door of ours
which is right around the corner of this thing
leaving a—a very tiny little small hallway....for
....originally the front door um....but you can barely open
and if you would come in—if you were to come in through the front door
you would sortof just....stand in front of this wall
which isn't much better than the kitchen door....no much worse
but anyway so there's this little thing with the kitty litter now
....sortof um....fenced off these days by the piano
which brings us back into the living room
....which is um a large room with um....some open flare
because it doesn't really have....walls to divide it
there are these two house walls
with the two windows
....but the....the division from our part of the house is just—our two Ann and mine
....rooms is just this—this funny closet and—and to the other side
....it is the—the basement staircase fencing
and—and the closet thing that's right in the middle of the house
which I'll describe a little mm later
so that's....that's the living room
which has sort of two....ways to get into
one being....the sortof where the main entrance is
the kitchen entrance
....which is a little hallway....that
....is between the living room and the kitchen
....right inside is—is the—if you come from the living room
is the kitchen door
on the lefthand side is the door to the basement

and if you sort of go further there's this funny-shaped little—niche
that's sortof left over from the basement stair thing
and—and this other cupboard-type thing a wardrobe-like thing
and....that sortof brings you into the kitchen
which is—which is maybe the largest room in the whole house
....um....we are having the big screen door on the....entrance....door side wall house
and....windows again
at the backyard side of the house
....and that then has one—one wall common with your room
which I think must be—I haven't mentioned yet
which you reach from—from the kitchen
if you go—foof—well out that way
it's a big room I suppose
and it's fairly—it's fairly square....sortof
um again having two windows
at the backyard side and towards the sideyard
to the party house
and....well....yeah not much else
and so this—this little hallway that
....you get into from the kitchen
that gets you to your room and for that matter to the little broom closet
is the other little hallway
that connects the kitchen to the living room and
....Ann's and my....mm whatever rooms
so I think that's just about it
do you want the basement too and the garage?

A.1.2 Ann

This interview was conducted on the back steps of the house.

Penni
this is Ann on September 28th

Ann
1986

Penni
could you please describe the layout of our house for me

Ann
the layout of our house
in terms of directions?

Penni
in terms of like a floor plan

Ann
ok
a floor plan
toward the front of the house
facing the street is the living room
....and directly in back of that is the kitchen
between the two....is a hallway
....which leads to Penni's room
beside the kitchen
and on the other end leads to....short hallway
that goes at right angles
then there's Claire's room on the right
my room on the left
and the bathroom in the middle

Penni
can you give me more detail of what the rooms are like

Ann
what the rooms are like
....you mean what they look like?

Penni
well you've said the relationship of the rooms to each other—

Ann
right

Penni
—and can you say more about what the layout of each room is

Ann
ah the layout of each room
furniture-wise

Penni
well and windows and doors—

Ann
oh what—whenever that comes up!
ok
well in the living room
....if we're facing the living room from the kitchen area or the hallway
there's a front door
on the righthand side
and a large window sortof in the middle of that wall
that faces Bub's Bar-B-Q
then there's a smaller window on the lefthand side
....then on....the wall opposite the large window are two doorways

one leads—this is complicated

Penni
mm-hm
that's part of the point

Ann
one—one....ok
the....doorway on the lefthand side
as we're facing out towards the street
....is....leads to a very short hallway
in which we have
....the side door
and opposite that the door that goes down to the basement
then in the righthand....doorway we have like a um
....we have a hall—large hallway that leads into the kitchen
and at right angles the smaller hallway that leads to the bedrooms
then in the kitchen
....there's a large window which faces the backyard
with two smaller windows directly flanking it
and....if we're facing....towards the backyard now
on the righthand side is....a sliding glass door
and....a few feet from that is a smaller window
towards the living room
then....on the wall which....would
....partition the kitchen from the living room
there is a closet
....and behind that closet would be the stairwell that goes down to the basement
so there's like a block between the kitchen and the living room
um....in the kitchen
if we're again facing the back yard
on the lefthand side is the stove
....then....a refrigerator
and beneath that large window....is the sink
and next to that
on the righthand side
is the dishwasher
boy if somebody tried to reconstruct it from this they would never get it
ok
Penni's room I'm not so familiar with
but I believe
....ok you walk in the door
....if you were go—going to Penni's room from the kitchen
and....we're facing the backyard
we turn around
and face the wall to our left
walk forward
which would bring us into the hallway
and on our righthand side would be a door

which leads to Penni's room
and you walk in there
....and there are two windows
I think
....in the....opposite corner from the one in which you enter
where the door is
....and they are at right angles to each other
one's on the lefthand wall
and one's on the wall that you would be facing
then on the righthand side....of her room
is the closet
or two closets
I'm not sure which
other than that not much of interest
ok
we turn around and leave Penni's room
and now we're facing the street again
and we walk down that....wide hallway
which is almost a room in itself
there's a closet
on the lefthand wall
in the lefthand wall let's say....of that hallway
then we go to that smaller hallway on the righthand side
to get to the other two bedrooms and bathroom
when we walk straight ahead
we go into the bathroom
which has a window straight in front of you
and on the—underneath that window is the toilet
on the righthand wall....would be the bathtub
....and right next to the door on the righthand wall would be the sink
so we leave that room
....and....if we're still facing in the hallway the same direction in which we came in
on the righthand side we go into Claire's room
and we walk in there
and the door again is in the same relationship to the windows as it is in Penni's room
it's kitty—corner from....where the windows are
and the windows are on the same walls
only this room is smaller
and the closet is also in the same place
so we leave Claire's room and we walk....across the hall
then
across this T....the top of the T
which would be
....um the small—a small hallway from well let's see
how should we describe this
....the hallway to the bedrooms
....in relationship to that wide hallway between the living room and kitchen is like a T
there's a short bit
....and then there's a top of the T

and that's the—at one end of the T is Claire's room
at one end of the T is my room
ok so we're walking across that T and we get to my room
....and again
let's see
the windows this time are not quite in the same position
cause they're not in the corners of the room
um if you walk in the door
the wall on the lefthand side has a window
approximately in the middle of that wall
then the wall opposite you has
....another window approximately in the middle of that wall
then....on the lefthand wall....is a closet door....which
if you have the door to the room open
the door to the closet is right in back of it
....then that's about it
oh yes
we leave that
and we go back out to that T-shaped hallway
the base of the T
the upright part of the T
....um....if we're facing the bathroom
has....a closet on the lefthand side
....and I didn't mention the closet in the living room either
but if we go back out then to the living room
facing the street there's a closet on the righthand wall
and in back of that is this front door
....next to which is the piano
oh god you want more detail?

Penni
no I think you've probably covered it

Ann
you asked for it

A.1.3 Dick

This interview was conducted on the back steps of the house. It was interrupted at one point (the sliding glass doors) by Ann.

Penni
ok
this is Dick on September 28th
and would you please describe for me the layout of this house

Dick
the layout of this house
ok
it's a one-level ranch house
you want the whole yard?

Penni
no just the house

Dick
just the house
ok
oh
ok there's a big—in the living room
there's a big picture window
looks out onto the street
you have....wooden floors
throughout the whole house
except for the kitchen
there's a piano
in the—in the livingroom
and a big sofa....and a few plants
not—not that—not much else for furniture
but—um....in the kitchen there's....sink out looks out over the back yard
....well the sink doesn't
um....it's a fairly spacious house
it's three bedrooms....bathroom....

Penni
how are they all related to each other?

Dick
ok
one....two of the bedrooms are adjoining
one looks out on the sideyard
and one looks out on the backyard
the third bedroom is opposite the bathroom
....which looks out in that—or looks out onto the front yard
....um....there's a....two-car garage
....and a little....walkway
patio
that leads to the garage from the house
it's covered
basically that's pretty much your house in a nutshell

Penni
ok how bout hallways and things like that and doors and—

Dick
oh....there's a....leading—once you walk into the house
there's a door that leads down into the basement
....um
in the kitchen there's one closet
....and a little cabinet space
....um....sliding glass doors....[interruption]

Penni
That's sliding!

Dick
get out of here!
one roommate's really obnoxious
um leading into the living room area
well there's....a squared-off area where there's one thin hallway
leads from the kitchen into the living room
then there's a wider one
that leads from....the kitchen into the living room
....about three times as wide
um going from the living room toward the bathroom
there's a T-shaped hallway
and at the top of each T—at the tip of each T there's a bedroom on each one
and directly ahead there's the bathroom
um the other bedroom....comes off the larger....hallway
....leading into the living room
....um....let's see
....there's a small closet on the side of
....well there's a closet
in the living room
the laundry closet
there's a smaller one
from a square subsection
leading into the living room
um off of the living room there's a small entrance for the front door
which isn't used that much
there's a small stairwell with a railing out front
um what else
colors?
any of that stuff?

Penni
no
just if you think you've described the physical house

Dick
the physical house

Penni
and that was sortof the—the layout

Dick
mm-hm
ok?

Penni
ok

A.1.4 Heidi

This interview was conducted over the telephone. Paragraph breaks indicate prompts from me. The remark about the chimney is not bizarre: the speaker grew up in a 200-year-old colonial house built around a central chimney.

this is uh uh Sunday September 28th 1986
and this is Heidi
and I don't know where the microphone is on this thing but—

yes
yes
your house?
uh um...well
it's a one-level....house
ah....ih....has....tw—three bedrooms
bathroom
....ah living room
and a combination kitchen-dining room
....um it's got a garage that's attached—you can get a walkway to the garage
that you don't get wet when you go out
ah....it's....fairly small....size
it's got a basement that's....finished I think
....um
....and you can go from the outside down to the basement
as in like a storm cellar
or you can get into the basement from the—from near the door
side door
there's a front door
side door
sliding glass door
....ah what else do you want to know?

oh
um well from the driveway
if you came in through the garage and walked across the walkway
you would go in the side door
which is next to the sliding glass door
and....as you walk in

straight ahead is the door to the cellar
....and....to your right is the kitchen-dining room combination
....it's built around a central....chimney?
it's not a chimney
central area
I guess it's the stairways
it's built around that um
and so off to your right is the dining room-kitchen
....and then diagonally up to your right
....as you go past the kitchen
is one of the bedrooms
....ah you go around the chimney to your left and
....in the front corner of the house is the living room
....and if you keep going to your—as you go around to the left
if you go right there's a closet to your left
and on the right is....ah another bedroom
....and on the left is another bedroom
and straight ahead is the bathroom

it's red
....ah it's got a—a lawn
a lawn in back
not too much lawn in front
....ah the neighbors are really close by
....um not too many trees in the yard
ah....looks back over a swamp
....anything else?
ah basically white....white walls....um
....ah the front door isn't used too much
um....it's got a closet in the front hall
and a closet—lots of closets
....ah....I don't know what else

A.1.5 Keith

This interview was conducted over the telephone. Paragraph breaks indicate prompts from me. I had asked Keith to repeat these prompts.

September 29th
....right and I must be Keith

Yes

the layout of your house
um....well from the outside
....um there's the garage connected by the covered....walkway with the wall on one side....to the house
and it....um

....I suddenly went blank on the number of cars you can put in the garage
but....it must be two yeah
um and then the walkway goes up to the main entrance
which.....is like in a miniature small hallway between the living room and the kitchen
and the kitchen has sliding glass doors
....and
....then there's—there's kind of a big central....room-thing
I mean like when you come in
....this seems very strange telling you this Penni um....um
....which leads to the stairs down to the cellar
....so there's a big central thing
which must be about....10 feet across or whatever
with closets in it
that kind of separates the whole house
then there's a living room on one side and a kitchen on the other side
....nd then to one side of the kitchen is your bedroom
....and then
....I guess the house is in the shape....of an....L
where your bedroom is on one of the inside legs
and Claire's bedroom's on the other
and the third bedroom is on the outside leg facing the road
and then the living room is on that leg
....and then the kitchen and the hallway and the living room make up the other....side
and then the basement's split into two rooms
one underneath....the kitchen and your room
....or just your room
I'm not entirely sure
and—and the other underneath the rest of the house
....and there's an outside entrance to the—to the basement
....and I guess that's how I would describe the layout of your house
although if I were probably describing it to anyone else
I might have given a little more size information

um....the kitchen....is probably....oh 10 by 15 maybe
with the 15 side on the sliding glass doors
it's kinda hard to tell
because it feels bigger on one side where you're near the....hallway
and the living room....is about the same size—
and then of course there's that weird—there's a weird little alcove that goes out to the front door
which of course no one in New England uses front doors
um....and your bedroom....is....a little smaller than the kitchen I guess
....and Claire's bedroom is....a little smaller than that
yours is more rec....tangular—I guess—well—I'm not sure—
I guess they're all fairly square
....so your bedroom must be....like around 12 feet square
....maybe a little more
and Claire's is....smaller than that
and the other one must be more like....8 by—no, 10 by....12 or 10 by 10 or something
.....and the garage is big enough for two cars!

A.2 Descriptions of Apartment Layout

17 August 1989

These interviews were conducted by a colleague.

A.2.1 Hannah

George
ok...

Hannah
is this *linguistics* or—

George
I *guess* so
...oh I've just been told what to ask you
I don't exactly know what's going on here

Hannah
ok

George
um...my name is George by the way and your name is

Hannah
Hannah

George
ok
and today I was told to say was the—August the 17th 1989
...and...um...I'd like you to describe what your apartment is like

Hannah
ok
...it's four rooms
...um...um...second floor of a large colonial house
...uh...it's renovated
comfortable...small...um...

George
can you say like what's where and what the rooms are

Hannah
ok...it has...a large front room that we use as a bedroom
it has a sunporch which has been converted into a kitchen

...and...a...a...family room
...it has a small entryway
...with a bookcase 'n a desk in it
...a large bathroom
...and a den

George
and um what's in the den

Hannah
um...the den has...ah...a study...and general workroom...for my husband and I
...uh it has two desks and an ironing board

George
uh huh
...so I'm trying to imagine what this would look like
...um...how would I get from one place to another

Hannah
ok
...ahm...rooms are kindof in an L
...with the den being at the top of the *long* arm of the L
...and the bedroom...being the entire bottom part of the L
the entryway's [??] about in the middle of the long arm of the L

George
...k...and...so the entryway...you have two doors one going in each direction?

Hannah
n—ahm—the ent—yes
the entryway has an entrance from the outside and
...another—and 'n an entrance into the main part—*two* entrances into the main part of the house

George
so which entrance [??]

Hannah
no *three* entrances
sorry

[both laugh]

Hannah
off the entryway you have an entr—a door into the den
a door into the bathroom
and an archway without a door into the sunporch

George
uh-huh
then to get to the kitchen I'd go how?

Hannah
through the archway
through the sunporch
and into the kitchen
the sunporch and the kitchen are basically one big room

George
uh-huh
ok
um could you compare it with Penni's ar—uh—apartment [??] I know what *that* looks like?

Hannah
yes...um...well Penni's is one long more like a capital I

George
uh-huh

Hannah
it doesn't have a *kink* in it [??] the bottom
...ahm...the bedroom is somewhat bigger then Penni's bed—the bed—the bedroom in Penni's apartment
ahm...hers is more straight through
not as many twists and turns
ahm...both of them have big walk-in closets
ahm...the bathroom in Penni's apartment is right off the bedroom
whereas in this apartment it's right off the entryway
much farther from the bedroom
ahm...I would s—describe Penni's as having three rooms and a
bathroom whereas ah this one I would describe more as having four rooms
ahm...although...really they probably both have three

George
mm-hm
because the kitchen and sunroom are really one room sortof

Hannah
one room yeah

George
uh-huh

Hannah
essentially
there's no door between them at all

George
I think that may be everything which we need

Hannah
ok!

George
great
I wonder how I stop this
just by pushing stop I bet

A.2.2 Mike

George
hi

Mike
hi

George
I'm George
and....you're

Mike
Mike

George
hi
um
Penni has asked me to ask you to describe your apartment
so that I would know what it's like if
...fact I've never been in it so
...it would give me some idea what it was like

Mike
ok let's see there's
...a den
...and hallway
...and livingroom kitchen together
and a very large bathroom and then a huge bedroom

George
uh-huh
and how are they all connected together?

Mike
ahm...the bathroom hall and den are on one line
and bedroom kitchen and livingroom are on the other line

George
mm-hm
...mm
...mm

Mike
it's like this

George
oh so it's...

Mike
nn

George
sortof like ah

Mike
one

George
Z-shape almost

Mike
well no
it's it's like...there're two parallel lines but they don't they don't they're not exactly they're they're parallel

George
right

Mike
they're

George
they don't right yeah it's not one line

Mike
ok

George
so...ahm...can you tell me what's in the different rooms?

Mike
ahm...in the den there's a stereo
...ah a small black and white tv
...we run [??] an entertainment center
we have records and
...a lot of miscellaneous things
[??] sewing
and then there's my bureau
and closet
and a big desk with a calendar on it and the cat sits on it
and a sewing machine

George
the cat sits on it?

Mike
yes

George
uh-huh

Mike
and there's ah...ah an air conditioner
the hallway has a small shelf
with theatre books in it
and television books
and travel books
and there are some...knickknacks on it that we got for our wedding
glasses and picture...s and
...and some *wooden* knickknacks
'n there's a bureau an old bureau that has...um
...our bills and...checkbooks and...pens and pencils and similar things
oh and then there's—there's also a closet here [??] in the hallway a walk-in closet
where—where we store everything
food
cat food
old toaster ovens

George
[laugh]

Mike
Tide
...um fabric softener sheets
um sugar
Christmas tree a fake Christmas tree
old sheet music
ah a wastepaper basket to put the New York Times in
winter clothes

ahm...lots and lots of boxes everywhere
ribbon
beer
...soda
um plastic liners for trash cans
um...'n bathroom is got green tile and white floor
fiberglas shower
's a closet in there that we keep the vacuum cleaner in
and cleaning supplies and bubble bath and PeptoBismol
...and a a mop and a bucket
and there's a Mike that's got a cover on it with lots of Hannah's books on it and her bubb—more bubblebath
and kleenex box and there's a sink
that has toothpaste on it
and there's a shelf next that that has um ah qtips and ah...straight razors and all—all of Hannah's makeup ...'s see living room has a dining room table
television with a vcr on top of it
and a storage compartment underneath with all kinds of movies 'n ah tapes
a large couch that's been *wrecked* on one side by the cat
it's *gouged*

George
[laugh]

Mike
quite gouged
and a beautiful new clock above it
and a parson's table next to that with ah two movie guides quick movie guides
[??] watching [??]
dining room table that we eat on
easy chair
...'s a rug on top of another—a rug like this
[??] and
...and there are windows all the way down in—into the little kitchenette
there's a storage shelf there
with...dishdrainer on it and let's see
...spare pots and pans and
um...matches and
...there's the sink—there's the refrigerator
with telephone—telephone books on top of it and telephone ...'n a stove and cou—small counter next to it
...and then there's the bedroom
...and there's a k—a huge kingsize bed
...and there's a—an endtable next to it
...with over with a a lamp and a clock radio
...and some stuff of mine like the Boston University cup
...and...um then there's a wall–
...and then there's a window behind that
...and there's a wall

and there's another window
and there's some...cedar chests of Hannah's
...that have blankets and sheets in them
...and...there's her bureau
...in the middle of two windows on either side
...with all of her makeup on top of it and clothes
...and there's...a small thing with all her clothes
...and there's another great big bookshelf and all her spare books
...and there's a small end table over on her side
...um...a small digital clock and more kleenex and
...ok
and then there's there's also there's also a hallway behind there we share with Penni so it's...with...more books
and there's Penni's bicycle [??]
so

George
that's very complete
could you um...compare your apartment with Penni's
which I know

Mike
um
Penni's is
Penni's is like what they call a railroad apartment
it goes straight back
'n ours has some angles in it
...that's—that's one—that's one of the big differences
her kitchen is much bigger
...'n her bedroom is much smaller
and her living room is...ahm...is—is somewhat bigger
...and hers is darker
it's cooler
and...let's see
her bathroom may be a little bit smaller than ours
but it has a window
...
...let's see we both get—we both get east sun
she gets a crossbreeze [??] which goes all the way back and we don't
just stays in there

George
uh-huh

Mike
...our apartment you can—you don't have to turn the heat up in January in the middle of the day if the sun is out

George
uh-huh

Mike
solar windows so
but in summertime it's very very hot over there
so you have to close—we have some shades
...
...I—I would say that was the second to the cool—that was—this is the third to the coolest
the next to the hottest apartment in this—

George
—this is the hottest

Mike
this is the hottest
that's what I would say
cause I've lived over here too

George
uh-huh

Mike
...ahm...let's see what else that's different about it...ahm
...
...I guess we have more closet space
in general
a little bit more

George
more closets in y—your apartment

Mike
there's a great big walk-in closet here and I don't remember if there's one in over here or not

George
mm

Mike
no there isn't
and...

George
th-there's one—there's one

Mike
I can remember—there's one in her bedroom?
or not
yeah there is

George
yeah

Mike
and there's one in here
'n there's also one there's also one in in the bedroom as well
y'know it goes back
so there are three
um...this is actually bigger this apartment is bigger

George
mm-hm

Mike
but it looks a little smaller when you come in cause of the way it's arranged
...I would say this apartment affords you more privacy this apartment here
if you have two people

George
mm-hm

Mike
...so
...ahm this one you could entertain in
with a group of people
in this one it's very difficult to because there's really there's only the main area is kindof narrow
and unless they go into this room in here
in this one here which a party last year Penni did

George
mm-hm

Mike
where people were able to use the kitchen living room go back and forth

George
yeah

Mike
so that's—you could entertain in this one maybe five or six people

George
mm-hm

Mike
...um

George
great

Mike
ok

Appendix B

Family Orders

This appendix includes 14 enumerations from the family data. (One of the 15 speakers was unable to provide any enumeration at all.) Each enumeration reflects one speaker's order of mentioning the family members. If family members were not mentioned by name but referred to collectively, this is indicated in parentheses. The diagram of the family is included here to aid in following the enumerations (figure B.1).

Four names are repeated in the family. Usually speakers did not explicate which family member they meant when they mentioned one of these names; it was assumed to be clear from context. However, I have annotated the enumerations as follows:

Ann the elder is Eugene's wife.

Ann the younger is Margaret's daughter.

David the elder is Alice's son.

David the younger is Diana's son.

Martha the elder is Margaret's daughter.

Martha the younger is Dorothy's daughter.

Andrew the elder is Jeff's son.

Andrew the younger is Dorothy's son.

Some enumerations mention a larger subset of the family than others. This may in part be due to forgetfulness, but is largely due to whether the description was of the family that was attending Thanksgiving that day, or the family that usually attends or has attended in the past (some of the family members mentioned were dead).

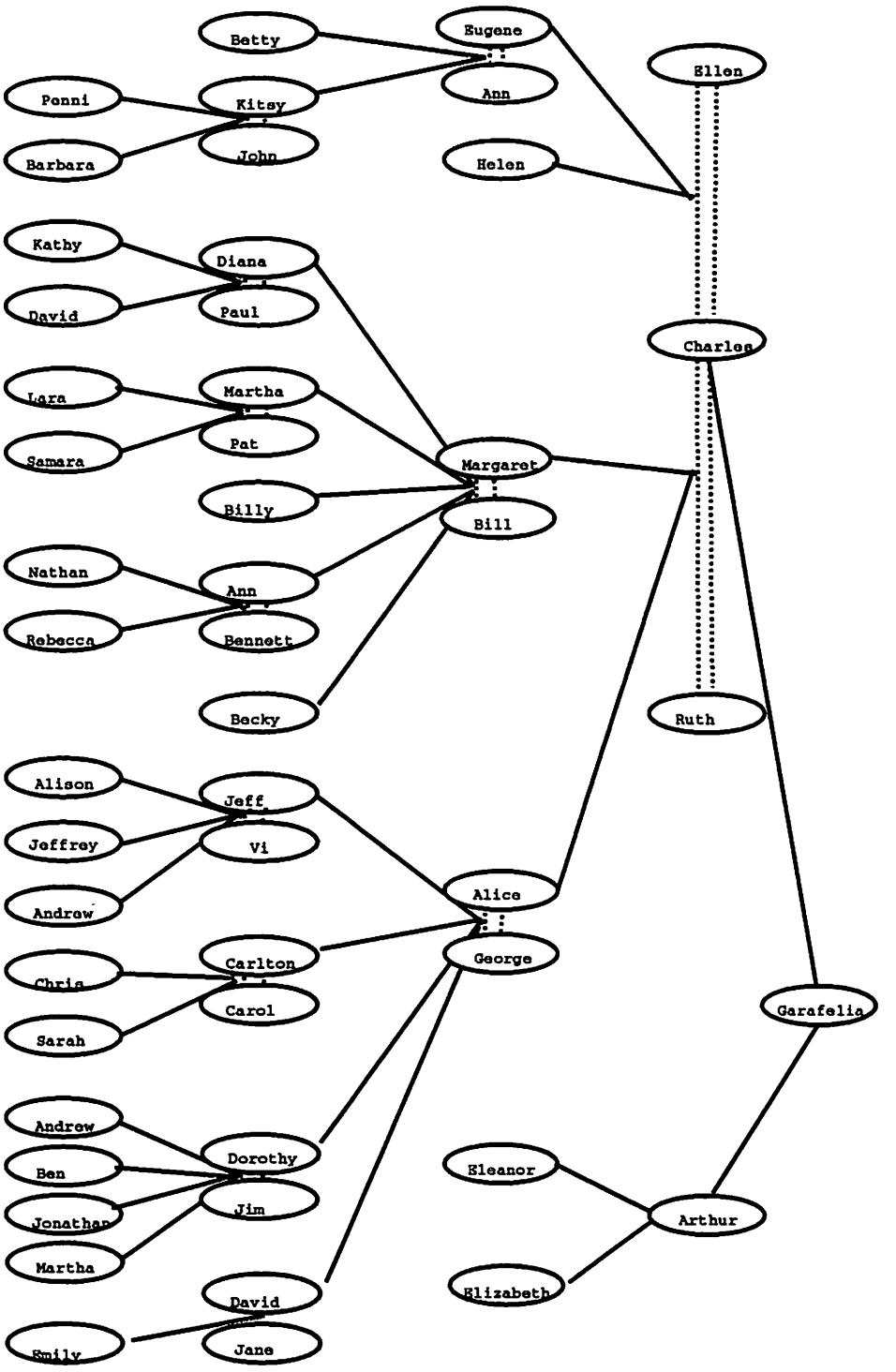

Figure B.1: Family diagram. Double stippled lines indicate spouse relationships; solid lines parent-child.

Speaker 1	Speaker 2
Margaret	Kitsy
Bill	John
Alice	Penni
Eugene	Barbara
Charles	Margaret
Ann (elder)	Bill
Betty	Alice
John	Becky
Penni	Billy
Barbara	Ann (younger)
Arthur	Bennett
Eleanor	Nathan
Elizabeth	Rebecca
Diana	Martha (elder)
Paul	Pat
Kathy	Lara
David (younger)	Samara
Martha (elder)	Diana
Pat	Paul
Lara	Kathy
Samara	David (younger)
Billy	Alice
Ann (younger)	David (elder)
Bennett	Jane
Nathan	Carlton
Rebecca	Carol
Becky	Christopher
Alice	Sarah
Jeff	Jeff
Vi	Vi
Alison	Alison
Jeffrey	Jeffrey
Andrew (elder)	Andrew (elder)
Carlton	Dorothy
Carol	Jim
Christopher	Jonathan
Sarah	Ben
Dorothy	Andrew (younger)
Jim	Martha (younger)
(their children)	Eleanor
David (elder)	Elizabeth
Jane	
Emily	

Speaker 3

Margaret
Bill
Eugene
Charles
Ellen
Penni
Kitsy
Helen
Ruth
Margaret
Alice
Diana
Martha (elder)
Billy
Ann (younger)
Becky
Alice
Jeff
David (elder)
Dorothy
Carlton
Diana
David (younger)
Kathy
Paul
Martha (elder)
Pat
Samara
Lara
Billy
Becky
Ann (younger)
Bennett
Nathan
Rebecca
Jeff
Jeffrey
Andrew (elder)
Alison
Carlton
Carol
Christopher
Sarah
Emily
Jane
David (elder)
Dorothy
Andrew (younger)
(Dorothy's other children)
Eleanor
Elizabeth
Eugene

Speaker 4

Margaret
Alice
Margaret
Bill
Alice
(Alice's children and their spouses)
(their children)
(Margaret's children, grandchildren)
(Penni's family)
Eleanor
Elizabeth

Speaker 5

Margaret
Alice
(their children)
(their grandchildren)
(Kitsy's family)
Eleanor
Elizabeth
Arthur
Charles

Speaker 6

Alice
Margaret
Kitsy
Eugene
Eleanor
Elizabeth
Alice
(Margaret's children)

(their children)
David (elder)
(his family)
Carol
Carleton
(their children)
Jeff
(his children)

Speaker 7

Margaret
Bill
Diana
(spouse and two children)
Bennett
Ann (younger)
(their children)
Kitsy
John
Penni
Barbara
Carlton
David
(their spouses and children)
Alice
Eleanor and Elizabeth

Speaker 8

Margaret
Bill
(their children)
Alice
(her family)
Kitsy
Charles
(Charles's two wives)
Eugene
Eleanor
Elizabeth
Arthur
Diana
Martha
Billy
Ann (younger)
Becky
Diana
Paul
David (younger)
Kathy
Martha
Pat
Lara
Samara

Speaker 9

Diana
Bill
Margaret
Kathy
David (younger)
Paul
Ann (younger)
Bennett
Rebecca
Nathan
Billy
Alice
Carlton
Carol
Christopher
Sarah
David (elder)
Jane
Emily
Kitsy
John
Penni
Barbara
Eleanor
Elizabeth

Speaker 10

Penni
Bill
Margaret
Kitsy

Barbara
John
Billy
Diana
Ann (younger)
Nathan
Rebecca
Diana
Kathy
David (younger)
Margaret
Alice
David (elder)
Jane
Emily
Carlton
Carol
Christopher
Sarah
Eleanor
Elizabeth

Speaker 11

Garafelia
Charles
Arthur
(their children, grandchildren,
great-grandchildren)

Speaker 12

Margaret
Alice
Kitsy
Eugene
Eleanor
Elizabeth

Speaker 13

Margaret
Bill
Eleanor
Elizabeth
Billy
(Margaret's daughters,
their spouses and children)
Eugene
Kitsy
John
(their children)
Betty
Eugene
Eleanor
Elizabeth
Margaret
Alice
Eugene
Charles
Arthur

Speaker 14

Bill
Margaret
Elizabeth
Eleanor
Alice
Diana
Paul
Martha (elder)
Pat
Billy
Ann (younger)
Bennett
Becky
Jeff
Vi
(their children)
Carlton
(his children)
Alice
David (elder)
Jane
Emily
Dorothy
Kitsy
Barbara
Penni

Andrew (elder)
Jeffrey
Alison
Carlton
Christopher
Sara
Emily
Ann (younger)
Nathan
Rebecca
Benjamin
Andrew (younger)
Martha (younger)
Jonathan
Benjamin
Jeff
Jeffrey
Carlton
David (elder)
Dorothy
Kathy
David (younger)
Martha (elder)
Lara
Samara
Billy
Ann (younger)
Nathan
Rebecca
David
Sarah
Rebecca
Christopher
Benjamin
Dorothy
George
Eugene
Helen
Margaret
Alice

Appendix C

Knowledge Base

This appendix includes the **relations**, **objects**, **propertys**, and **directions** in the KB. Also included are the relation *links* between objects and other objects and properties. The lists of links from the family domain have been abbreviated.

```
;;; ================================================================
;;;
;;; The relations
;;;
;;; ================================================================

(defrelation unknown
    lex-item-none :there-insertion set-unknown)
(defrelation is-east-of
    lex-item-none :spatial-deixis-as-pp+be set-is-east-of)
(defrelation is-west-of
    lex-item-none :spatial-deixis-as-pp+be set-is-west-of)
(defrelation is-north-of
    lex-item-none :spatial-deixis-as-pp+be set-is-north-of)
(defrelation is-south-of
    lex-item-none :spatial-deixis-as-pp+be set-is-south-of)
(defsymmetric-relation next-to
    lex-item-none :spatial-deixis-as-pp+be set-next-to)
(defcomputed-relation next-to-same-dir
    compute-next-to-same-dir :spatial-deixis-as-pp+be)
(defcomputed-relation next-to-ortho-dir
    compute-next-to-ortho-dir :spatial-deixis-as-pp+be)
(defcomputed-relation next-to-opp-dir
    compute-next-to-opp-dir :spatial-deixis-as-pp+be)
(defsymmetric-relation next-to-doors
    lex-item-none :spatial-deixis-as-pp+be set-next-to-doors)
(defcomputed-relation next-to-doors-same-dir
    compute-next-to-doors-same-dir :spatial-deixis-as-pp+be)
```

```
(defcomputed-relation next-to-doors-ortho-dir
    compute-next-to-doors-ortho-dir :spatial-deixis-as-pp+be)
(defcomputed-relation next-to-doors-opp-dir
    compute-next-to-doors-opp-dir :spatial-deixis-as-pp+be)
(defrelation faces
    lex-item-face :verb set-faces)
(definverse-relations comprises lex-item-have :verb nil
                      composes lex-item-none :default nil
                      set-comprises)
(definverse-relations contained-by lex-item-none :default nil
                      contains lex-item-have-in-it :verb+pp nil
                      set-contained-by)
(defrelation has-property lex-item-none :default set-property)
(defrelation has-owner lex-item-none  :default set-has-owner)

(defrelation has-child
    lex-item-child :noun+be set-has-child nil has-parent)
(defrelation has-parent
    lex-item-parent :noun+be set-has-parent nil has-child)
(defrelation has-sibling
    lex-item-sibling :noun+be set-has-sibling nil has-sibling)
(defrelation has-spouse
    lex-item-spouse :noun+be set-has-spouse nil has-spouse)
(defrelation has-daughter
    lex-item-daughter :noun+be set-has-daughter has-child has-parent)
(defrelation has-son
    lex-item-son :noun+be set-has-son has-child has-parent)
(defrelation has-mother
    lex-item-mother :noun+be set-has-mother has-parent has-child)
(defrelation has-father
    lex-item-father :noun+be set-has-father has-parent has-child)
(defrelation has-sister
    lex-item-sister :noun+be set-has-sister has-sibling has-sibling)
(defrelation has-brother
    lex-item-brother :noun+be set-has-brother has-sibling has-sibling)
(defrelation has-wife
    lex-item-wife :noun+be set-has-wife has-spouse has-spouse)
(defrelation has-husband
    lex-item-husband :noun+be set-has-husband has-spouse has-spouse)
```

```
;;; ================================================================
;;;
;;; The objects
;;;
;;; ================================================================


(defobject kitchen (room non-path-room) lex-item-kitchen)
(defobject kitchen-sink furnishing lex-item-sink)
(defobject large-kitchen-window structure lex-item-window)
(defobject backyard orientation lex-item-the-backyard)
(defobject flanking-windows structure lex-item-two-flanking-windows)
(defobject refrigerator furnishing lex-item-refrigerator)
(defobject stove furnishing lex-item-stove)
(defobject dishwasher furnishing lex-item-dishwasher)
(defobject small-kitchen-window structure lex-item-window)
(defobject sliding-glass-door (doorway structure)
           lex-item-sliding-glass-door)
(defobject kitchen-wall-with-closet structure lex-item-wall)
(defobject kitchen-closet room lex-item-closet)
(defobject stairwell (path structure) lex-item-stairwell)
(defobject living-room (room non-path-room) lex-item-livingroom)
(defobject large-living-room-window structure lex-item-window)
(defobject short-hallway (path room) lex-item-short-hallway)
(defobject long-hallway (path room) lex-item-long-hallway)
(defobject front-hallway (path room) lex-item-front-hallway)
(defobject side-door (doorway) lex-item-side-door)
(defobject entrance-hallway (path room) lex-item-entrance-hallway)
(defobject ann-room (room non-path-room) lex-item-bedroom)
(defobject claire-room (room non-path-room) lex-item-bedroom)
(defobject penni-room (room non-path-room) lex-item-bedroom)
(defobject bathroom (room non-path-room) lex-item-bathroom)
(defobject ann other "Ann")
(defobject claire other "Claire")

(defobject garafelia family-member "Garafelia" t) ; for proper-name?
(defobject arthur family-member "Arthur" t)
(defobject charles family-member "Charles" t)
(defobject eleanor family-member "Eleanor" t)
(defobject elizabeth family-member "Elizabeth" t)
(defobject eugene family-member "Eugene" t)
(defobject helen family-member "Helen" t)
(defobject margaret family-member "Margaret" t)
(defobject alice family-member "Alice" t)
(defobject betty family-member "Betty" t)
(defobject kitsy family-member "Kitsy" t)
(defobject diana family-member "Diana" t)
(defobject marthaj family-member "Martha" t)
(defobject billy family-member "Billy" t)
```

```
(defobject annj family-member "Ann" t)
(defobject becky family-member "Becky" t)
(defobject jeff family-member "Jeff" t)
(defobject carleton family-member "Carleton" t)
(defobject dorothy family-member "Dorothy" t)
(defobject davidt family-member "David" t)
(defobject penni family-member "Penni" t)
(defobject barbara family-member "Barbara" t)
(defobject kathy family-member "Kathy" t)
(defobject davids family-member "David" t)
(defobject lara family-member "Lara" t)
(defobject samara family-member "Samara" t)
(defobject nathan  family-member "Nathan" t)
(defobject rebecca family-member "Rebecca" t)
(defobject alison family-member "Alison" t)
(defobject Jeffrey family-member "Jeffrey" t)
(defobject andrewt family-member "Andrew" t)
(defobject chris family-member "Chris" t)
(defobject sarah family-member "Sarah" t)
(defobject jonathan family-member "Jonathan" t)
(defobject andrewl family-member "Andrew" t)
(defobject benjamin family-member "Benjamin" t)
(defobject marthal family-member "Martha" t)
(defobject emily family-member "Emily" t)
(defobject john family-member "John" t)
(defobject ellen family-member "Ellen" t)
(defobject ruth family-member "Ruth" t)
(defobject annc family-member "Ann" t)
(defobject bill family-member "Bill" t)
(defobject george family-member "George" t)
(defobject paul family-member "Paul" t)
(defobject pat family-member "Pat" t)
(defobject bennett family-member "Bennett" t)
(defobject vi family-member "Vi" t)
(defobject carol family-member "Carol" t)
(defobject jim family-member "Jim" t)
(defobject jane family-member "Jane" t)
```

```
;;; ================================================================
;;;
;;; The directions
;;;
;;; ================================================================

(defdir north south east west)
(defdir east west north south)
(defdir south north east west)
(defdir west east north south)

;;; ================================================================
;;;
;;; The properties
;;;
;;; ================================================================

(defproperty picture-window-ness lex-item-a-picture-window)
(defproperty largeness lex-item-large")
(defproperty smallness lex-item-small)

(defproperty female lex-item-female)
(defproperty male lex-item-male)

;;; ================================================================
;;;
;;; The links
;;;
;;; ================================================================

(set-next-to-doors side-door entrance-hallway)
(set-next-to side-door entrance-hallway)
(set-next-to-doors kitchen kitchen-closet)
(set-next-to-doors kitchen living-room)
(set-next-to-doors living-room short-hallway)
(set-next-to-doors short-hallway bathroom)
(set-next-to-doors short-hallway ann-room)
(set-next-to-doors short-hallway claire-room)
(set-next-to-doors kitchen long-hallway)
(set-next-to-doors penni-room long-hallway)
(set-next-to-doors living-room long-hallway)
(set-next-to-doors short-hallway long-hallway)
(set-property large-kitchen-window largeness)
(set-property large-kitchen-window picture-window-ness)
(set-comprises large-kitchen-window flanking-windows)
(set-faces large-kitchen-window backyard)
(set-next-to large-kitchen-window sliding-glass-door)
(set-next-to large-kitchen-window stove)
(set-next-to large-kitchen-window kitchen-sink)
```

```
(set-beneath large-kitchen-window kitchen-sink)
(set-next-to stove kitchen-sink)
(set-next-to stove refrigerator)
(set-next-to sliding-glass-door small-kitchen-window)
(set-next-to sliding-glass-door dishwasher)
(set-next-to kitchen-sink dishwasher)
(set-next-to small-kitchen-window kitchen-closet)
(set-property small-kitchen-window smallness)
(set-next-to stairwell kitchen-closet)
(set-property large-living-room-window largeness)
(set-next-to living-room kitchen)
(set-next-to living-room kitchen-closet)
(set-next-to living-room ann-room)
(set-next-to penni-room kitchen)
(set-next-to penni-room long-hallway)
(set-next-to long-hallway short-hallway)
(set-next-to long-hallway living-room)
(set-next-to entrance-hallway living-room)
(set-next-to entrance-hallway kitchen)
(set-next-to front-hallway living-room)
(set-next-to short-hallway living-room)
(set-next-to short-hallway ann-room)
(set-next-to short-hallway claire-room)
(set-next-to short-hallway bathroom)
(set-next-to ann-room bathroom)
(set-next-to claire-room bathroom)
(set-is-south-of kitchen-sink dishwasher)
(set-is-north-of kitchen-sink stove)
(set-is-east-of kitchen-sink large-kitchen-window)
(set-is-south-of large-kitchen-window sliding-glass-door)
(set-is-north-of large-kitchen-window stove)
(set-is-west-of large-kitchen-window kitchen-sink)
(set-is-south-of stove large-kitchen-window)
(set-is-south-of stove kitchen-sink)
(set-is-west-of stove refrigerator)
(set-is-east-of refrigerator stove)
(set-is-north-of dishwasher kitchen-sink)
(set-is-north-of sliding-glass-door large-kitchen-window)
(set-is-west-of sliding-glass-door small-kitchen-window)
(set-is-east-of small-kitchen-window sliding-glass-door)
(set-is-west-of small-kitchen-window kitchen-closet)
(set-is-south-of kitchen-closet small-kitchen-window)
(set-is-west-of kitchen-closet stairwell)
(set-is-west-of kitchen-closet living-room)
(set-is-east-of stairwell kitchen-closet)
(set-is-west-of stairwell living-room)
(set-is-east-of living-room kitchen-closet)
(set-is-east-of living-room kitchen)
(set-is-west-of kitchen living-room)
```

```
(set-is-west-of kitchen kitchen-closet)
(set-is-south-of penni-room kitchen)
(set-is-north-of kitchen penni-room)
(set-is-east-of long-hallway penni-room)
(set-is-west-of penni-room long-hallway)
(set-is-west-of long-hallway short-hallway)
(set-is-north-of long-hallway short-hallway)
(set-is-east-of short-hallway long-hallway)
(set-is-south-of short-hallway long-hallway)
(set-is-north-of living-room short-hallway)
(set-is-south-of short-hallway living-room)
(set-is-south-of bathroom short-hallway)
(set-is-north-of short-hallway bathroom)
(set-is-east-of ann-room short-hallway)
(set-is-east-of ann-room bathroom)
(set-is-west-of short-hallway ann-room)
(set-is-west-of bathroom ann-room)
(set-is-east-of short-hallway claire-room)
(set-is-east-of bathroom claire-room)
(set-is-west-of claire-room short-hallway)
(set-is-west-of claire-room bathroom)
(set-is-east-of ann-room claire-room)
(set-is-west-of claire-room ann-room)
(set-is-north-of side-door entrance-hallway)
(set-is-south-of entrance-hallway side-door)
(set-is-west-of kitchen entrance-hallway)
(set-is-east-of entrance-hallway kitchen)
(set-is-west-of entrance-hallway living-room)
(set-is-east-of living-room entrance-hallway)
(set-is-east-of front-hallway living-room)
(set-is-west-of living-room front-hallway)
(set-contained-by kitchen large-kitchen-window)
(set-contained-by kitchen refrigerator)
(set-contained-by kitchen stove)
(set-contained-by kitchen kitchen-sink)
(set-contained-by kitchen dishwasher)
(set-contained-by kitchen small-kitchen-window)
(set-contained-by kitchen sliding-glass-door)
(set-contained-by living-room large-living-room-window)
(set-has-owner ann-room ann)
(set-has-owner claire-room claire)
(set-has-owner penni-room penni)

(set-property garafelia female)
(set-property arthur male)
```

etc.

```
(set-has-mother arthur garafelia)
(set-has-son garafelia arthur)
```

etc.

```
(set-has-brother arthur charles)
(set-has-brother arthur charles)
```

etc.

```
(set-has-wife charles ellen)
(set-has-husband ellen charles)
```

etc.

Appendix D

Trace of Execution

This appendix shows excerpts from traces of Salix building texts in both the house and the family domains. Included is the beginning of the trace of a house domain text (see chapter 5) and the end of the trace of a family domain text (see chapter 4). The functions traced are the metastrategy **choose** and the function **apply-strategy** (see chapter 3). The trace of **choose** shows the current node and candidates at each multiple match impasse. The trace of **apply-strategy** shows all strategies that are applied. The output text also appears in the trace.

Trace of the beginning of a text in the house domain.

```
(describe-house side-door)

--->CHOOSE
    Node:       #<object side-door>
    Candidates: ((say-first-object #<object side-door>)
                 (say-object #<object side-door>)
                 (find-next-to-say #<object side-door>
                                   #<object entrance-hallway>
                                   #<relation next-to>)
                 (find-next-to-say #<object side-door>
                                   #<object entrance-hallway>
                                   #<relation next-to-doors>))

--->APPLY-STRATEGY
    [(choose (say-first-object #<object side-door>))]

--->APPLY-STRATEGY
    [(say-first-object #<object side-door>)]

OK
we can start at the side door
--->CHOOSE
    Node:       #<object side-door>
```

```
    Candidates: ((find-next-to-say #<object side-door>
                                   #<object entrance-hallway>
                                   #<relation next-to>)
                 (find-next-to-say #<object side-door>
                                   #<object entrance-hallway>
                                   #<relation next-to-doors>))

--->APPLY-STRATEGY
    [(choose
      (find-next-to-say #<object side-door> #<object entrance-hallway>
                        #<relation next-to-doors>))]

--->APPLY-STRATEGY
    [(find-next-to-say #<object side-door> #<object entrance-hallway>
                       #<relation next-to-doors>)]

--->CHOOSE
    Node:       #<object entrance-hallway>
    Candidates: ((say-object #<object entrance-hallway>)
                 (find-next-to-say #<object entrance-hallway>
                                   #<object living-room>
                                   #<relation next-to>)
                 (find-next-to-say #<object entrance-hallway>
                                   #<object kitchen>
                                   #<relation next-to>)
                 (find-next-to-say #<object entrance-hallway>
                                   #<object kitchen>
                                   #<relation next-to-ortho-dir>)
                 (find-next-to-say #<object entrance-hallway>
                                   #<object living-room>
                                   #<relation next-to-ortho-dir>))

--->APPLY-STRATEGY
    [(choose (say-object #<object entrance-hallway>))]

--->APPLY-STRATEGY
    [(say-object #<object entrance-hallway>)]

then there's the entrance hallway
--->CHOOSE
    Node:       #<object entrance-hallway>
    Candidates: ((find-next-to-say #<object entrance-hallway>
                                   #<object living-room>
                                   #<relation next-to>)
                 (find-next-to-say #<object entrance-hallway>
                                   #<object kitchen>
                                   #<relation next-to>)
                 (find-next-to-say #<object entrance-hallway>
                                   #<object kitchen>
```

```
                                #<relation next-to-ortho-dir>)
                (find-next-to-say #<object entrance-hallway>
                                  #<object living-room>
                                  #<relation next-to-ortho-dir>))

--->APPLY-STRATEGY
    [(choose
      (find-next-to-say #<object entrance-hallway> #<object kitchen>
                        #<relation next-to-ortho-dir>))]

--->APPLY-STRATEGY
    [(find-next-to-say #<object entrance-hallway> #<object kitchen>
                       #<relation next-to-ortho-dir>)]

--->CHOOSE
    Node:       #<object kitchen>
    Candidates: ((say-elaborating-relation
                  #<object kitchen>
                  ((#<object large-kitchen-window> .
                    #<relation contains>)
                   (#<object refrigerator> .
                    #<relation contains>)
                   (#<object stove> .
                    #<relation contains>)
                   (#<object kitchen-sink> .
                    #<relation contains>)
                   (#<object dishwasher> .
                    #<relation contains>)
                   (#<object small-kitchen-window> .
                    #<relation contains>)
                   (#<object sliding-glass-door> .
                    #<relation contains>))
                  #<relation contains>)
                 (find-salient-object #<object kitchen>
                                      #<object large-kitchen-window>)
                 (say-object #<object kitchen>)
                 (find-next-to-say #<object kitchen> #<object living-room>
                                   #<relation next-to>)
                 (find-next-to-say #<object kitchen> #<object penni-room>
                                   #<relation next-to>)
                 (find-next-to-say #<object kitchen> #<object penni-room>
                                   #<relation next-to-ortho-dir>)
                 (find-next-to-say #<object kitchen> #<object living-room>
                                   #<relation next-to-opp-dir>)
                 (find-next-to-say #<object kitchen>
                                   #<object kitchen-closet>
                                   #<relation next-to-doors>)
                 (find-next-to-say #<object kitchen> #<object living-room>
                                   #<relation next-to-doors>)
```

```
                (find-next-to-say #<object kitchen> #<object long-hallway>
                                  #<relation next-to-doors>)
                (find-next-to-say #<object kitchen> #<object living-room>
                                  #<relation next-to-doors-opp-dir>))

--->CHOOSE
    Node:       <multiple-match>
    Candidates: ((say-multiple-sweep #<object kitchen>
                                     #<object large-kitchen-window>
                                     ((#<object sliding-glass-door>
                                       #<object small-kitchen-window>)
                                      (#<object stove>
                                       #<object refrigerator>)
                                      (#<object kitchen-sink>
                                       #<object dishwasher>)))
                (choose (say-object #<object kitchen>)))

--->APPLY-STRATEGY
    [(choose
      (say-multiple-sweep #<object kitchen> #<object large-kitchen-window>
                          ((#<object sliding-glass-door>
                            #<object small-kitchen-window>)
                           (#<object stove> #<object refrigerator>)
                           (#<object kitchen-sink>
                            #<object dishwasher>))))]

--->APPLY-STRATEGY
    [(say-multiple-sweep #<object kitchen> #<object large-kitchen-window>
                         ((#<object sliding-glass-door>
                           #<object small-kitchen-window>)
                          (#<object stove> #<object refrigerator>)
                          (#<object kitchen-sink> #<object dishwasher>)))]

then
in the kitchen
there is a window
--->CHOOSE
    Node:       #<object large-kitchen-window>
    Candidates: ((say-elaborating-relation #<object large-kitchen-window>
                                           ((#<object backyard> .
                                             #<relation faces>))
                                           #<relation faces>)
                 (say-elaborating-relation #<object large-kitchen-window>
                                           ((#<object flanking-windows> .
                                             #<relation comprises>))
                                           #<relation comprises>)
```

```
                (say-property #<object large-kitchen-window>
                              #<property largeness>)
                (say-property #<object large-kitchen-window>
                              #<property picture-window-ness>))

--->APPLY-STRATEGY
    [(choose
      (say-property #<object large-kitchen-window>
                    #<property picture-window-ness>))]

--->APPLY-STRATEGY
    [(say-property #<object large-kitchen-window>
                   #<property picture-window-ness>)]

which is a picture window
```

Trace of the latter part of a text in the family domain.

```
--->APPLY-STRATEGY
    [(jump #<object chris>)]

--->CHOOSE
    Node:       #<object chris>
    Candidates: ((say-elaborating-relation #<object chris>
                                           ((#<object carol> .
                                             #<relation has-mother>)
                                            (#<object carleton> .
                                             #<relation has-father>))
                                           #<relation has-parent>)
                 (say-elaborating-relation #<object chris>
                                           ((#<object sarah> .
                                             #<relation has-sister>))
                                           #<relation has-sister>)
                 (say-object #<object chris>)
                 (find-next-to-say #<object chris> #<object sarah>
                                   #<relation has-sister>)
                 (find-next-to-say #<object chris> #<object carol>
                                   #<relation has-mother>))

--->APPLY-STRATEGY
    [(choose (say-object #<object chris>))]

--->APPLY-STRATEGY
    [(say-object #<object chris>)]

and then there's Chris
--->CHOOSE
    Node:       #<object chris>
    Candidates: ((say-elaborating-relation #<object chris>
                                           ((#<object carol> .
                                             #<relation has-mother>)
                                            (#<object carleton> .
                                             #<relation has-father>))
                                           #<relation has-parent>)
                 (say-elaborating-relation #<object chris>
                                           ((#<object sarah> .
                                             #<relation has-sister>))
                                           #<relation has-sister>)
                 (find-next-to-say #<object chris> #<object sarah>
                                   #<relation has-sister>)

                 (find-next-to-say #<object chris> #<object carol>
                                   #<relation has-mother>))
```

```
--->APPLY-STRATEGY
    [(choose
      (say-elaborating-relation #<object chris>
                                ((#<object sarah> .
                                  #<relation has-sister>))
                                #<relation has-sister>))]

--->APPLY-STRATEGY
    [(say-elaborating-relation #<object chris>
                               ((#<object sarah> . #<relation has-sister>))
                               #<relation has-sister>)]

Chris's sister is Sarah
--->APPLY-STRATEGY
    [(noop-on-inhibit-recursion)]

--->CHOOSE
    Node:       #<object chris>
    Candidates: ((say-elaborating-relation #<object chris>
                                           ((#<object carol> .
                                             #<relation has-mother>)
                                            (#<object carleton> .
                                             #<relation has-father>))
                                           #<relation has-parent>)
                 (find-next-to-say #<object chris> #<object carol>
                                   #<relation has-mother>))

--->APPLY-STRATEGY
    [(choose
      (say-elaborating-relation #<object chris>
                                ((#<object carol> . #<relation has-mother>)
                                 (#<object carleton> .
                                  #<relation has-father>))
                                #<relation has-parent>))]

--->APPLY-STRATEGY
    [(say-elaborating-relation #<object chris>
                               ((#<object carol> . #<relation has-mother>)
                                (#<object carleton> .
                                 #<relation has-father>))
                               #<relation has-parent>)]

and his parents are Carol
--->APPLY-STRATEGY
    [(noop-on-inhibit-recursion)]

and Carleton
--->APPLY-STRATEGY
    [(noop-on-inhibit-recursion)]
```

```
--->APPLY-STRATEGY
    [(jump #<object jane>)]

--->CHOOSE
    Node:       #<object jane>
    Candidates: ((say-elaborating-relation #<object jane>
                                           ((#<object davidt> .
                                             #<relation has-husband>))
                                           #<relation has-husband>)
                 (say-elaborating-relation #<object jane>
                                           ((#<object emily> .
                                             #<relation has-daughter>))
                                           #<relation has-daughter>)
                 (say-object #<object jane>)
                 (find-next-to-say #<object jane> #<object emily>
                                   #<relation has-daughter>))

--->APPLY-STRATEGY
    [(choose (say-object #<object jane>))]

--->APPLY-STRATEGY
    [(say-object #<object jane>)]

then there's Jane
--->CHOOSE
    Node:       #<object jane>
    Candidates: ((say-elaborating-relation #<object jane>
                                           ((#<object davidt> .
                                             #<relation has-husband>))
                                           #<relation has-husband>)
                 (say-elaborating-relation #<object jane>
                                           ((#<object emily> .
                                             #<relation has-daughter>))
                                           #<relation has-daughter>)
                 (find-next-to-say #<object jane> #<object emily>
                                   #<relation has-daughter>))

--->APPLY-STRATEGY
    [(choose
      (say-elaborating-relation #<object jane>
                                ((#<object emily> .
                                  #<relation has-daughter>))
                                #<relation has-daughter>))]
```

```
--->APPLY-STRATEGY
    [(say-elaborating-relation #<object jane>
                               ((#<object emily> .
                                 #<relation has-daughter>))
                               #<relation has-daughter>)]

her daughter is Emily
--->APPLY-STRATEGY
    [(noop-on-inhibit-recursion)]

--->APPLY-STRATEGY
    [(say-elaborating-relation #<object jane>
                               ((#<object davidt> .
                                 #<relation has-husband>))
                               #<relation has-husband>)]

and her husband is David
--->APPLY-STRATEGY
    [(noop-on-inhibit-recursion)]

--->APPLY-STRATEGY
    [(say-all-done)]

that's it
```

Bibliography

Agre, P. and D. Chapman (1987), "Pengi: An Implementation of a Theory of Activity." *Proceedings AAAI*, pp 268-272.

Allen, J. (1984), "Toward a General Theory of Time and Action." *Artificial Intelligence*, **23**(2), pp 123-154.

Anderson, S. and E. Keenan (1985), "Deixis." In T. Shopen, ed., *Language Typology and Syntactic Description, Volume III: Grammatical Categories and the Lexicon*, Cambridge University Press.

Anderson, S., P. Sibun, D. Forster, and B. Woolf (1989), *Plan-Based Paragraph Comprehension.* COINS Technical Report 89-121, Department of Computer and Information Science, University of Massachusetts.

Appelt, D. (1985), "Planning English Referring Expressions." *Artificial Intelligence* **26**, pp 1-33.

Aristotle (1926), *The Art of Rhetoric.* J. Freese, tr., The Loeb Classical Library.

Atkinson, J. and J. Heritage (1984), *Structures of Social Action: Studies in Conversation Analysis.* Cambridge University Press.

Becker, J. (1974), *The Phrasal Lexicon.* Bolt Beranek and Newman Inc. Report No. 3081, A. I. Report No. 28.

Cawsey, A. (1990), "Generating Explanatory Discourse." In R. Dale, C. Mellish, and M. Zock, eds., *Current Research in Natural Language Generation*, Academic Press, pp 75-101.

Cawsey, A. (1991), "Planning Interactive Explanations." Submitted to the *International Journal of Man-Machine Studies.*

Cohen, P. and C. Perrault (1979), "Elements of a Plan-Based Theory of Speech Acts." *Cognitive Science* **3**(3), pp 177-212.

Conklin, E. (1983), *Data-Driven Indelible Planning of Discourse Generation Using Salience.* COINS Technical Report 83.13, University of Massachusetts.

Cullingford, R. (1986), *Natural Language Processing.* Rowman & Littlefield.

Dale, R. (1988), *Generating Referring Expressions in a Domain of Objects and Processes.* PhD thesis, University of Edinburgh.

Dale, R. (1990), "Generating Recipes: An Overview of Epicure." In R. Dale, C. Mellish, and M. Zock, eds., *Current Research in Natural Language Generation*, Academic Press, pp 229-255.

Davis, R. (1980), "Meta-rules: Reasoning about Control." *Artificial Intelligence* **15**, pp 179-222.

Defrise, C. and S. Nirenburg (1990), "Aspects of Text Meaning: Speaker Attitudes in Language Generation." *Proceedings of the Fifth International Workshop on Natural Language Generation*, pp 150-155, Linden Hall, Dawson, PA.

De Smedt, K. (1990), *Incremental Sentence Generation.* Nijmegen Institute for Cognition Research and Information Technology, Technical Report 90-01.

De Smedt, K. and G. Kempen (1987), "Incremental Sentence Production, Self-correction, and Coordination." In G. Kempen, ed, *Natural Language Generation: New Results in Artificial Intelligence, Psychology and Linguistics*, pp 365-376. Martinus Nijhoff Publishers.

Doyle, J. (1980), *A Model For Deliberation, Action, and Introspection.* MIT AI Technical Report 581.

Ehrich, V. and C. Koster (1983), "Discourse Organization and Sentence Form: The Structure of Room Descriptions in Dutch." *Discourse Processes* **6**, pp 169-195.

Ehrlich, K. and P. Johnson-Laird (1982), "Spatial Descriptions and Referential Continuity." *Journal of Verbal Learning and Verbal Behavior.* **21**(3), pp 296-306.

Firby, R. (1987), "An Investigation into Reactive Planning in Complex Domains." *Proceedings of the Sixth National Conference on Artificial Intelligence*, Seattle, pp 202-206.

Forbus, K. (1985), "Qualitative Process Theory." In D. Bobrow, ed., *Qualitative Reasoning about Physical Systems*, MIT Press, pp 85-168.

Forgy, C. (1981), *On the Efficient Implementation of Production Systems.* PhD thesis, Department of Computer Science, Carnegie-Mellon University.

Forster, D. (1989), "Generating Temporal Expressions in Natural Language." In *Proceedings of the 11th Meeting of the Cognitive Science Society*, pp 259-266.

Franklin, N. and B. Tversky (1990), "Searching Imagined Environments." *Journal of Experimental Psychology, General.* **119**(1), pp 63-76.

Fromkin, V. (1971), "The Non-anomalous Nature of Anomalous Utterances." *Language* **47**(1), pp 27-52.

Garrett, M. (1975), "The Analysis of Sentence Production." *Psychology of Learning and Motivation*, Volume 9, pp 133-177.

Georgeff, M. and A. Lansky (1987), "Reactive reasoning and planning." *Proceedings of the Sixth National Conference on Artificial Intelligence*, Seattle, pp 677-682.

Goldman, N. (1975), "Conceptual Generation." In R. Schank, *Conceptual Information Processing.* North Holland.

Goodwin, C. (1981), *Conversational Organization: Interaction Between Speakers and Hearers.* Academic Press.

Grosz, B. and C. Sidner (1986), "Attention, Intentions, and the Structure of Discourse." *Computational Linguistics*, (**12**)3.

Halliday, M. (1976), *System and Function in Language.* Oxford University Press.

Herskovits, A. (1986), *Language and Spatial Cognition: An Interdisciplinary Study of the Prepositions in English.* Cambridge University Press.

Hobbs, J. (1985), "On the Coherence and Structure of Discourse." Report No. CSLI-85-37, Center for the Study of Language and Information, Stanford University.

Hovy, E. (1987) *Generating Natural Language Under Pragmatic Constraints.* Technical Report YALEU/CSD/RR #521, Yale University.

Hovy, E. (1988), "Two Types of Planning in Language Generation." In *Proceedings of the 26th Annual Meeting of the Association for Computational Linguistics.*

Hovy, E. (1990a), "Parsimonious and Profligate Approaches to the Question of Discourse Structure Relations." In *Proceedings of the Fifth International Workshop on Natural Language Generation*, pp 128-136, Linden Hall, Dawson, PA.

Hovy, E. (1990b), "Unresolved Issues In Paragraph Planning." In R. Dale, C. Mellish, and M. Zock, eds., *Current Research in Natural Language Generation*, Academic Press, pp 17-45.

Hovy, E. and K. McCoy (1989), "Focusing Your RST: A Step toward Generating Coherent Multisentential Text." In *Proceedings of the 11th Meeting of the Cognitive Science Society*, pp 667-674.

Joshi, A. (1987), "The relevance of tree adjoining grammar to generation." In G. Kempen, ed, *Natural Language Generation: New Results in Artificial Intelligence, Psychology and Linguistics*, pp 233-252. Martinus Nijhoff Publishers.

Kaplan, R. and J. Bresnan (1982), "Lexical Functional Grammar: a formal system for grammatical representation." In J. Bresnan, ed, *The Mental Representation of Grammatical Relations.* MIT Press.

Kass, R., and T. Finin (1988), "Modeling the User in Natural Language Systems." *Computational Linguistics*, (**14**)3, pp 5-22.

Kay, M. (1979), "Functional grammar." *Proceedings of the Fifth Annual Meeting of the Berkeley Linguistic Society.* pp 142-158.

Kempen, G. & E. Hoenkamp (1987), "An incremental procedural grammar for sentence formulation." *Cognitive Science* **11**, pp 201-258.

Laird, J., A. Newell, and P. Rosenbloom (1987), "Soar: An Architecture for General Intelligence." *Artificial Intelligence*, **33**, pp 1-64.

Lenat, D. and R. Guha (1990), *Building Large Knowledge-Based Systems.* Addison-Wesley.

Levelt, W. (1982), "Linearization in Describing Spatial Networks." In S. Peters and E. Saarin, eds, *Processes, Beliefs, and Questions*, pp 199-220.

Levelt, W. (1989), *Speaking: From Intention to Articulation.* MIT Press.

Levinson, S. (1983), *Pragmatics.* Cambridge University Press.

Lin, L., R. Simmons, and C. Fedor (1989), *Experience with a Task Control Architecture for Mobile Robots.* Carnegie Mellon University Robotics Institute Technical Report 89-29.

Linde, C. (1974), *The Linguistic Encoding of Spatial Information.* Doctoral Dissertation, Columbia University.

Linde, C., and J. Goguen (1978), "Structure of Planning Discourse." *Journal of Social and Biological Structures*, **1**, pp 219-251.

Luff, P., N. Gilbert, and D. Frohlich (1990), *Computers and Conversation.* Academic Press.

Mann, W. (1983a), *An Overview of the Nigel Text Generation Grammar.* Report ISI/RR-83-113, Information Sciences Institute.

Mann, W. (1983b) *An Overview of the Penman Text Generation System.* Report ISI/RR-83-114, Information Sciences Institute.

Mann, W., M. Bates, B. Grosz, D. McDonald, K. McKeown, and W. Swartout (1981), *Text Generation: The State of the Art and the Literature.* Report ISI/RR-81-101, Information Sciences Institute.

Mann, W. and S. Thompson (1987), *Rhetorical Structure Theory: A Theory of Text Organization.* Report ISI/RS-87-190, Information Sciences Institute.

Maybury, M. (1990), "Using Discourse Focus, Temporal Focus, and Spatial Focus to Generate Multisentential Text." *Proceedings of the Fifth International Workshop on Natural Language Generation*, pp 70-78, Linden Hall, Dawson, PA.

McDermott, D. (1978), "Planning and Acting." *Cognitive Science.* **2**, pp 71-109.

McDonald, D. and J. Pustejovsky (1985), "Description-Directed Natural Language Generation." *Proceedings of the Ninth International Joint Conference on Artificial Intelligence*, Los Angeles.

McDonald, D., M. Vaughan, and J. Pustejovsky (1987), "Factors contributing to efficiency in natural language generation." In G. Kempen, ed, *Natural Language Generation: New Results in Artificial Intelligence, Psychology and Linguistics*, pp 219-230. Martinus Nijhoff Publishers.

McKeown, K. (1985), *Text generation: Using discourse strategies and focus constraints to generate natural language text.* Cambridge University Press.

Meehan, J. (1977), "TALE-SPIN, an interactive program that writes stories." In *Proceedings of the Fifth International Joint Conference on Artificial Intelligence.*

Meteer, M. (1990), "Abstract Linguistic Resources for Text Planning." *Proceedings of the Fifth International Workshop on Natural Language Generation*, pp 62-69, Linden Hall, Dawson, PA.

Meteer, M., D. McDonald, S. Anderson, D. Forster, L. Gay, A. Huettner, and P. Sibun (1987), Mumble-*86: Design and Implementation.* COINS Technical Report 87-87, University of Massachusetts, 1987.

Miller, G. and P. Johnson-Laird (1976), *Language and Perception.* Belknap Press.

Moore, J. and W. Swartout (1989), "A Reactive Approach to Explanation." In *Proceedings of the International Joint Conference on Text Generation IJCAI.*

Newell, A. (1973), "Production Systems: Models of Control Structures." In W. Chase, ed., *Visual Information Processing.* Academic Press, pp 463-526.

Newell, A. (1990), *Unified Theories of Cognition.* Harvard University Press.

Newell, A., J. Shaw, and H. Simon (1958), "Elements of a theory of human problem solving." *Psychological Review*, **65**, pp 151-166.

Newell, A. and H. Simon (1963), "GPS, a program that simulates human thought." In E. Feigenbaum and J. Feldman, eds, *Computers and Thought*, McGraw-Hill.

Nirenburg, S., R. McCardell, E. Nyberg, P. Werner, S. Huffman, E. Kenschaft, and I. Nirenburg (1988), *DIOGENES-88,* CMU Technical Report CMU-CMT-88-107.

Palmer, M. and T. Finin (1990), "Workshop on the Evaluation of Natural Language Processing Systems." *Computational Linguistics* **16**(3), pp 175-181.

Paris, C. (1988), "Tailoring Object Descriptions to a User's Level of Expertise." *Computational Linguistics* **14**(3), pp 64-78.

Pattabhiraman, T. and N. Cercone (1990), "Section: Salience, Relevance and the Coupling between Domain-Level Tasks and Text Planning." In *Proceedings of the Fifth International Workshop on Natural Language Generation*, pp 79-86, Linden Hall, Dawson, PA.

Perelman, C. and L. Olbrechts-Tyteca (1969), *The New Rhetoric: A Treatise on Argumentation.* J. Wilkinson and P. Weaver, tr., University of Notre Dame Press.

Pierrehumbert, J. (1980), *The Phonology and Phonetics of English Intonation.* PhD Thesis, Massachusetts Institute of Technology.

Poizner, H., E. Klima, and U. Bellugi (1987), *What the Hands Reveal about the Brain.* MIT Press.

Pustejovsky, J. and S. Nirenburg (1987), "Lexical Selection in the Process of Language Generation." *Proceedings of the 25th Annual Meeting of the Association for Computational Linguistics.* pp 201-206.

Rambow, O. (1990), "Domain Communication Knowledge." In *Proceedings of the Fifth International Workshop on Natural Language Generation*, pp 87-94, Linden Hall, Dawson, PA.

Rambow, O., R. Kittredge, and T. Korelsky (submitted), "Domain Communication Knowledge." Invited submission to *Computational Intelligence: Special Issue on Natural Language Generation*, Volume **7**(4), December 1991.

Reichman, R., *Getting Computers to Talk Like You and Me.* MIT Press, 1985.

Reiter, E. (1990), "A New Model for Lexical Choice for Open-Class Words." In *Proceedings of the Fifth International Workshop on Natural Language Generation*, pp 23-30, Linden Hall, Dawson, PA.

Retz-Schmidt, G. (1986), *Deictic and Intrinsic Use of Spatial Prepositions: A Multidisciplinary Comparison.* VITRA Memo 13.

Retz-Schmidt, G. (1988), "Various Views on Spatial Prepositions." *AI Magazine*, **9**(2), pp 95-105.

Rubinoff, R. (1986), "Adapting Mumble: Experience with Natural Language Generation." In *Proceeding of AAAI-86*, pp 799-805.

Russell, G., S. Warwick, and J. Carroll (1990), "Asymmetry in Parsing and Generation with Unification Grammars: Case Studies from ELU." *Proceedings of the 28th Annual Meeting of the Association for Computational Linguistics.* pp 205-211.

Sacerdoti, E. (1977), *A Structure for Plans and Behavior.* Elsevier.

Shanon, B. (1984), "Room Descriptions." *Discourse Processes* **7**, pp 225-255.

Sibun, P. (1987), *Apt: A System to Direct and Control Natural Language Generation.* COINS Technical Report 87-42, Department of Computer and Information Science, University of Massachusetts.

Sibun, P. (1990), "The Local Organization of Text." In *Proceedings of the Fifth International Workshop on Natural Language Generation*, pp 120-127, Linden Hall, Dawson, PA.

Sibun, P. and A. Huettner (1989), *Spatial Deixis in Generating Descriptions.* COINS Technical Report 89-34, Department of Computer and Information Science, University of Massachusetts.

Sidner, C. (1979), *Towards a Computational Theory of Definite Anaphora Comprehension in English Discourse.* Report AITR 537, Massachusetts Institute of Technology.

Simmons, R. and J. Slocum (1972), "Generating English Discourse from Semantic Networks." *Communications of the ACM*, Vol. 12, No. 5, pp 891-905.

Simon, H. (1970), *The Sciences of the Artificial.* The MIT Press.

Small, S. and C. Rieger (1982), "Parsing and Comprehending with Word Experts (a theory and its realization)." In W. Lehnert and M. Ringle, eds., *Strategies for Natural Language Processing.* LEA Publishing, pp 89-147.

Sondheimer, N. and B. Nebel (1986), "A Logical-form and Knowledge-base Design for Natural Language Generation." In *Proceedings of the Fifth National Conference on Artificial Intelligence.*

Strzalkowski, T. and P. Peng (1990), "Automated Inversion of Logic Grammars for Generation." *Proceedings of the 28th Annual Meeting of the Association for Computational Linguistics.* pp 212-219.

Ullmer-Ehrich, V. (1982), "The Structure of Living Space Descriptions." In R. Jarvella and W. Klein, eds., *Speech, Place, and Action*, John Wiley & Sons, Ltd.

Taylor, H. and B. Tversky (1990), *Spatial Descriptions and Depictions.* Paper presented at the meetings of the Psychonomic Society, New Orleans, November, 1990.

The Penman Project (1989), *The* Penman *Documentation: User Guide, Primer, Reference Manual, and Nigel Manual.* Technical Report USC/ISI, Information Sciences Institute.

Thompson, H. (1977), "Strategy and Tactics: A Model for Language Production." *Papers from the 13th Regional Meeting of the Chicago Linguistics Society*, pp 651-658.

Tversky, B. (1990), *Induced Pictorial Representations.* Report to the AFOSR.

Wahlster, W., E. Andre, S. Bandyopadhyay, W. Graf, and T. Rist (1991), "WIP: The Coordinated Generation of Multimodal Presentations from a Common Representation." In O. Stock, J. Slack, and A. Ortony, eds., *Computational Theories of Communication and their Applications.* Springer-Verlag.

Waterman, D. and F. Hayes-Roth, eds. (1978), *Pattern-Directed Inference Systems.* Academic Press.

Webber, B. (1983), "So What Do We Talk About Now?" In M. Brady and R. Berwick, Eds., *Computational Models of Discourse*, The MIT Press, pp 331-371.

Webber, B. (1987), "Event Reference." In *Position Papers for TINLAP-3: Theoretical Issues in Natural Language Processing-3*, Las Cruces, NM, pp 137-142.

Webber, B. (1988), "Discourse Anaphora." In *Proceedings of the 26th Annual Meeting of the Association for Computational Linguistics*, Buffalo, pp 113-122.