May 1990

Natural Language Generation as an Intelligent Activity (Proposal for Dissertation Research)

Robert Rubinoff
University of Pennsylvania

University of Pennsylvania Department of Computer and Information Science Technical Report No. MS-CIS-90-32.

Natural Language Generation as an Intelligent Activity (Proposal for Dissertation Research)

Abstract

In this proposal, I outline a generator conceived of as part of a general intelligent agent. The generator's task is to provide the overall system with the ability to use communication in language to serve its purposes, rather than to simply encode information in language. This requires that generation be viewed as a kind of goal-directed action that is planned and executed in a dynamically changing environment. In addition, the generator must not be dependent on domain or problem-specific information but rather on a general knowledge base .that it shares with the overall system. These requirements have specific consequences for the design of the generator and the representation it uses. In particular, the text planner and the low-level linguistic component must be able to interact and negotiate over decisions that involve both high-level and low-level constraints. Also, the knowledge representation must allow for the varying perspective that an intelligent agent will have on the things it talks about; the generator must be able to appropriately vary how it describes things as the system's perspective on them changes. The generator described here will demonstrate how these ideas work in practice and develop them further.

Comments

University of Pennsylvania Department of Computer and Information Science Technical Report No. MS-CIS-90-32.

Natural Language Generation
As An Intelligent Activity

Proposal For Dissertation Research

MS-CIS-90-32
LINC LAB 173

Robert Rubinoff

Department of Computer and Information Science
School of Engineering and Applied Science
University of Pennsylvania
Philadelphia, PA 19104

May 1990

Natural Language Generation as an Intelligent Activity

Proposal for Dissertation Research

Robert Rubinoff[1]

May 23, 1990

[1]This research was partially supported by NSF grant DCR84-10413, Ben Franklin Partnership grants 07.317RUE and 06-312RV, DARPA grant N00014-85-K0018, and ARO grant DAAL03-89-C-0031PRI.

Contents

List of Figures

Abstract

In this proposal, I outline a generator conceived of as part of a general intelligent agent. The generator's task is to provide the overall system with the ability to use communication in language to serve its purposes, rather than to simply encode information in language. This requires that generation be viewed as a kind of goal-directed action that is planned and executed in a dynamically changing environment. In addition, the generator must not be dependent on domain or problem-specific information but rather on a general knowledge base that it shares with the overall system. These requirements have specific consequences for the design of the generator and the representation it uses. In particular, the text planner and the low-level linguistic component must be able to interact and negotiate over decisions that involve both high-level and low-level constraints. Also, the knowledge representation must allow for the varying perspective that an intelligent agent will have on the things it talks about; the generator must be able to appropriately vary how it describes things as the system's perspective on them changes. The generator described here will demonstrate how these ideas work in practice and develop them further.

Chapter 1

Generation and Intelligent Systems

1.1 Goals and Motivations

The principal aim of the work proposed here is to develop a natural language generator that is conceived of as an integral part of an intelligent system.

That is, the task of the generator is to provide the overall system with the ability to use linguistic communication to serve its purposes. This contrasts with the more conventional view of the generator as an independent component whose task is to translate the information it is given into some natural language. This contrast can sometimes be subtle, but it leads to radical differences in the design of the generator and the knowledge representations that it uses. This proposal develops some of the consequences of the integrated intelligent system view and sketches out a proposed generator that embodies it.

The contrast between the approach proposed here and the independent component view is not primarily over the organization of an intelligent system but rather over how the components of the system interact and work together in the course of generating utterances. The generator can still be a discrete component, but it must allow its work to be integrated with the rest of what the system is doing. In the conventional approach, the system simply hands over a request to the generator and then assumes that the generator will carry it out properly. The integrated approach, in contrast, provides for continued involvement of the overall system when needed.

Designing the generator as an independent component restricts interaction between the system and the generator in two ways: it provides a fixed division of labor between the system and the generator, and it prevents the work done within the generator from depending on or affecting the rest of the system. If the generator is designed independently, it will have a fixed task (or set of tasks) that it can perform and a fixed interface for request from the system. In particular, this assumes that the generator can accomplish whatever the system wants, since the system would have no way to provide the generator with help (or even to understand what the problem was). Conversely, this approach also assumes that any way of carrying out the system's request is just as good, since the generator has no way to ask for additional information to choose between them. Furthermore,

since the generator runs completely independently of the overall system, there can be no interactions between generation and other activities of the system. Language use will therefore be independent of the rest of the system; linguistic choice cannot depend on the system's current goals, beliefs, attitudes, or actions, since the generator has no way to find out what they are, and could not understand their representation even if it did have access to them.[1] Similarly, generation cannot be integrated with other actions of the system. For example, the system may be able to display pictures, but there is no way for the generator to coordinate its output with the display, since it can neither find out what is being shown nor bring about the display of a particular image.

All of these restrictions are consequences of the view of the generator as an independent component with a narrow interface, designed to be driven by a wide variety of systems. This approach walls off language from the rest of the system; all of the planning, reasoning, and decision-making are done with representations and mechanisms completely unconnected to the system's language-using components. The integrated approach proposed here, in contrast, removes these restrictions by closely integrating the generator with the rest of the system. The generator outlined here will share much of the representation and mechanisms used in planning, reasoning, and decision-making with the rest of the system. The generator's task is viewed not as expressing information in English (perhaps subject to some constraints), but rather as telling the system how it can use a particular kind of action to further its goals.

There are a number of reasons to adopt this view. The obvious reason is the practical one: it leads to generators that work better. It provides solutions to various limitations of current approaches to generation, as discussed below. Beyond the immediate practical advantages, however, there are theoretical advantages to this approach. Viewing generation as part of an intelligent system ties the development of a generator in with larger issues in both cognitive science and artificial intelligence.

One of the long-term goals of cognitive science is to build artificial intelligent systems, i.e. computer systems that can act intelligently in pursuit of whatever purpose we give them. An intelligent system will, of course, have to be able to communicate in natural language;[2] thus an intelligent system will need to use a natural language generator. Furthermore, the generator will have to accommodate

[1]The interface to the generator could be designed to provide this sort of information, but this doesn't necessarily help. The system has no way of knowing what may potentially be relevant, so it would have to essentially include everything it knows in the request. Not only would this be enormously unwieldy, but there's no way to guarantee that it would even be expressible in the interface formalism, since the formalism would be designed independently of the system. Even if this could all be made practical, the generator can't rely on the information, since the system's beliefs, etc. might change while the generator is working.

[2]In theory, of course, one could imagine an intelligent system that can only communicate in some formal language. But this seems unlikely to be useful, particularly since it would put the burden of learning to communicate on the user, defeating one of the main purposes of an intelligent system. Furthermore, an intelligent system would need to use the tremendous body of information that already exists in natural language form.

the needs of the overall system; the system must be able to control and monitor the generator so that it generates utterances that do what the system wants. The connections between the generator and the overall system must therefore be designed around how the system will use the generator. But designing a generator as an independent component leads to an interface based on the needs of the generator, i.e. derived from the design of the generator. There is no guarantee that such a generator will provide the services that an intelligent system will need. To ensure this, we must design the generator from the start as part of the overall system.

Indeed, designing the generator this way can help with the design of the overall system. The construction of an intelligent system is, of course, an immensely difficult problem. Building a generator, however, can provide a window onto the solution of the larger problem. Studying how language is and can be used by intelligent agents (both people and artificial systems) indicates much of what the generator must be able to do. But intelligent systems must be able to support whatever the generator can do *and* however the generator can be *used*. That is, the system must provide whatever general resources the generator needs (e.g. world knowledge, models of the hearer's beliefs, situation models) and must be able to make whatever decisions and obtain whatever specific information is needed to use the generator. Thus the generator design can give some indication of the knowledge representations and architecture of the overall system. In this way, knowledge about language can be used to help study broader issues in cognitive science.[3] In fact, there is a kind of design feedback here, since the intelligent system design deriving from (among other things) the generator design will undoubtedly suggest revisions to the generator design, perhaps cycling through numerous revisions. However this cycle can only get started if the generator is designed as part of an intelligent system; a generator designed as an independent component will neither suggest constraints on an intelligent system nor be able to incorporate revisions based on the design of such a system.

Even if practical issues are the main concern, the most important applications for a generator will require that it be part of an intelligent system. It is often useful to have natural language output, of course, but that doesn't necessarily require a generator. Compilers, for example, have been managing for thirty years with canned text; unclear error messages are usually because the compiler can't figure out what's wrong, not because of inadequate linguistic abilities. Simple template-based approaches have been used successfully in expert systems and data bases (e.g. [Miller 83,Scott 84]). In these systems, there is a fixed domain model encoded in the expert system or data base that can be used to drive the generation, and the kind of information the system needs to express can be determined in advance. Thus appropriate templates can be stored in the domain model; all the generator really has to do is smooth up the result of assembling the templates to make sure it's grammatical. This approach will only fail when the system is not limited to using a fixed model of a particular domain, i.e. when the information

[3]This is not the only way that studying language contributes to cognitive science, of course.

the system uses or the way it models it can vary in unpredictable ways. But this can only happen with an intelligent system, for only an intelligent system can deal with this unpredictability. Thus it is intelligent systems that really need full-fledged generators. Of course, it is possible to build a sophisticated generator for an unintelligent system, but such a system can't really take advantage of the increased flexibility the generator provides. In fact, attempts to improve explanations in expert systems have often required redesign of the system to make it more intelligent [Swartout 83,Clancey 81]. Thus generators that go beyond simple template-filling approaches will really only be useful as part of intelligent systems, and should therefore be designed that way from the start.

The linguistic knowledge that guides the generator design must include at least knowledge of what people do with language, i.e. of linguistic behavior. This is the whole point of natural language processing; to get computers to communicate in a way that is natural to people. It is also possible, though, to design the generator to mimic the organization of human language abilities (to the extent that they are understood). The question is whether the generator should merely do the same things that people do, or whether it should do them the same way. This correlates loosely with the distinction between cognitive modeling, which attempts to simulate and study human cognition via computers, and intelligent system design, which simply attempts to build systems that behave intelligently. Intelligent system design is generally concerned with how human cognition works only as an example of one possible design; if some particular human behavior is poorly understood or too complex, another design can be used. In the case of the generator design, however, the mechanisms of human linguistic behavior are relevant even if (as is the case here) cognitive modeling is not the goal. The problem is that the generator must be designed not just to mimic certain behavior, but to mimic that behavior as part of an overall system whose design is not fully defined. Human language production is an example not just of language production but of language production by an intelligent system. Thus to the extent that the generator is organized the same way as human linguistic apparatus, it will meet the goal of being (potentially) an integrated part of an intelligent system. The mechanisms of human language use are poorly understood, however, so the generator can hardly be designed to copy them precisely. Evidence from human behavior will simply be treated as strongly suggestive where it is available and its implications seem both clear and reasonable.[4]

For all of these reasons, the generator design outlined here will be developed to be usable as a fully integrated part of an intelligent system. The details of this design are not always explicitly motivated by this concern. The generator's role in an overall system, however, is at least implicitly part of all the design decisions. It determines what sort of issues and evidence are relevant to the decisions. Indeed, it helps determine what design decisions must be made. Thus even when not explicitly invoked, this conception of the generator as part of an overall system is a major design factor.

[4]The generator will not, for example, attempt to generate phonetic errors or misspellings.

1.2 Design Criteria

It's easy enough to say that generators should be designed as part of intelligent systems, of course, but it's not so easy to say what that means. After all, it's not as if we could simply open up a copy of a manual for intelligent system design and check the chapter on the generator interface. The organization and functioning of an intelligent system is a highly speculative subject at best. Nevertheless, we can get a general idea of some characteristics that such a system must have by considering what it will have to do. An intelligent system must be able to decide on a course of action that will best further its goals (even if, as is likely, those goals are constrained to something like "do whatever helps the user"), even when the decision can't be (or hasn't been) anticipated by the system's designers. This means it must be able to understand the effects of its actions in a general way, so that it can analyze how to deal with new situations. Thus the system must view itself as an agent, i.e. it must understand how it performs actions in the world (even if those actions are limited to communicating via a computer terminal or speech system). The system must also be able to understand a wide range of domains and how they interact, since these interactions may inform the effects of the system's actions.

These considerations have specific consequences for the design of the generator:

1. The semantics and pragmatics underlying the generator's linguistic knowledge should be the same as for the system as a whole. In other words, the generator shouldn't have separate conceptual structures or world knowledge. If the generator is to be a resource for the overall system, then the system must be able to understand what the generator is doing. This requires that they have the same semantics, for otherwise the system could not be sure it really understood the concepts the generator was expressing. Indeed, it's hard to see what advantage there could possibly be to having separate semantics or pragmatics for the generator. This would mean either that the generator would disagree with the system over the meaning of the utterances it produces, or that one part of the system would not have access to knowledge about the world that the other had.[5]

   Admittedly, some of this information is particularly connected to language. The knowledge that profanities have to have certain euphemisms substituted for them in polite company, for example, seems a particularly linguistic knowledge. The Gricean principles of cooperative communication [Grice 75] also seem closely connected to language use. But even this sort of knowledge is dependent on more general knowledge. The generator's understanding of linguistic profanity and euphemism must be grounded in

[5]Of course, if the goal is to build a generator as a portable "back end" for a wide range of systems, then it makes sense to give it its own semantics and pragmatics, because the systems it must work with may have very limited semantics and pragmatics of their own. But I am here concerned with general intelligent systems, which will have broad semantics and pragmatics.

general notions of politeness, rudeness, and offense. Similarly, Grice's Maxim of Quantity, which states that a (cooperative) comment should include as much information as necessary, and no more; this presupposes an ability to determine what information is necessary in particular situations. Indeed, much of what Grice says could apply to any form of communication. For example a traffic light displaying a green right arrow is interpreted, in line with the Maxim of Quantity, as meaning that turning right is allowed, but not going straight or turning left. The Maxim of Relation ("Be Relevant" [Grice 75, p. 46]) can only be interpreted with an understanding of the whole range of reasoning available to intelligent agents.[6]

Nevertheless, there is a purely linguistic aspect to much of this knowledge. Which words are profanities, and which are euphemistic alternatives, seems to be an arbitrary fact of the language. Some of the implicatures that arise from Grice's maxims (the "conventional" implicatures) are tied to particular words; the difference between "and" and "but", for example. So there will be some knowledge that is local to the generator (and the interpreter). It's only the purely linguistic knowledge, though, that is local. Whatever general knowledge or reasoning the generator needs to use must be the same as that used by the overall system.

2. Generation must be viewed as a kind of action, not just as a process of encoding information into natural language. The system invokes the generator because it's trying to accomplish something, not just because it had an urge to emit some information. The system's goal may indeed be simply to communicate certain information, but this is generally done in order to accomplish some larger goal. If this were not the case, the system would merely be "talking to hear its own voice".

   The notion of generation as action is widely endorsed in principle. Indeed, speech act theory is devoted to studying how language is used to perform acts beyond simply transmitting information. But the focus has been on what sorts of effects different utterances can have, not on how speech production can be organized to use these effects. Furthermore, research in generation has made only limited use of this view. The major focus has been on inferring the user's plan from his or her query in order to figure out what to say, which is really a matter of viewing the user's generation as action, not the system's. "Text planning" is usually concerned with organizing information into a coherent text, not with using speech acts to achieve goals. Even when the generation as action view has been taken seriously, as in KAMP [Appelt 85] and (to some extent) UCEgo [Chin 88], the concern has been on deciding what information to include, not on the actual generation. Once the information has been assembled, the generator simply turns it into English, with no concern about how the choices it makes may

[6]Grice, of course, is only concerned with communication between people; but the ideas apply equally well to artificial systems.

affect the plan the system is carrying out.

In an intelligent system, generation must be viewed as action at all levels of the system. The fact that it involves encoding information into language is actually not very important, in the same way that it's not very important that moving your hand involves sending electrical signals along your nerves and contracting various muscles. The important thing is how speech acts can achieve certain effects (one of which, of course, is the transmission of information) that the system might find useful. This means that speech is an action available to the system, but driven by the overall system, for which it is simply one option. Furthermore, speech must be a planned action down to the lowest levels, with goal-based choices at all levels, since all of these choices can alter the effects of an utterance.

3. The system's linguistic knowledge must not be tied to a specific domain or task. The system's knowledge of the meaning of a particular word or a particular syntactic construction cannot assume that it is being used in any particular domain or for any particular task. Of course, many terms have meanings that are associated with particular domains. For example "magnetic flux density" is generally only discussed as part of physics. Furthermore, there may be domain-specific indexing of expressions, so that they can be produced more readily when the system is working in a particular domain. But the system may need to discuss topics that cross domain boundaries, or that lie in domains it has never encountered before. It may not even know what domain, if any, the discussion is within.

   In addition, an intelligent agent must be able to cope with extensions of terms from one domain to another. A system that knows it can call 100°F a "hot" day, and a 500°F degree oven a "hot" oven, and that has appropriate domain knowledge about stars, should know it can call a 20,000°C star "hot" but not a 1000°C one. People can talk about things they've never experienced before, so domain-specificity of language can at best be a matter of efficiency or fluency.

4. Generation (like all activities of an intelligent agent) is a real-time activity carried out in a dynamic environment. The system must be aware of other things happening in the world that may affect what it wants to say and of the consequences of how long it's taking to say it. At its crudest, this means that the system should stop talking if the user leaves; similarly, the system shouldn't spend five minutes coming up with an eloquent way to tell the user his train leaves in thirty seconds. More generally, an intelligent agent can't assume that the things it is talking about are static; it must be prepared to revise what it's in the middle of saying if what it's talking about changes. The train that leaves in thirty seconds may suddenly be delayed for some reason; the system shouldn't continue on blithely advising the user to rush to the platform. Furthermore, the amount of time the system takes to generate utterances can itself have consequences; "don't do that" uttered rapidly is a

warning of danger, whereas uttered slowly it's just advice. Thus the system must be able to adjust to the consequences of its speed. At the very least, it must have a sense of how long it can go without saying anything without undesirable consequences.[7]

Generating in a dynamic environment thus requires being sensitive to how the environment can change and to how pressure to generate *something* builds up with time. The changing environment means that the generator must be able to patch plans as they are running, and the time pressure means that it must be able to start uttering before it has finished planning the utterance. Thus the generator must be dynamic in two ways: it must be able to alter the goals it is planning for in the middle of planning, and it must be able to produce possible (although not necessarily optimal) utterances before it is done planning the entire utterance. It can't simply start from scratch when changes in the environment cause revisions to its goals, because it may never finish constructing an utterance before the environment changes. Similarly, the utterance cannot emerge all at once, fully formed, because it could take arbitrarily long to form it, and the generator may be operating under time pressure.

Combining these criteria yields a novel characterization of the generator. The generator cannot simply apply a set of concept-to-language mappings, perhaps augmented by some structuring algorithms and by special-purpose domain and/or user models. Instead, the generator must solve the problem of how to use language, in combination with the system's other abilities, to achieve goals, drawing on a body of general knowledge that is shared with the rest of the system and that spans multiple domains.[8] Instead of being an "encoder" that takes some input, works to produce a corresponding linguistic output, and then stops; the generator must be able to run constantly as both its input and the environment in which it runs change, and it must be able to produce output without having completed all of its work. This is how a generator must work if it is to be fully integrated into an intelligent system.

Existing generators, of course, are (explicitly or implicitly) designed as independent components, and thus do not fit in with this characterization. In fact, existing generators by and large do not meet *any* of the criteria outlined here:

1. Generators usually don't have any semantics or pragmatics, just a lexicon and a grammar. Their main concern is the mapping from concepts to language and the well-formedness of the utterances they produce. They rarely do any reasoning about what to say in order to achieve particular goals, so they don't really need any semantics or pragmatics. Even generators that

[7]This depends on various aspects of the social and discourse situation and the task the system is performing; the PAULINE generator [Hovy 87] investigates some of these factors.

[8]Building such a knowledge base is, of course, a very difficult problem. But it will be a necessary step in building an intelligent system anyway, so allowing the generator to use it doesn't make the task any harder.

do reason about what to say generally use a private knowledge base to support that reasoning.[9] TEXT, for example, selects and organizes information using a special-purpose domain model designed specifically for it, not the actual database it is supposedly talking about. This is not surprising, since very few (if any) systems have an adequate knowledge base for generation. Even when generators do look directly at the driving system's knowledge base, it is usually just to look up "translations" of concepts; the knowledge base is not actually used in any of the generator's reasoning about what to say or how to say it [Sondheimer 86,McDonald 79].

2. As already discussed, the only real attempt to take the notion of generation as action seriously is Appelt's work on KAMP. And even KAMP is mainly concerned with selecting information for inclusion. In fact, most of its effort is devoted to making sure that the utterance doesn't refer to concepts the hearer doesn't understand. It has no general notion of the effects of utterances; thus KAMP could not handle a goal of "make the user happy" or "reassure the user", and it must assume that the user will always attempt to carry out any requests that KAMP makes. The UCEgo system also plans what to say based its own goals; however it also doesn't really have any model of the effects of speech acts. In fact, UCEgo leaves much of the work of generation, including some of the decisions about what information to express, to an ad-hoc generator built specifically for this purpose.

3. Most work on generation has been limited to a specific domain. Systems that have been applied to multiple domains have usually required completely redoing the linguistic knowledge to port the system. In fact, porting MUMBLE to TEXT [Rubinoff 86] required changing not just the lexicon but almost all of the grammar! Of course, there are generators that can work in various domains, e.g. Pennman/NIGEL [Mann 83,Sondheimer 86]. But the information for one domain will conflict with or disrupt the information for other domains, so the generator can't really be used as part of a general-purpose system.

4. Virtually all work on generation assumes implicitly that the generator can simply take as long as it needs (although there has been much concern about speeding up the generator). One exception is the incremental generation work of DeSmedt and Kempen [Smedt 87]. Their work, however, is more concerned with modeling human behavior than building a useful generator, and they are concerned only with the design of the tactical level of the generator (their "lexico-syntactic" module). They do not investigate how or when incremental generation might be useful; their work implicitly assumes that each component in the generator sends off its output as soon as it is ready. Thus the generator isn't really responding to a dynamic environment;

[9] Some recent work on text planning involving Rhetorical Structure Theory does do some reasoning that uses domain knowledge [Hovy 88b]. The knowledge is encoded in a very domain-dependent fashion, though, and is only used to support high-level structuring of the text.

it's just that the low-level components can get ahead of the high-level ones and produce output before they have all the relevant input. PAULINE [Hovy 88a] can determine and respond to time pressure, but only in an abstract sense. It can't respond to the actual time passing, only to the simulated time it is reasoning about.

1.3 Meeting the Criteria

The rest of this proposal outlines a design that aims to meet these criteria. While it won't fully satisfy all of them, it is a step towards a fully intelligent generator.

Chapter 2 discusses the architecture of the generator. Here the primary criteria are the need to view generation as action and as a dynamic, real-time activity. The former criterion will prove to require a mechanism for feedback from the "realization" component to the "planning" component in order for low-level decisions to be sensitive to the goals driving the generator. The latter criterion will necessitate an additional component (the "utterer") that can respond to time pressure.

Chapter 3 discusses the representations used by the generator. Here the primary criteria are the need for the world knowledge used by the generator to be shared with the overall system and to be independent of any particular domain or task. This amounts to requiring the representation to be suitable for the full range of reasoning, planning, and learning that an intelligent agent is capable of. Satisfying this requirement would be far beyond the scope of the current work, so the focus will be narrowed to building a representation that allows the generator to work within or across a variety of domains. The crucial feature of the representation will be a notion of *perspective* that allows the generator to focus on some information and ignore other information and to rearrange its knowledge when it shifts domains.

Chapter 4 describes the additional work that will be done to refine and implement the proposed generator and lays out some criteria for evaluating the work.

Chapter 2

Architecture

Designing the generator as part of an intelligent agent will obviously have consequences for the interface between the generator and the rest of the system. In particular, since the generator's task is to use language to further the aims of the overall system, the generator's input must consist of goals the generator must satisfy, not just information to be structured and encoded in language, and the generator must be able to let the system know how well it has accomplished those goals. It might seem, though, that the internal structure of the generator needn't be affected by this view, since it isn't visible to the overall system. Any architecture that can support the necessary behavior will do.

Strictly speaking this is true; the generator's role within the overall system doesn't directly determine anything about its internal structure. In practical terms, though, the requirements this role places on the generator do place significant constraints on its architecture. The architecture of the generator (or indeed of any computational system) determines what kinds of decisions it will (or can) make, which decisions will depend on which other decisions, and what information it can use to make those decisions. The architecture must reflect the kinds of decisions the overall intelligent system needs it to make and the kinds of information the system considers relevant to those decisions. Thus the design criteria laid out in Section 1.2 imply specific characteristics for the generator's architecture:

- Requiring the generator to use the same semantics and pragmatics as the whole system is of course already an architectural constraint. Since the generator proposed here won't actually be embedded in a complete system, this constraint isn't directly relevant. It does imply, though, that within the generator, the different components must share a common body of world knowledge. Also, the input to the generator must be expressed using the same knowledge representation that the generator uses internally.

  This doesn't mean that the generator can't have any specialized knowledge or representation schemes. Information which is specifically linguistic, such as the syntactic knowledge of the system, can be specific to the generator or even to particular components of the generator.[1] This kind of information,

[1] Presumably, at least some of this information should be shared with the language interpreter.

though, is by its very nature not directly accessible to the rest of the system. What isn't permissible is a specialized representation of (some or all of) the conceptual and factual knowledge available to the entire system.

- If generation is a kind of action directed towards one or more goals, then decisions at all levels must be made in light of how they affect those goals. It's not enough to just pick out some relevant information and find any coherent, grammatical, and comprehensible way to express it. The generator's job isn't just to express information, or even to find the appropriate information and express it. The generator's job is to use language to achieve the system's goals. Therefore decisions about any aspect of an utterance must potentially be sensitive to how well the result furthers those goals. A factor such as grammaticality or coherence can never be the sole constraint guiding the work of the generator.

- Since generation is a real-time activity, the generator must be capable of working incrementally. That is, it must be able to start producing output before it has constructed a complete utterance or set of utterances that achieve its input goal. The generator can't always take as long as it needs to construct an utterance; it always operates under some amount of time pressure. While the generator is working, the world can change in ways that invalidate its work, or perhaps even alter its goals. Furthermore, the amount of time it takes the generator to start uttering can itself affect the system's interactions with the user. So the generator must be able to produce parts of unfinished utterances when further delay would cause problems.

As we shall see, (most of) the architectures used in work on generation do not fit this description. In fact, the architectural limitations tend to cause problems even for generators intended simply as tools for encoding information into language, because language inevitably reflects the way it is used by people. Some attempts have been made to devise architectures that can overcome these problems, but none of them really satisfy the requirements of a generator for an intelligent system.

2.1 The Standard Model of Generation

Most generators use an architecture roughly like the one in Figure 2.1. An information component collects all the appropriate information to be expressed, usually in response to a request by the user, using some simple criterion of relevance to the request. This information is then passed on to a strategic component, which arranges it into a coherent ordered list of messages using some model of discourse structure. Finally, the ordered messages are sent off to a tactical component that turns each message into an appropriate sentence in English (or some other natural language).

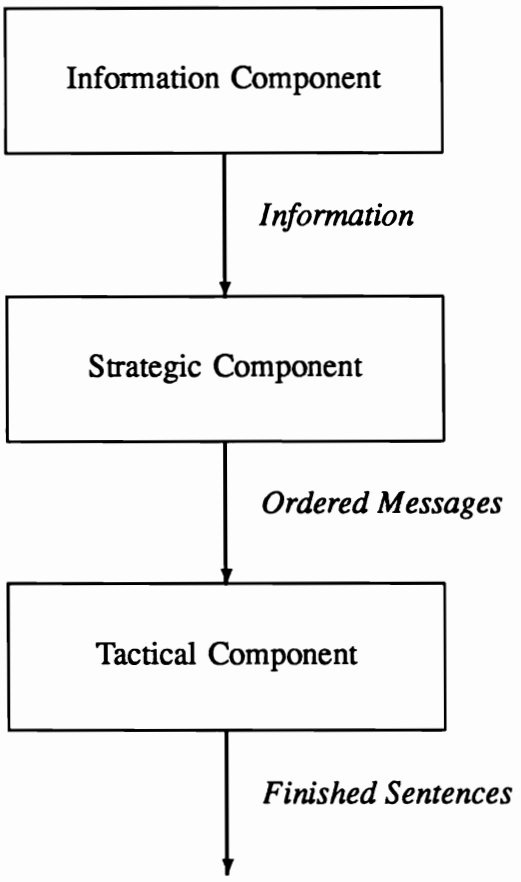


Figure 2.1: Architecture of Most Generators

There are some common variations; the strategic component is often combined with one of the other components, or the information component is simply left out of the generator, its work done by some other system before the generator is invoked. The components are also sometimes given different names: the "strategic" and "tactical" components were called that in some older work (e.g. [McKeown 85,Thompson 77,Danlos 87][2], but are also called "planning" and "realization" (e.g. [McDonald 83,Hovy 88a]) or simply the "what to say" vs. "how to say it" (e.g. [Reithinger 90,Danlos 87]). The precise allocation of work to the various components can also vary; the strategic/planning component, for example, can produce "ordered messages" that are very close to language, leaving the tactical/realization component with little to do, or "messages" that are simply pieces of the information in an appropriate order, forcing the tactical/realization component to do a lot more work. Some version of the standard model, though, continues to underly much of the work in generation. For example, DIOGENES [Nirenburg 88], EPICURE [Dale 89], SPOKESMAN [Meteer 89], Sibun's work on local organization of text [Sibun 90] and COMET [?] all follow the standard model. McDonald has even argued for extending the model to a large number of components [McDonald 88].

No version of the standard model, though, has the architectural characteristics needed by a generator that is part of an intelligent system. This kind of generator must be incremental, use a single shared knowledge base, and be able to consider how decisions at all levels affect the achievement of its goals. The standard model does allow for incremental generation, but only in a limited sense: the low-level component can put out parts of utterances as it finishes them. There is no way, though, to make this incrementality sensitive to the current time pressure, so that the generator can spend more or less time constructing utterances when it has more or less time available. Similarly, although there is no a priori reason the components can't share a body of world knowledge, in practice the complete independence of the components leads to separate knowledge bases, even when the information they contain overlaps. Furthermore, since the generator is independent of whatever system it is generating for, the generator's knowledge base(s) end up being separate from the driving system's. More fundamentally, the standard model doesn't allow the generator's goals to be considered in making decisions at lower levels, since only the highest-level component can see those goals. The goals can only influence lower-level decisions in ways that the highest-level component can anticipate. Since the high-level component can't know what the low-level component(s) will do, this influence is very limited.

The basic flaw of the standard model arises from its basic characteristic: information flows in one direction, from the high-level component to the low-level one. This means that each component must operate independently, without access to the decisions and decision criteria of the other components. The low-level component(s), of course, can see the output of the higher-level component(s), but

[2]Although Danlos uses "syntactic" rather than "tactical", and her use of "strategic" is following McKeown; see the note on [Danlos 87, p. 122].

that can include only the information the higher-level component(s) can anticipate a need for. Even in a generator that is not intended to be part of an intelligent system, this restriction is problematic; this kind of architecture will work only under two assumptions: the output of the high-level component is always fully specified for the low-level component's needs, and the low-level component can successfully process anything the higher-level component sends it. That is, the low-level component is never either unable to decide between options on the basis of its input or unable to find any acceptable choices; it can always find a unique best choice.[3] As we shall see, this is not the case with generation; the "tactical" component cannot always produce a single best way of expressing a particular piece of information. Decisions at every level must be sensitive to the high-level goals driving the generator, so the goals cannot be encapsulated in a high-level component; there must be a way to detect and handle decisions that seem appropriate by low-level criteria but conflict with goals. The lack of this sensitivity to interactions between low-level and high-level factors is the real problem; even modifications to the architecture that eliminate some of the practical difficulties don't eliminate this fundamental limitation.

2.1.1 Linguistic Limitations of the Standard Model

(1) a. John killed him with a gun.
    b. John shot him dead.
(2) a. John infected him with a virus.
    b. **John virused him sick.(?)
(3) a. **John homed him with an order.(?)
    b. John ordered him home.

The standard model of generation assumes that the low-level component's decisions never matter to the high-level component. This is not the case, though, as can be seen from the alternation in (1). The sentences (1)a and (1)b express essentially the same information, so if the generator is attempting to express this information, it must choose between them at some point. In the standard model, though, there's no point at which the decision can be made.[4] It can't

[3]One way to relax these assumptions would be to leave the output of the high-level component sufficiently underspecified that the low-level component always has at least one option and to pick one option at random when there is no other way to choose. This approach assumes that these "random" low-level decisions don't matter, which is not the case for linguistic decisions. Some generators do in fact deal with multiple valid options by choosing at random (or taking the first in an arbitrary order). They can do this without causing problems because the criteria that should affect the choice are beyond the sophistication of the generator; the necessary information doesn't exist anywhere in the generator.

[4]This depends, of course, on precisely where the division between high-level and low-level is made, and on whether there are two or more than two components. The discussion here assumes only that the component concerned with ensuring that the generated utterance achieves the generator's goals is different than the one that knows the specific details of the language being generated. If this is not true, then it's hard to see how there is any meaningful division into components at all, since there would be one component handling the entire range of the

be in the high-level component, because the availability of the choice depends on the particular linguistic resources available in English. This can be seen by comparison with (2) and (3), in which only one alternative is available. In fact, a different alternative is available in each case. Since the high-level component doesn't know which alternative(s) is/are available, it can't choose between them; the low-level component must make the choice.

On the other hand, the decision has to be made by the high-level component, since it can depend on and affect the goals the generator is trying to achieve. The choice between (1)a and (1)b should depend (in part) on what the generator is primarily trying to talk about. (1)a is more appropriate if the generator is going to continue talking about the gun, whereas (1)b is more appropriate if the main concern is the ramifications of the victim's death. Since the high-level component is the one that deals with this information, it must choose between the alternatives. Also, the choice between (1)a and (1)b determines what information can be easily omitted; cutting off the end of the sentence leaves out mention of the use of a gun in (1)a and the death of the victim in (1)b. Since the high-level component knows the consequences of omitting information, it must make the choice of which alternative to use and whether to abbreviate it. It might seem that the high-level component could simply indicate exactly what information it wants included in the utterance. But why should the generator always assume a strategy of saying as little as possible? Furthermore, decisions about what information to include may interact with other decisions. For example, the generator may want to emphasize the victim's death but not care about the means of death; it might then choose (1)b for the emphasis even though (1)a would let it skip mention of the gun. This kind of decision-making can only be done by the high-level component.

The decisions in (1), (2), and (3) involve lexical choice and dividing up information between different element of the sentence. So it might seem that this is where the problem really lies, that there could still be some way to organize the components of the standard model generator to deal with these decisions. The same problem occurs, though, in (4), (5), (6), where the choice is purely syntactic; the same lexical items are used in each alternative:

(4) a. I shut the box with a nail.
    b. I nailed the box shut.
(5) a. I covered the box with a tarpaulin.
    b. **I tarpaulined the box covered.(?)
(6) a. **I evened the edge with a trim(?)
    b. I trimmed the edge even.

As before, we can see from (5) and (6) that the available alternative(s) depend on the particular resources of English. Hence the choice must be made by the low-level component. Yet the choice also depends on and affects the concerns of the high-level component. The first alternative (when available) lets the generator leave out the instrument or means of the action; the second alternative allows

generator's tasks.

omission of the result.[5] Thus the dilemma is repeated here: the decision between (4)a and (4)b can't be assigned to any single component. Yet the standard model of generation provides no way for the task of making that decision to be shared between the two components.

This problem recurs when we try to see where to handle lexical choice. It seems that lexical choice has to be handled by the high-level component, since it depends very much on what the generator is trying to accomplish. For example, the choice of describing someone as either "firm", "obstinate", or "stubborn" should depend on what else the generator wants to say about him, as should the choice between "meek" and "wimpy". Or the generator might describe how justice was served by an "execution" rather than how the prisoner was "murdered" by the state. Similarly, the generator might deride the comments of a "dreamer", but praise the insights of a "visionary". These kinds of lexical choices can only be made by the component that handles the generator's goals.

On the other hand, there are a number of reasons why lexical choice has to be handled by the low-level component:

1. Lexical choice is very dependent on the particular linguistic vocabulary of the language being generated. Thus French, for example, distinguishes "connaître" (for knowing people) and "savoir" (for knowing information), but English just uses "know" for both. Or, in English, there is a word "giant" meaning "large man", but no corresponding word meaning "large car".

2. There is no guarantee, in general, that there will be any lexical item to express a given concept. For example, there is no word in English for the concept of a car with a removable door. There's no inherent reason why there couldn't be; after all, there's a word for a car with a removable roof. This is just a particular fact about English.

3. Since lexical choice interacts with syntactic decisions, it cannot be done in advance of choosing syntactic structures. For example, a generator cannot decide to use "probable" instead of "likely" without knowing if the completed utterance could be the ungrammatical "he is probable to be early". Similarly, the verb "drink" can't be chosen without knowing whether the clause will have a direct object; "he drinks apple juice" and "he drinks" actually have quite different meanings.

So here too it is impossible to assign the decision to a single component; the decision must be made by both components. The standard model provides no way to do this.

The problem is not merely that allowing the high-level component to make the decision could leave the low-level component with no available choices. That problem can be handled by some simple modifications of the standard model (as

[5]There are probably also pragmatic differences between (4)a and (4)b that the generator should care about, but they are too subtle to easily identify, so I won't discuss the question here.

discussed in Section 2.1.4). The real problem is that both components must be aware of and approve the decision; this requires eliminating the one-way flow of information that is the central feature of the standard model.

2.1.2 Self-Corrections and the Standard Model

Examining the mechanisms of human language production can be helpful in the design of a generator even though the goal here is not specifically to replicate those mechanisms. In general, whatever we can learn about those mechanisms suggests possible ways to organize the generator. Furthermore, human production provides an instance of mechanisms that clearly do work. Thus in particular, if human language production is organized according to the standard model of generation, then that model must be a viable one. On the other hand, if observed human behavior isn't in accordance with the standard model, then it suggests ways that the model should be modified.

The inadequacy of the standard model for describing human language production can be seen in examples of self-corrections, i.e. cases where the speaker interrupts him/herself and restarts some portion of the utterance. Consider the following examples (taken from [Kroch 81]):

(7) "used to have a sss– a match stick, a wooden match stick"

(8) "but aaa, bands like aaa– aaa– aaa– errr– like groups – groups, not bands,– groups, you know what I mean like aaa."

(9) "oh, but it's aaa– it's aaa– is it a doctor or is it midwife."

In each of these utterances, the low-level component runs into a problem that can only be solved by the high-level component. In (7), the speaker decides to describe the object with just a noun, but then isn't able to find an adequate one. So he decides to add a modifier to the noun, but again is unable to find an adequate one and must add a second modifier. In (8), the speaker discovers two words, each of which comes close to the intended meaning, and has difficulty deciding which is the most suitable. It is the existence of this near synonym pair in English that allows this indecision; if only one of the words were available in the speaker's vocabulary, then there would be no decision to make. In (9), we see a case where the speaker didn't feel a need to determine the precise role of the person being mentioned until a gap in the available vocabulary forced it. Had there been a (common) English word available meaning "generic medical service person" or the like, then the speaker could have used it and never considered whether the person was a doctor or a midwife. In all three of these examples, the speaker has decided what to say and organized the information without any trouble. The difficulty only arises when the speaker tries to find particular linguistic elements to include in the utterance that there is a problem. Thus the low-level component has been given a request it can't fulfill without further help from the high-level component.

These examples all involve lexical choice. The same kind of alteration occurs with structural decisions:[6]

(10) "I mean it was generally – you generally shied away from him"
(11) "and I really enjoyed the full spectrum I – of my training there"
(12) "I used to – there was a black guy that I was really good friends with"

In (10), the speaker switches from an impersonal passive construction to an active construction using the generic "you", possibly because the first attempt was becoming awkward. In (11), the original version was replaced with a structure that focuses more attention on the training and less on the speaker. In (12), the clause structure is rearranged to allow the euphemism "really good friends" to replace the original statment the the speaker dated a black man (which is indicated by the surrounding context). The precise difference in effect between alternatives, (and hence the reason for choosing one over the other) depends on the details of the pragmatics of English and is not important here. The relevant point is that the speaker is choosing between different ways of expressing the same information.

Self-correction, of course, is a kind of breakdown in the generation process; the speaker "decides" that the utterance so far is not right and alters it. This raises two questions: how is the breakdown detected, and how is corrective action taken; that is, where in the generation process is it decided that the current utterance should be changed, and how is the change made? Broadly speaking, there are two possible answers to the first question. A separate "monitoring" process might examine the utterance as it's being said; when it detects something it doesn't like it could stop the generator and modify the input the generator is working on to avoid the problem. Alternatively, the generator itself might be organized in such a way that it can change its decisions even after it has (partially) produced output affected by those decisions. The examples here seem to indicate that both alternatives are true. In (7), (8), and (12), for example, the speaker seems to decide that the utterance is not acceptable as it is, either because it doesn't adequately describe the object or because it involves a statement the speaker is uncomfortable with (in this situation). These both seem like the result of monitoring, i.e. interpreting the utterance and evaluating its effect. In other examples, the decision to change the utterance seems to come from within the generation process. In (10) and (11), the generator is continuing an utterance that has already been partially expressed when it decides to use a different construction, because the current one is either awkward or has the wrong pragmatic consequences. In (9), the speaker is simply unable to find an appropriate lexical item to finish the NP he has begun; thus the generation process is interrupted to try to deal with the problem. Thus there are several examples of self-correction triggered by the generator itself, in addition to the

[6]This sort of example is harder to find and to interpret than lexical examples, because it's usually very difficult to tell how the rejected alternative would have continued. For lexical examples, it is usually simple to determine the rejected word, since part or all of it is actually spoken. Nevertheless, it is possible to find a few examples or structural alteration where it's clear what is going on.

examples which seem to be the result of independent monitoring of the utterance.[7] Indeed, self-correction seems to be triggerable by several of the components within the generator. In (10), the decision is a purely low-level one: the construction is becoming awkward. In (11), the problem is detected at the interface between the components; the original version encodes the right information, but the alternate version has pragmatics that better fit in with the overall discourse structure. It's difficult to tell exactly what's going on in (9), but there seem to be two breakdowns (note that the speaker self-corrects twice, although this is very common and thus may not be significant here): first the low-level component is unable to find an appropriate lexical item, and then the high-level component is unable to further specify the concept.

The really important issue, though, is how the correction gets done, i.e. which component(s) are involved, how does information flow between them, and which one controls the process. One possibility, of course, is that the entire utterance is abandoned and the speaker simply starts the generation process over again. This does seem to happen sometimes, but it's not very interesting. It certainly doesn't explain examples where only a small part of an utterance is corrected; and it also doesn't explain why the speaker doesn't simply generate the same utterance over again, complete with the same self-correction, ad infinitum (or until he gives up entirely). So presumably at least some cases of self-correction involve some part of the generation process reconsidering a previous decision and proposing an alternative. In the standard model, since each component works independently, only one component can be involved in this reconsideration process. Of course, the results of the reconsideration can be passed on to lower-level components, but they then simply respond to this new input; they can't affect the reconsideration itself. The examples here, though, show self-corrections which involve a combined response by both high-level and low-level components. These are not simply low-level mistakes (e.g. phonetic or grammatical errors) or high-level mistakes (e.g. mentioning the wrong object, or giving information in the wrong order). The examples here involve low-level decisions that cause problems for the high-level component; thus the high-level component must reconsider its decisions on the basis of what the low-level component's options are. This is only possible if the low-level component can pass back to the high-level component a description of what it's done and what the problem is. For example, in (7) the low-level component needs to tell the high-level component that it left out information so the high-level component can decide what else needs to be added. Similarly, in (8) the low-level component has to "pass up" the need for information to distinguish between "group" and "band". This kind of "backward" information flow from lower to higher components is precisely what the standard model of generation forbids.

[7]There are also many apparent self-corrections where the speaker abandons the entire utterance and goes on to say something entirely different. These examples are presumably independent of the entire generation process, just as, say, an analysis of human vision need not account for situations where people turn around and look at something entirely different from what they had previously been looking at.

2.1.3 Limitations in Existing Systems

The division of the generator into separate components is intended to simplify the overall design. Each component can work on a particular part of the problem; no component ever has to deal with the full complexity of generation. If the generator is inappropriately organized, though, this division will tend to break down, because the components will have to make decisions without all the necessary information. The standard model of generation provides no way to make decisions that involve both low-level and high-level issues. The effects of this limitation can be seen in existing systems, which have had to violate their intended organization when faced with such decisions.

The TEXT system, for example, the strategic component is supposed to make "all decisions about what to include in the text" [McKeown 85, p. 5]; the tactical component merely "uses a grammar to translate the message into English" [McKeown 85, p. 5]. Yet some decisions about what information to include are in fact encoded into the tactical component. The attribute value "**WATER**" is used in TEXT to indicate that some object travels in or under the water. When included in a message in a context where it is appropriate to translate it as an adjective, it is simply omitted. This is because the only available adjective is the somewhat awkward "water-going", and it turns out that the cases where "**WATER**" appears in an adjective context don't include any where it's important to say it. Thus this strategic decision (that "**WATER**" can be omitted) is encoded permanently into the tactical component; there is no way for the strategic component to control this decision.

In MUMBLE [McDonald 83], in contrast, the interactions have pushed low-level information into the text planner. For example, MUMBLE can take the already constructed phrase "Fluffy, Floyd's dog, buries bones" and modify it with the information that this was reported by Helga to produce "Helga reported that Fluffy, Floyd's dog, buries bones". But in order to do this, it has to mark the information about Helga with the "**new-main-clause**" feature. So the planner has to know what clauses are, know that the earlier information was turned into a clause, and know that making "Helga reports" a new main clause is a useful thing to do. Many of the low-level decisions are thus actually being made by the planner.

The limitations of the standard model have not only affected the design of the generator, but in at least one case they have even affected the the design of the formalism the generator uses. TEXT has two different forms of propositions that describe subclasses of an object type. One of them is used when there is only one set of subclasses; a second form, which includes less detail, is used when there is more than one independent set of subclasses "in order to avoid putting too much information into a single sentence."[McKeown 85, p. 233] Of course, this only makes sense under the assumption that every instance of this type of proposition will be expressed as a single sentence; there isn't even a way for the strategic component to sometimes defer the decision to the tactical component. Doing this properly would require the tactical component telling the strategic component how

it had realized the proposition so the strategic component could detect sentences with too much information. The standard model provides no way to do this, though.

2.1.4 Variations on the Standard Model

There have been some attempts to modify the standard model of generation in order to alleviate some of its limitations. These modified architectures have indeed overcome some of the symptoms of the problem, but none of them have addressed the heart of the problem: the inability of high-level components to monitor and react to the actions of low-level components.[8]

2.1.4.1 Interleaving of components: MUMBLE and TELEGRAM

One variation involves interleaving the work of the components. The high-level component passes partial results on to the low-level component; when the low-level component encounters an unspecified part of its input, it sends it back to the high-level component to be fleshed out. For example, MUMBLE allows interleaving of planning and realization [McDonald 83][9] and TELEGRAM allows interleaving of the planner and the grammar [Appelt 83].

This technique does allow the surrounding linguistic context to have some effect on the choice of what to say (and when to say it). It is limited, though, to situations where there are pieces obviously missing from the high-level component's output. It doesn't handle cases where the low-level component needs a better description of a piece of its input, or where it can't find any appropriate options, or where it comes up with an option that seems appropriate but causes other problems for the high-level component.

2.1.4.2 Backtracking on failure: KAMP and Kalipsos

A few generators deal with deadlock in the low-level component by simply backtracking and reconsidering previous decisions when confronted with a decision with no acceptable choice. KAMP, for example, has a hierarchical planner in which failure to find a low-level plan causes a backup to redo higher-level planning [Appelt 85]; in a generator built for the Kalipsos system, the syntactic component backtracks to the linguistic component when it can't find a grammatical way of expressing its input [Nogier 89].[10] This approach guarantees that if there is a way to accomplish the generator's aims, it will be found; the generator will never get stuck and give up.

On the other hand, this approach only helps in cases where the low-level component has no options. It doesn't help when the low-level component has

[8]The one exception is discussed in Section 2.1.4.5.

[9]It's not clear, though, whether this facility was ever actually used.

[10]In theory, the ATN-based strategic component in TEXT could backtrack if it got stuck, but in practice the use of a limited lookahead eliminated any need for backtracking.

multiple options it can't choose between, or when an option that seems fine to the low-level component is unacceptable for higher-level reasons. Even in the cases where it does help, it's only recourse is to back up the entire generator a bit and try a different option. Yet the low-level component might only need a slight change in its input to proceed. There is simply no way for the components to work together to correct the problem.

2.1.4.3 Enlarging scope of strategic component (Danlos)

Laurence Danlos has argued for an enlarged strategic component that makes decisions regarding order of information, syntactic structure, and lexical choice.[11] Her argument is based on the interdependence of all of these decisions; since none of them can be given priority over the other, they must all be decided together. It's certainly true that these decisions are all interdependent, but that doesn't mean that they must all be made at the same time. The interdependencies could also be handled by backtracking when a desired option is ruled out by a previous decision; the architecture proposed below provides another way to handle them.

Nevertheless, Danlos's approach is a legitimate way to cope with interdependencies between different kinds of decisions. Combining several different kinds of decisions, though, makes the decision process very complicated. In order to be able to handle this complicated process, Danlos has to lay out all the possible decisions in advance and indicate which combinations are acceptable. The net result is that the decisions are all made either by arbitrary domain-specific decision trees (lexical choice) or by checking a list of possible combinations of decisions and picking any acceptable one (ordering information and syntactic choice). Choosing words by an arbitrary and domain-specific procedure is not acceptable for a general-purpose generator; the system must know, for example, that the same word "high" is being used in "her temperature was very high" and "the plane was flying high". Listing and rating all possible combinations of decisions becomes increasingly difficult as the range and flexibility of the generator increases. If the generator were also to decide what information to include, it would become prohibitively difficult. Also, there is no way to choose between the acceptable options; any acceptable option is as good as any other, even though there might be differences that matter. So Danlos's approach limits the flexibility of the generator.

2.1.4.4 Interrogation by Low-Level Component: Penman and POPEL

Some generators allow the low-level component to ask the high-level component for further information. In Penman, for example, the grammar (NIGEL) can use its "choosers" to get additional information from the planner when it has to make decisions [Mann 83,Sondheimer 86]. In POPEL the "how to say it" component

[11]Note that she assumes explicitly that the generator is given a complete specification of the information to go in the message, so she is still using the standard model for part of her generator.

(POPEL-HOW) can ask the "what to say" component (POPEL-WHAT) for additional information if it finds its input underspecified [Reithinger 90]. It might seem that this kind of backward interrogation fundamentally repudiates the standard model. The high-level component is free to incorporate whatever high-level issues it wants into its response to the interrogation; it can even remember what it was asked and let that affect its subsequent decisions. The problem, though, is that the high-level component has no way to tell why it is being queried or how its response will be used. Since the query is triggered by low-level concerns, the high-level component cannot understand the relevance of the query and response without understanding the details of what the low-level component is doing. For example, suppose NIGEL queries whether some expression is a process in order to decide what tense to use in a sentence. The planner doesn't know anything about tense, so it can't understand why it's being queried. The planner also can't assume that the resulting sentence indicates that (the referent of) the expression is (or isn't) a process, since it has no way to tell what, if anything, NIGEL has done with the response. The interrogation remains entirely under the control of the low-level component, and the high-level component has no way to use the queries to adjust its own work.

2.1.4.5 Restrictive Planning: PAULINE

Hovy's PAULINE generator [Hovy 88a,Hovy 88c] is perhaps the closest in spirit to the current work. PAULINE is intended to capture how generation can be affected by pragmatic issues such as the knowledge, opinions, emotional state, and goals of the participants in a conversation and by their relationship and the physical and social setting of the conversation. These kinds of issues, of course, only make sense in terms of an intelligent system; an unintelligent database can't have opinions or an emotional state and can't have a relationship with a person. It's not surprising, then, that the demands of the task forced PAULINE to use an architecture that differs significantly from the standard model.

PAULINE does in fact allow information to flow back from low-level to high-level components by means of what Hovy calls "restrictive planning". In restrictive planning, the planner doesn't build up a detailed plan and send it off to an execution or realization component to be carried out. Instead, the low-level component presents several options to the planner, which simply chooses the one that is best for its purposes. This kind of combined operation allows the planner to be sensitive to low-level issues without requiring it to understand all the factors that might be relevant. Thus restrictive planning appears to solve the problems of the standard model.

This appearance is illusory, though. Restrictive planning is primarily intended to avoid spending a lot of effort on top-down planning in situations where it's not very useful. If the realizer only has a few options available for some message, it's simpler and faster to just let the planner check which of these it likes best instead of working out a plan in great detail. The final result, though, is the same as what prescriptive planning would have produced. PAULINE simply uses

top-down planning in some situations and bottom-up planning in other cases for efficiency. Either way, the final result has to meet the criteria of both the planner and the realizer; there is no way for either component to insist on something that the other doesn't accept.

The model proposed below, in contrast, provides for information to flow back and forth between the two components, allowing them to negotiate over what will be said. Each component can over-rule the other's rejection of an expression if the alternatives are inadequate from its point of view. The goal is not efficiency but rather ensuring that relevant criteria at all levels are applied. In contrast to PAULINE, the "planner" here will be able to reject all the options the "realizer" proposes if none of them are acceptable and insist on additional options to choose from; conversely the "realizer" will be able to report that there is no way to realize exactly what the planner wants and suggest alternative messages to realize. Neither component will have priority over the other; in fact, there will have to be a third component (the "utterer") that will resolve the conflicts between them (see Section 2.6). In restrictive planning, the low-level component can only help the planner narrow the focus of its work; in the architecture here, the low-level component can also help the planner broaden its focus.

A second difference between PAULINE and the current work lies in how the low-level components construct and describe the options for the planner.[12] PAULINE's realizer consists of a large number of small "specialists" that handle specific tasks, e.g. `SAY-EVENT-SENT` or `SAY-PRONOUN`. These specialists can be arbitrary procedures, so the realizer doesn't have any overall theoretical or linguistic basis. Of course, individual specialists may be designed according to various linguistic principles, but this is irrelevant to the nature of the realizer as a whole. Crucially, there is no overall common basis for the specialists; they are essentially ad hoc. The linguistic component here, in contrast, is grounded in a general knowledge representation designed specifically to meet the needs of an intelligent generator (see Chapter 3 for details). The linguistic component uses general techniques to derive options from this representation and specific linguistic knowledge. Thus, unlike PAULINE's realizer, it does have a general theoretical basis underlying its work. The linguistic component is independent of the specific domain and task the generator is talking about; it reflects a general model of how to use language to realize the planner's requests.

2.2 Revising the Standard Model

If the standard model isn't good enough, then what is? One possibility would be to just have one big component that simultaneously considers all aspects of the generation process. This would complicate the generator enormously, though. Furthermore, it seems counterintuitive to suggest that deciding whether to use a pronoun, say, is done in the same way and at the same time as deciding what to

[12]Of course, this isn't really an *architectural* difference, but it does reflect whether the generator is viewed as part of an overall intelligent agent or simply deals with issues related to intelligence.

talk about. There does seem to be a natural distinction between different levels of decisions in the generator, and it would be nice to take advantage of that distinction.

What is needed is a way to have separate components that deal with different levels of decision-making while allowing them to co-operate on decisions depending on more than one level. The natural way to provide this co-operation is to allow the components to talk to each other. The standard model only allows for one-way communication; to achieve true co-operation we need two-way communication, i.e . there must be feedback from the low-level component(s) to the high-level one(s). The high-level component can then add to or modify what it passed on to the low-level component on the basis of how the low-level component responded to its original input.

The model suggested here has the following characteristics:[13]

- Each component will only deal with a particular level of the generation process. All of its decisions will be at its own level, and it will only be able to consider information from other levels when that information has been explicitly provided by another component. This will rule out the kinds of compromises seen in Section 2.1.3.

- Each component will treat the output it sends to a lower-level component not simply as a message to be further refined and translated but rather as a specification of what it wants the lower-level component to do. The difference is that the lower-level component might not follow the specification precisely; the higher-level component must be able to respond to differences between what it wanted and what was actually done. This is necessary because the low-level component can't guarantee it will always be able to follow the high-level component's requests, as can be seen from the self-correction examples (e.g. (7) and (9)).

- Feedback from the low-level component(s) will inform the high-level component(s) when the low-level component either has no available options, has multiple options it can't decide between, or when it has made a choice with consequences at a higher level that weren't anticipated by the high-level component. This will allow the high-level component to deal with these problems by adding to or modifying its output to the low-level component. Decisions that involve multiple levels, such as the decision in TEXT of whether to include the word "water-going" (see Section 2.1.3) can be made in a principled manner, without violating the separation of levels of processing.

[13]I'm not assuming any specific division into components here. The specific architecture proposed below will, of course, have a particular set of components. But the characterization here is independent of how many components there are or precisely what each one does.

2.3 Outline of the Generator

The design of the generator must meet the criteria laid out at the beginning of this chapter. It must be incremental: the generator must be able to produce partial output before it has completely decided what to utter. It must share a single knowledge base with the rest of the system: the input goals and all the components of the generator must be grounded in the same world model. Most importantly, it must be deeply goal-sensitive: decisions at all levels must be sensitive to how they affect the generator's achievement of its goals. The standard model of generation makes this impossible, as detailed in Section 2.1, because the interface between the different components hides the low-level decisions from the high-level component. The generator proposed here will instead follow the revised model of Section 2.2; the interface between the components will allow for appropriate kinds of interaction between the them.

The crucial property of this interface is that the low-level component can hand back linguistic expressions with descriptions of what they do, rather than simply assuring the high-level component that whatever it has produced is absolutely (or adequately) correct. When the low-level component is unable to accomplish exactly what it is asked, or when it has no basis to choose between several acceptable expressions, it can simply describe the available options to the high-level component. The high-level component can then decide whether to make do with one of the available options (and which one), or to reformulate the request and try again. The low-level component can remain ignorant of the concerns that underly this decision. Similarly, the high-level component need not worry about whether its requests to the low-level component are achievable; if there is a problem, the low-level component will let it know. This interface allows the generator to maintain a clean separation of different levels of its work despite the need for interaction between the levels.

The interface, of course, is just part of an overall architecture, which is laid out in Figure 2.2. The communicative planner and the linguistic specialists are the "high-level" and "low-level" components of the system. The workspace is the interface between the planner and the specialists, where they put requests and results. The utterer is the part of the generator that actually ships off utterances to be spoken or written; the need for a separate component to do this will become clear. These components work together to build up and output utterances. The planner is the component that drives the rest of the system, since it knows when and why the system wants to produce an utterance. The various components run in parallel, however, because each one's task is largely independent of the others'; they are connected solely by means of working on the same entries in the workspace.

Each component of the generator handles a particular aspect of the generation task, involving different kinds of knowledge and different constraints on the generator's work:

Communicative Planner This is a special-purpose planner that accepts goals

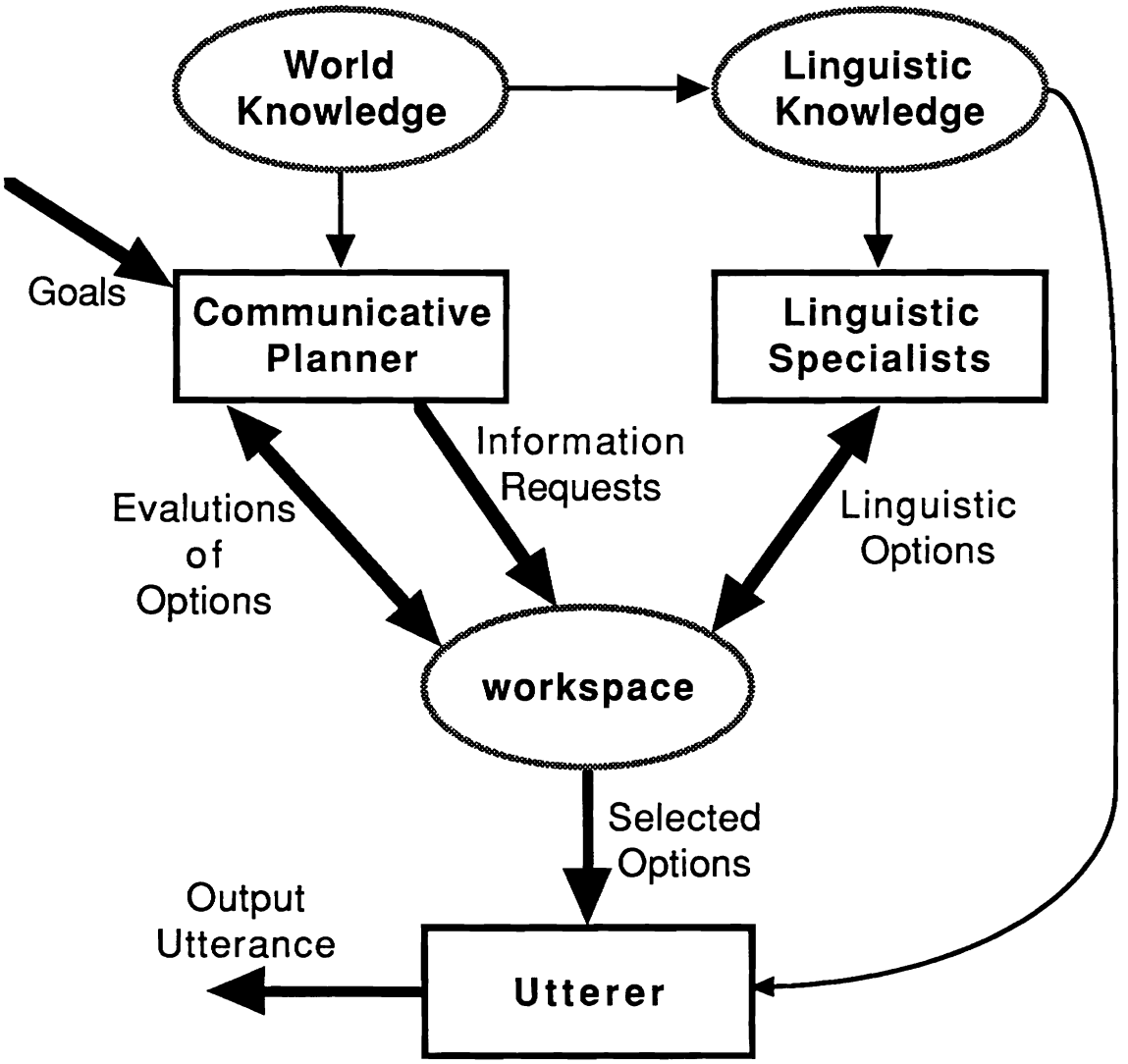


Figure 2.2: Architecture of the Generator

from the overall system and plans out ways to achieve them via communication. These can be "communicative" goals such as "transmit this information to the user" (or more precisely "get the user to believe this information"), but they can also be social interaction goals such as "establish a deferential (or collegial, or superior) attitude toward the user" or cooperative behavior goals such as "be straightforward" or "transmit as little information as possible". These goals can also be either "foreground" or "background" goals. Foreground goals are the ones that the system specifically wants to achieve; these are the ones the planner must plan for. Background goals are persistent goals that the system would like to accomplish if and when possible; the planner should prefer plans that achieve them, but not deliberately plan for them.

The communicative planner draws on a set of specialized plans that capture how communication acts can achieve various goals. The planner must be a dynamic planner in two senses. First, it must be able to plan in a changing environment.[14] In particular, the overall system in which the generator is embedded may decide to abandon some of its goals or to achieve them through some other means. In addition, when the planner starts building a plan, it cannot assume that it has a complete and accurate description of the effects of its "atomic" actions. The linguistic specialists cannot always produce an expression with the exact intended meaning. Thus the communicative planner must be able to revise its initial plan to accommodate any mismatches between its requests and the responses from the specialists. In other words, the planner has to cope with the limitations of the language being generated.

Linguistic Specialists This is the part of the system that produces the actual bits of language that make up the utterance. The linguistic specialists act like "consultants" to the communicative planner, responding to its requests with linguistic expressions that attempt to capture part or all of the information the planner wants to express. There are several linguistic specialists because there are several kinds of linguistic knowledge that come into play. For example, there are specialists that handle lexical choice, clause structure, and phrases of various types. In a system producing speech, there would also be a specialist for intonational structure. Other specialists could be added if appropriate, e.g. a specialist for rhymes in a system to generate poetry. Also, some of these specialists may prove to need splitting up into several specialists; the precise division of labor among the specialists is an open question. This division does not, however, correspond to different kinds of requests from the planner or different kinds of information in those requests.[15] Different specialists may suggest different ways of expressing

[14]Actually, as discussed in Section 2.4.1, the implementation I intend to build will not deal with this problem. The architecture, however, can easily accommodate a revised planner with this ability.

[15]Except, of course, when those differences are directly built into the structure of language.

the same information or similar ways of expressing different kinds of information. From the planner's point of view, there is no way of telling which specialist suggested which expression.

The specialists are primarily intelligent lookup components; they respond to the planner's requests by looking up linguistic expressions that capture the intended meaning. The results can be "lexical", i.e. words or phrases whose meaning corresponds to the planner's request; structural, i.e. specific syntactic structures that convey meaning;[16] or possibly intonational in a speech generator.[17] The linguistic specialists go beyond simple lookup in two ways. First, they actually produce a continual stream of options rather than simply returning a single option or list of options. This allows the utterer to respond to time pressure on the system (see Section 2.6), and it also prevents the specialists from spending a lot of time looking for alternatives when an adequate expression has already been found. Second, the specialists attach annotations to the expressions that they look up. Some of these, such as those describing stylistic information, may be permanently attached to the expressions, but others must be constructed by the specialists because they describe dynamically determined information such as dependencies between different expressions and mis-matches between the planner's request and the expression produced by the specialist.

Workspace This is where the other components of the system work together to build up the utterance. It contains two types of entries: information that the planner wants to express, and linguistic options suggested by the specialists to express them. The entries can be annotated to indicate further constraints on how the information should be expressed or additional effects of the options beyond what the planner requested; these annotations make it possible for the generator to cope with mismatches between the planner's requests and the resources of the language. The workspace in some ways resembles a blackboard [Nii 86b,Nii 86a]; it contains several possible utterances that are described at different levels of structure and are built up incrementally by various independent components. On the other hand, it doesn't have the strict hierarchical structure that usually characterizes blackboard systems; each request from the planner may correspond to several linguistic options, and the linguistic options may handle (parts of) more than one request. Also, blackboard systems usually have sequential scheduling of actions, whereas the planner, the utterer, and the specialists all run in parallel.

The workspace is also the only channel of communication between the var-

[16]Note that syntax can express both information about the topic being discussed and "discourse" information about how statements connect together. Of course, lexical choice can have this effect also.

[17]Even in a text generator, roughly comparable effects could be achieved using punctuation and typography, e.g. capital letters, boldface, etc.

ious components.[18] Direct communication between the components, while apparently simpler, would actually limit the generator. The planner and the specialists will often need to incrementally modify their initial entries to achieve an adequate match between the planner's request and the specialists' options; having them in a common workspace simplifies this process. More importantly, the planner and the specialists may split up the work in different ways; thus the specialists need to examine many requests simultaneously, and conversely the planner may need to examine several responses together.

The communicative planner places into the workspace descriptions of information that it wants to express. These can include information in the normal sense, i.e. descriptions of objects, events, actions, and so on; but they can also include desired pragmatic or intersocial effects, e.g. placing a particular object in focus, or indicating respect for the hearer.[19] These requests can be annotated with constraints on how the information should be expressed, for example requesting colloquial phrasing or emphasis on a particular aspect of an action. These annotations are not absolute constraints on the specialists, but they direct the specialists what to try first and affect the ratings of the options they produce.

Linguistic options are placed into the workspace by the linguistic specialists in response to the requests from the planner. These consist of pieces of syntactic structure (including the words); they can range from a single word or syntactic node to a fully formed sentence. These options can have several different kinds of annotations. They all have at least two annotations: an indication of which part(s) of which request(s) they express,[20] and a rating of how well they do so. In addition, each option may have annotations indicating parts of the requested meaning that they don't capture, additional implications beyond the requested meaning, and a description of the style and manner of the option, indicating for example if the option is formal or colloquial. These annotations are used by the planner to evaluate the options; they allow it to choose between the options on a principled basis without having any knowledge of the language they are expressed in.

Utterer This component makes the final selection from the options developed by the linguistic specialists, combines them and fixes up the syntax of the resulting utterance, and ships the result off to be written or spoken. Normally,

[18]This is not strictly true; the planner does actually set certain parameters that determine the utterer's preferences. But the workspace is the only channel for communicating *information* between the components of the generator.

[19]It's easier to see how the linguistic specialists could respond to this kind of intersocial request in a language like Japanese, where explicit honorific terms are common. Even in English, though, there are expressions such as "please" or "as you know" that seem to have primarily intersocial effects.

[20]Note that a particular option may express only part of the information in a request, and may express information in more than one request. This is one of the ways that the workspace differs from blackboards, at least as they are usually organized.

of course, the utterer follows the recommendations of the communicative planner; but it is still necessary to have it as a separate component. One reason for this is that while the annotations tell the planner the individual effects of each option, the planner has no way to anticipate the effects of combining various options into an utterance. For example, will combining two slightly awkward expressions reinforce their awkwardness to make an unacceptable utterance, or will the result also be just slight awkward? This sort of question involves specific linguistic knowledge that the planner has no access to; thus the utterer must make these decisions. The utterer must balance the planner's preferences against the need to minimize awkwardness and redundancy and maximize consistency of tone.

The utterer also captures the effects of time pressure on the generator. Since there is always a better way of saying something (or at least another way), the system could spend forever refining its output. The utterer ships off some expressions whenever the balance of time pressure and current ranking of candidates indicates that the disadvantage of waiting any longer outweighs the imperfections of the best available option so far. The amount of time pressure, and the importance of speaking carefully, can of course be varied. Thus the generator can be set to speak quickly and carelessly or slowly and carefully depending on the situation in which it is being used.

2.4 The Communicative Planner

The communicative planner is the part of the generator responsible for ensuring that the generated utterance achieves the intended goals. It draws on knowledge of the effects of communication to plan out how to achieve various kinds of goals by expressing various kinds of information. The planner depends on the linguistic specialists to tell it how to express the information in language, and it must adjust its plan based on how closely the specialists succeed in doing so. This is not an entirely negative situation, however; the recommendations of the specialists may sometimes suggest revisions to the plan that actually improve it.

The communicative planner is similar to the "strategic component" or "text planner" of many generators, but it actually has less responsibility than these components usually have. The planner is not the final authority on what will be produced. The goals driving the generator are only one of the constraints on the generator's output. Of course, normally they will be the most important constraints, but in some circumstances the goals can be overruled by the need to maintain fluency or consistency of tone or the pressure to generate quickly.[21] Thus the planner must strongly guide but not completely determine the construction of the utterance.

On the other hand, the communicative planner in some ways has wider responsibilities than most "strategic components". Most generators are designed to

[21] These issues are handled primarily by the utterer; see Section 2.6.

let the rest of the system decide what needs to be said; the strategic component simply organizes the information presented to it, or perhaps selects a subset of the information that it can assemble into a coherent text. The communicative planner, however, can in principle include anything in the system's general body of knowledge and belief that might support the goals it is given. Even if the planner has only been charged with transmitting some specific information, it may decide to include additional information that will make the resulting text clearer. Thus the division between the planner and the system driving the generator is less sharp than in most generators.

The planner is invoked when the overall system decides to accomplish some goals by communication.[22] There are several types of goals that can be accomplished via language. The obvious one, of course, is to communicate information. But the generator can also respond to social interaction goals such as "be deferential" or "establish a dominant (or subservient) status with respect to the user" either by varying the tone of an utterance expressing some other information or by generating an utterance specifically for that purpose (e.g. "Excuse me" or "you insolent slob!"). In addition, the generator can deal with goals concerning how co-operative the system should be, and in what ways. These might indicate that the system should simplify things if the user is likely to be confused, or conversely to be precise and detailed. The planner might even be given a goal "lie to the user" (although this generator is intended to be ethical, and will thus have no plans that achieve this goal)! The distinction between these kinds of goals is not a sharp one, but since the planner responds to all of them in the same way (given that the plans for different kinds of goals will likely have different kinds of actions), there is no need to sharpen it.

More important is the distinction between foreground and background goals, which the planner does handle differently. Foreground goals are goals the planner explicitly tries to achieve. Background goals, on the other hand, are goals that the system considers useful but not essential. The planner therefore includes them in the plan if convenient but doesn't deliberately plan to achieve them. Generally foreground goals are communicative (e.g. "tell the user this information") and background goals are intersocial or co-operative (e.g. "be polite" or "be direct"), but this is not always the case. The system might be given background communicative goals such as "mention this object if it can be conveniently worked into an utterance" or foreground intersocial goals such as "say something deferential"). The foreground/background distinction is a distinction between reasons for adopting goals, not between the kind of actions in the goals.

This notion of foreground and background goals is similar to the one used by UCEgo [Chin 88]. There, however, the distinction is only captured in the way goals become active. Foreground goals are always active and hence always being worked on. Background goals are dormant until the system independently notices a plan for achieving one, at which point the background goal becomes

[22]How the system decides to use communication rather than some other action is an important question, but it lies outside the scope of this proposal.

(effectively) a foreground goal. The communicative planner here handles background goals similarly; it doesn't explicitly plan for them, but it prefers plans that achieve them. There is a crucial difference, though, when a plan (or sub-plan) that achieved a background goal has to be abandoned. In UCEgo, the background goal remains active even if there is no longer any plan for it under consideration. The communicative planner simply continues to prefer plans that achieve the background goal when it encounters them, since the importance of the goal to the overall system hasn't changed.

At heart, the communicative planner is a planner in the traditional AI sense. It is given one or more goals to achieve, and it consults a library of possible actions and plans, assembling a complete plan that achieves the goal(s) by searching through the space of possible plans. The planner's task is complicated, however, by several factors. The planner is operating in an active environment, so the situation may change in ways that invalidate the plan before it is carried out. The user may say something, thus changing the discourse context. The physical environment may change in ways that affect the conversation.[23] Furthermore, the goals given to the planner may be changed or withdrawn, either in response to a changing environment or because the overall system modified its intentions.

The communicative planner must also cope with the fact that the linguistic specialists can't always fulfill its requests precisely. From a traditional planning perspective, this amounts to not having complete descriptions of the actions available to be built into plans. The planner must be able to modify its plans when the specialists return expressions that only approximate the intended meaning. In some cases, of course, the deviations will not prevent the expression from achieving its intended goal, so the planner can ignore them. In other cases, though, the deviations will sabotage the plan, so the planner must either modify the plan (with as little disruption as possible) or push the specialists towards an alternative that will support the existing plan. This kind of interaction is not entirely negative, though; the specialists may sometimes propose expressions that suggest ways of improving or simplifying the plan. Ideally, the communicative planner should be able to take advantage of this.

To handle all of these difficulties, the planner must be a fully dynamic one; it must be able to cope with changes in its task, i.e. changing goals and environment, and limits on its knowledge of the available actions. The focus of this work, however, is on the generator and its internal structure. I therefore intend to ignore the problem of changing goals and environment and concentrate on getting the planner to respond to the options presented by the linguistic specialists. Thus the communicative planner will build an initial plan assuming it can unproblematically express whatever it wants to. The planner will then place requests for the information it wants to express into the workspace for the specialists and respond appropriately to the options they produce. The options may adequately express the requested information, in which case the planner need merely verify this and

[23]Of course this assumes that the system could detect such changes; this is generally not true now, but will presumably become more common as computer vision and other sensory devices become more sophisticated.

approve them. If the options turn out to differ from the planner's requests in ways that sabotage the plan, the planner must revise its plan or reformulate its request to push the specialists in the right direction (or both). This revision by the planner is a special case of the general need for dynamic planning, but it at least constrains the task to a particular kind of "environmental" change, and it seems to mostly require only localized changes in the plan. It is also the minimum dynamic planning behavior needed for the overall architecture to work, and is thus crucial in building the planner.[24]

2.4.1 Building a Plan

The communicative planner starts off by building an initial plan to achieve the goals it is given. Since the method the planner uses to do this is not a major focus of this work, I intend to use either some version of a standard AI planning algorithm (as in, for example, [Nilsson 80]) or a text planner from an existing generator such as TEXT or Penman, modified slightly to prefer plans that achieve background goals as well. The planner will draw on a library of text plans, which will probably be fairly ad hoc; the main point is to be able to construct a plan, not to worry about constructing it in a principled or general fashion.[25]

What is crucial is the form of the resulting plan. It cannot be just a sketch of the information to go into the text and its order. The plan must indicate what each piece of information in the plan is intended to accomplish and how it supports the purpose of the larger text containing it. Often the purpose for including some information is just to express that information, but not always; larger chunks of text almost always have some purpose beyond mere expressions. Examples of the type of plan the planner must build can be seen in Figure 2.3. This model of text plans meshes well with the model of discourse structure developed by Grosz and Sidner [Grosz 85,Grosz 86b], in which the purpose of each discourse segment is an important part of the structure. The text planner in Penman [Hovy 88b] produces plans along these lines and may be an appropriate model for this part of the planner.

[24]This division might suggest that the generator could be reorganized to eliminate feedback by letting the "planning submodule" of the planner send its plans to the linguistic specialists, and letting the "response submodule" just operate on the specialists' output. The division, though, is a simplification of the architecture for implementation purposes; in principle, "planning" and "evaluating" are not separable. More immediately, this reorganization will not work because the "response submodule" will sometimes need to modify the plan, and thus change the input to the linguistic specialists. So there is no escaping the need for a feedback cycle.

[25]I am implicitly assuming here that the communicative planner is separate from other planning facilities in the overall system, which is questionable. This separation seems necessary as a working assumption; building a general intelligent agent to integrate the planner with would be a much harder task than building the generator. Furthermore, text (or speech) plans do seem to be largely distinct from other plans; non-linguistic acts are only occasionally integrated into them. Still, such integration is possible, so the separation of the communicative planner from other kinds of planners must eventually be weakened.

1. Overall Goal: clarify concept `SHIP`

2. `(identification SHIP WATER-VEHICLE ...)`: directly fulfill (1)

3. `(evidence SHIP ...)`: support (2) by evidence

4. `(attributive SHIP ...)`: fulfill (1) by extending (2)

5. `(particular-illustration)`: support (2) by example

1. Overall Goal: Transmit weather information for next few days

2. Support (1) by sequencing information in day-by-day frame

3. Support (2) with first day's information

4. Support (2) with second day's information

⋮

N Support (2) with last day's information

Figure 2.3: Examples of Initial Plans

2.4.2 Responding to Linguistic Options

Once the communicative planner has built up an initial plan of what it wants to say, it puts requests for the individual bits of information it wants to express into the workspace for the linguistic specialist to respond to. These requests can contain both the actual information to be expressed and annotations indicating preferred properties of the expressions. These annotations can indicate such things as part of the information to be mentioned explicitly, or implicitly, or to be downplayed as much as possible; a framework within which objects should be portrayed, or an entity to be (or not to be) placed in focus. The annotations are part of the text plan, indicating how to express the information to best support the goals of the plan. They will be used by the linguistic specialists to alter their evaluation of different options, preferring ones that follow the annotations.

The planner, however, cannot assume that the annotations, or indeed the prescribed information, will be followed precisely; the linguistic specialists may not be able to find a way of completely satisfying the planner's requests. Furthermore, different options may support or conflict with background goals that were not considered when forming requests. The planner must therefore examine and evaluate the responses that the specialists put in the workspace. These responses consist of annotated linguistic structures; the annotations can include all the types of annotations that the planner can use; they can also indicate which parts of which requests the option covers, extra information the option expresses, parts of the information the option leaves out, and variations in tone or style.[26] The planner looks at these annotations (since it has no way to analyze the linguistic options themselves) to see how well they fit into the plan. Options that achieve their intended goals have their rating adjusted according to how well they do so. Options that help achieve background goals get a higher rating than they otherwise would. Options that don't achieve all their goals, or that interfere with other parts of the plan, can dealt with in two ways: the planner can just give them a low rating, or it can adjust the plan to achieve the unmet goals some other way.

The algorithm for all this is summarized in Figure 2.4. When the planner encounters an option that requires modifying the plan, it doesn't immediately adopt the modified plan, because it doesn't know yet whether the option will actually be used. Instead it simply places requests into the workspace for all the modified steps in the plan, and adds an annotation to the option indicating the plan modifications it induces. Subsequently, the utterer will decide whether or not to use the option, forcing the adoption of the corresponding version of the plan; it will then remove the requests and options pertaining to the other version(s). In the meantime, the planner and the specialists can work on producing options for both versions of the plan, allowing the overall suitability of each version to affect the decision of which one to use.

[26]The content of the annotations is one of the main foci of this research; this description should be considered a minimum list of what the annotations should include. Also, there are some annotations not mentioned here because the planner doesn't pay attention to them; see Sections 2.5 and 3.7.2.

The planner examines the annotations for each option, responding based on the type of the annotation:

- For extra information expressed:
  - if contained later (elsewhere?) in plan, then improve rating.
  - if tied into any current background goals, adjust it based on whether it supports or undercuts goal.
  - if not contained elsewhere in plan, then adjust based on system's interest in general in expressing this info.
- For missing info:
  - if can be added elsewhere to plan without disruption, do so and ignore.
  - if important and doesn't fit elsewhere, decrease rating
  - if not important, leave rating alone.
- For concepts activated in the context:
  - if contained later (elsewhere?) in plan, then improve rating.
  - if conflicting with something in plan, then decrease rating.
  - if not contained elsewhere in plan, then adjust based on system's interest in general in expressing this info.
- For things like tone, awkwardness, etc. adjust rating based on how they fit in with background goals.

Figure 2.4: Linguistic Option Evaluation Algorithm

2.5 The Linguistic Specialists

The linguistic specialists are the part of the system that actually uses linguistic knowledge to construct the expressions that will go into the utterance. They constantly watch the workspace, looking for requests from the planner, to which they respond by suggesting linguistic expressions that capture some or all of the information in the request. These expressions are placed in the workspace as options for the planner to evaluate, and are eventually either uttered or flushed by the utterer. The specialists continue suggesting options for a request as long as that request remains in the workspace. The options are pieces of surface structure (or perhaps phonetic structure in a speech system), and they can be at any level of structure and be only partially specified. Thus an option could be a full sentence, phrase, or word, or it could be a particular clause structure (e.g. a topicalization or it-cleft) with no further detail filled in, or a noun phrase with a determiner but with the head noun left unspecified. The wide range of types of expressions the specialists can produce is the main reason for splitting this component up into several pieces. It seems likely that different kinds of linguistic knowledge will be most naturally organized in different ways; thus there will be a different linguistic specialist for each kind of linguistic knowledge.[27]

The details of how the specialists work depends on how the knowledge they use is organized. There are, however, some common features. The linguistic specialists are primarily intelligent lookup components. What this means is that they don't do a lot of computation to construct the expressions they produce; they simply have a set of possible expressions, indexed in some fashion, in which they check for expressions matching the planner's request. They do have to verify that the expressions they find are really appropriate; the indexing is intended to speed up the lookup but not to guarantee finding a useful expression.[28] The linguistic information, however, is intended to be in a declarative form, so the specialists should be able to use it fairly straightforwardly.

The specialists, however, do not simply look up each individual request from the planner. They can decide to look up only part of a request, or to look for expressions that capture several requests. Each specialist has criteria for deciding when to broaden or narrow the scope of the request(s) it is responding to. In addition, the specialists always check to see if an expression they have looked up corresponds to a larger or smaller piece of the request(s) it was responding to. If so, they continue trying to look up the enlarged or reduced request as well as the previous request, and they tell all the other specialists to do so also. The reasoning behind this is that finding an expression for a collection of information indicates that the information seems to fit well into the structure of the language, and thus that there are likely to be other expressions that capture it.

The linguistic specialists are also responsible for annotating the options they

[27]Possibly more, if it turns out that there is more than one useful way of organizing a particular kind of linguistic knowledge.

[28]This is similar to the indexing scheme in PHRED [Jacobs 85], where the fetching of pattern-concept pairs uses a hashing scheme that directs it towards likely possibilities.

put into the workspace. These annotations allow the other components in the system to evaluate the options without having any linguistic knowledge of their own. (The annotations are discussed further in Section 3.7.2.) From the specialists point of view, there are several kinds of annotations:

- Some annotations are inherent parts of expressions. These include things like tone, style, fluency. The specialists simply notice these annotations when they look up the expression and include them in the entry on the workspace; they don't need to understand or manipulate them at all.[29]

- Some annotations indicate connections to other entries in the workspace. These indicate either which request(s) the option expresses or other options that the annotated option is dependent on. The former is trivial for the specialists to provide, and the latter reflects cases where a specialist has produced an option that needs the other option(s) in order to have the expected meaning.

- Some annotations indicate ways in which the information expressed by the option differs from the information in the request(s) it is responding to. These need to be computed by the specialists; there is no way to compute them in advance, because the request(s) the option is applied to may be different than the one(s) it was looked up for. The details of how this is done depend on the representation of meaning the generator is using, but a reasonable first approach is to compare the meaning representations simply list all the differences. These annotations are central to the planner's ability to evaluate the options; they are the heart of the connection between the planner and the specialists.

- Some annotations indicate additional effects that an option has beyond what it directly expresses. In particular, annotations may indicate that use of an option will add certain other objects or concepts to the current discourse context. These kinds of annotations may in some cases be directly associated with the option, but in general they will have to be computed by the specialists, because they depend on how the option relates to and affects other concepts in the system's world model.

The design of the linguistic specialists has strong implications for the organization of the linguistic knowledge they draw on. The specialists need to draw on knowledge of available linguistic structures and their contribution to the meaning of the utterance at all levels, so that they can provide the necessary annotations for the planner. This means that the meaning of expressions must be available; there can't simply be an opaque procedure that produces linguistic expressions

[29] Actually, sometimes these annotations may need to be computed, but the method for computing them will be part of the linguistic knowledge the specialists looks up. The specialists don't need to know how to compute them in advance.

with no information as to how or why they are appropriate.[30] Furthermore, the representation must allow for integration of different levels and types of knowledge. A particular piece of information might be expressed via intonation, a word, a phrase, a syntactic construction, or even implicitly by the ordering of other information.[31] The generator's linguistic knowledge must thus be organized to allow expressions at different linguistic levels to have similar meaning representations, so that they can be recognized as alternative options for requests from the planner.

2.6 The Utterer

The utterer is responsible for the issues involved in the final assembly and output of the utterance. The main concern in outputting the utterance is time pressure to utter something.[32] The system can't go too long without uttering *something* or it will lose its turn, because the user will either utter something else or get tired of waiting and leave. Also, when the utterer assembles utterances from the options in the workspace, it must worry about issues such as minimizing awkwardness, maximizing consistency of tone, and minimizing redundancy. Of course, these may conflict with each other and with the time pressure on the utterer, so the utterer must strike a balance among these criteria, according to the relative importance of each of them.

These issues cannot be handled by the communicative planner, even though they do affect how well the utterance accomplishes the system's goals, because they involve non-local aspects of the construction of the utterance. ("Local" here means local to one linguistic option in the workspace, not necessarily local to one part of the utterance.) The planner can only deal with local aspects of the utterance because, having no linguistic knowledge itself, it cannot anticipate the consequences of combining several options, even though it has descriptions of the effects of each individual option.[33] The utterer, on the other hand, does have access to the necessary knowledge to evaluate the combined options; indeed, that is its primary type of knowledge. This also allows it to judge when time pressure outweighs any deficiencies in the available options and assemble and send out an utterance.

The actual work of the utterer is fairly straightforward. The utterer watches the

[30]This is not quite the same as saying that the linguistic knowledge must be stored declaratively. There could very well be some opaque procedure that mediates access to the linguistic knowledge (perhaps compiled somehow from a declarative version) as long as it provides a declarative meaning representation, perhaps constructed on demand.

[31]e.g. the implication of temporal or causal information in conjoined clauses; see [Grice 75]

[32]The utterer should probably also be responsible for not uttering something too soon and for controlling the rate of utterance; these issues seem less crucial, however, and are not being addressed here.

[33]The planner can, of course, deal with non-local aspects of the text plan; it does so in evaluating the options and in deciding when and how to revise its plans. But what it understands are the global communicative effects, not the global linguistic effects.

workspace to see what the planner and the specialists put there. As the specialists respond to each request from the planner, the utterer identifies the best option or set of options for the request. Normally the utterer will have time to wait for the planner to evaluate the options, but if the system is in a hurry, the initial ranking by the specialists may be used. As several requests are processed by the specialists, the utterer keeps track of the best set of options to handle all of the requests (which may not include the best option for each individual request). The utterer evaluates the options based on their individual rankings modified by the effects of combining them. The effects of the combination of options are determined based on three factors: redundancy (i.e. more than one option expressing the same requested information), consistency of tone, and awkwardness (combining two or more awkward expressions magnifies the awkwardness of the individual expressions). The relative importance of these criteria is determined by parameters that can be set by either the communicative planner or the overall system.[34] The accumulated time pressure establishes a minimum rating for an option to be considered acceptable; this rating gradually decreases, representing the increasing importance of uttering something, even if it is not very useful. The utterer looks for options for the next request from the planner whose rating, when combined with options for other requests in the workspace, exceeds the minimum rating established by time pressure. Assuming the communicative planner has in fact made a request (i.e. assuming the system has decided to communicate), the minimum will eventually fall below the rating of some option in the workspace, making that option acceptable.[35]

When it finds one or more acceptable options, it outputs the highest rated one together with any other options that are needed to flesh it out syntactically (e.g. the subject, if the option is a verb whose subject has not yet been uttered).[36] The utterer then removes from the workspace all the requests for which it has just uttered options, and all the options for those requests. It also resets the minimum rating for acceptable utterances to its original value. Finally, if any of the options were marked as inducing revisions in the text plan, the utterer tells the planner to commit to those revisions, and the planner abandons any alternative versions of the plan and removes from the workspace any associated requests and options. This sets the generator up to continue working on the rest of the text plan, with the entries for the uttered options removed, since they are no longer needed.

[34]These seem to depend more on the situation than on the particular goals the generator is addressing; perhaps background goals are particularly relevant here.

[35]Of course, it may take a long time for this to happen, and the system may have abandoned its original goals in the meantime, or the user may have interrupted or quit. This is how this architecture would handle situations like Case 5 in [Hovy 87], where the cost of saying something inappropriate is very high, and the conversation is fairly short, so the generator doesn't have time to construct a highly rated utterance, and the minimum acceptable rating never falls very much.

[36]Note that this means that utterances will usually express information in the order the planner requested it, but some (usually local) variation may occur.

2.7 The Workspace

The workspace is the central communication channel for the other components of the generator; the planner, the specialists, and the utterer coordinate their activities through the entries they read and write in it. Each entry consists of three parts: a request from the planner, a set of linguistic options from the specialists, and ratings of how well the responses fulfill the requests. The requests can be annotated to indicate constraints on the expressions that realize them; the linguistic options can be annotated to indicate how well they match the requests, as well as to indicate certain other properties of the options (see Section 2.5 for more details and Section 2.8 for some examples). The entries are kept in the order that the requests come in from the planner; this is the only way that the workspace directly affects the output of the generator. Entries are eventually removed from the workspace by the utterer when it selects options from them, or occasionally by the planner if it removes the original request while revising its plan. Thus all the manipulation of the data is handled by other components; the workspace only has to store the entries and modifications to them.

Since the workspace can be modified by several components that run in parallel, it might seem that some sort of concurrency control is needed. Concurrency, however, is not a problem because of the way the different components use the workspace. The real dangers of unconstrained concurrency are that there might be contention for limited resources and that multiple simultaneous changes to shared data might cause some or all of the changes to be lost.[37] There are no limited resources here, though, so there's no problem of contention. Of course, access to the workspace itself is a kind of limited resource, but this can be dealt with simply by having the workspace respond to requests in the order they come in.

Furthermore, although there is multiple simultaneous access to the workspace, there are never simultaneous changes to the same piece of information. In fact, most of the modifications to the workspace are really adding new information, not changing what's already there. The only exception to this is the planner, which can change the ratings of options. The planner, though, is a single component, so it can only modify one rating at a time. Since there will never be two components trying to simultaneously modify the same data in the workspace, there is no need to protect against that possibility. There is, of course, the possibility of an attempt to modify or add to an entry that has been removed from the workspace; but this can be handled simply by throwing away any changes to deleted entries. This approach, along with responding to requests in the order they arrive, is the only concurrency control needed for the workspace.

[37]The simplest solution to the second problem, using some form of locking to control access to shared data, turns it into a case of the first problem. Since it turns out to be safe to make multiple simultaneous changes to the workspace, however, that's not a problem here.

2.8 An Example

A more detailed example of how the generator works will help indicate what the various components do and how they work together to generate an utterance. (The discussion here focuses on how the various components interact; Section 3.7 discusses in more detail how the linguistic specialists produce the options in this example.) Consider a case where the planner is given a goal to inform the user about the weather over the next three days. The planner then constructs an initial plan indicating what information to include and how to structure it to achieve this goal:

1. Overall Goal: Transmit weather information for next 3 days
2. Support (1) by sequencing information in day-by-day frame
3. Support (2) with information about the first day's weather
4. Support (2) with information about the second day's weather
5. Support (2) with information about the third day's weather

Requests to express the information about each day's weather are then placed in the workspace; (part of) the information about the first day would describe the weather:

```
(temperature within-range <60°F 80°F>) over-time-span
<September 25 1987>
```

Once this request is in the workspace, the linguistic specialists try to provide options to express it. From the specialists' point of view, the request has four parts to be expressed:[38]

a) `temperature within-range`

b) `<60°F 80°F>`

c) `over-time-span`

d) `<September 25 1987>`

The specialists provide several options for `temperature within-range`, annotated to indicate how well they express the information:[39]

- "The temperature be": `(Makes-explicit temperature)`
- "The weather be": `(Indirectly-suggests temperature)`
- "It be": `Simple-construction, (Makes-implicit temperature)`

Once these options are placed on the workspace, the planner can examine them and evaluate how well they support the plan.[40] In this case, the fact that the

[38]This is not inherent in the request, it's just how the specialists happen to respond in this case. In theory, a specialist might come up with a response for more than one of these parts, or for a smaller piece within one of them. To keep the example simple, however, I am ignoring this possibility.

[39]Note that only the relevant annotations are shown here.

[40]Note that in this example the specialists find reasonable options for all of the requested information, so the planner need only evaluate the options and not revise the plan.

system is talking about temperature is not an important part of the plan, so the planner prefers mentioning it implicitly. On the other hand, merely suggesting it indirectly obscures the intended meaning, so the planner rates the three options medium, low, and high, respectively.[41]

The rest of the request is handled in the same way. The specialists respond to <60°F 80°F> with:

- "high": (Missing-info (scale <request> temperature-scale))
- "warm": *no relevant annotations*
- o ["warmer":(From-context <previous-temperature>), (relates-to <previous-temperature>)]

The third option would only come up if there were some cooler temperature already contextually available; since this is the first day's weather, that is not the case here, so this option doesn't go into the workspace. As before, the temperature scale is not very important, but it's part of the information, so the planner ranks the first option low and the second option high.

Similarly, over-time-span gets the following options:

- tense=future: *no relevant annotations*
- "during": (Activates-in-context (instance <arg 2>[42] linear-span))
- "over": (Activates-in-context (instance <arg 2> linear-span))
- "on": (Activates-in-context (instance <arg 2> linear-position))

These options affect how the system (and the hearer) perceive the time span being talked about. Since the central organizing principle of the plan is the sequencing of the information by days, the planner wants to maintain the perception of the day as a point within the span of the next three days. Unfortunately, the middle two options activate the perception of the day as a span of time, so they are rated low. The first option leaves the question open, so it is rated medium. The last option reinforces the desired perception, so it gets a high rating.

Finally, the specialists tackle <September 25 1987>, producing:

- "tomorrow": (Activates-in-context few-days-time-scale)
- "Monday": (Activates-in-context week-time-scale)
- "the 25th": (Activates-in-context month-time-scale)

Here the planner's concern is to get the closest match to the time scale in the plan, so the three options are rated high, medium, and low respectively.

[41]The rankings given here are just relative ones, since the details of how numeric rankings should be assigned haven't been worked out yet, and because the relative rankings are all that really matters for this example.

[42]"<arg 2>" here means the second argument of the over-time-span relation in the planner's request; see page 90.

Now that the planner has ranked all the options for the request, the utterer tries to assemble an utterance from them.[43] It first tries to assemble the highest rated option for each part of the request:

a) It be

b) warm

c) on

d) tomorrow

This, however has a problem: the combination "on tomorrow" is awkward (or maybe even ungrammatical). So the utterer tries to replace either "on" or "tomorrow" with the next-highest rated alternative. There are two ways it can do this: by using tense instead of "on" to express `over-time-span`, or by using "Monday" instead of "tomorrow" to express `<September 25 1987>`. The ratings used here provide no way to choose between them (although a more fine-grained rating system might), so the utterer will simply choose one at random. The final result is one of:

- "It will be warm tomorrow"
- "It will be warm on Monday"

either one of which is a reasonable way to express the planner's request.

This example indicates how the architecture organizes and coordinates the generation of language. The communicative planner constructs an initial plan indicating what information needs to be expressed and why. It also selects from the linguistic options for that information based on how well each option supports the purpose for including the information in the plan. The linguistic specialists indicate what options the language provides for expressing information and the particular consequences of choosing each option. The utterer assembles options for multiple pieces of information, resolving conflicts between different options. These components work independently, using the entries in the workspace to coordinate their activity as they build up utterances in natural language.

[43] Actually, the utterer starts trying to assemble an utterance as soon as any options get into the workspace. Assuming that time pressure is not serious, though, it will usually have time to wait until the entire request has options produced by the specialists and ranked by the planner.

Chapter 3

Representation

3.1 Representing Linguistic and Conceptual Knowledge

What does a generator need to know in order to choose appropriate linguistic structures? One obvious answer is that it needs to know the grammar of the language that it is using. The generator must be able to construct utterances that conform (more or less) to the constraints the language obeys. This includes, of course, syntactic constraints as well as constraints at lower levels (e.g. appropriate use of affixes and contractions) and higher levels (e.g. producing coherent sequences of utterances). Indeed, much of the work on generation is organized around the particular formalism used to enforce grammatical and other constraints. Thus, for example, the NIGEL generator is designed around systemic grammar, PHRAN and KING are designed around unification grammar, MUMBLE is designed around its own notion of grammar (integrated somewhat with TAGs). Of course, there is more to all of these generators than this; but there is a general implicit assumption that the main linguistic knowledge that the generator uses is simply knowledge of what is grammatically acceptable.

This kind of grammatical knowledge is, of course, necessary; the generator has to be able to produce reasonably well-formed utterances. The precise nature of how this information is represented, though, is not really very important. There are currently a large number of grammatical formalisms available (e.g. unification grammar, systemic grammar, GPSG, HPSG, LFG, TAG, GB, DCG) that have been used to build computational grammars. They all have strengths and weaknesses, and the bottom line is that none of them provide a complete solution to the problems of describing the grammar of English (or any other language). Thus the choice of a particular formalism for representing grammatical knowledge is not a critical one, particularly since a grammar for generation can safely be conservatively designed. If there is some doubt as to precisely when some construction is grammatical, the doubtful cases can be ruled out; the generator will simply not use the construction in some cases where it would have been acceptable.[1]

[1]This contrasts with the case of an analysis grammar, which must be able to handle anything

Thus any of the existing formalisms will do an adequate but not perfect job of representing the grammatical knowledge the generator needs.

What is more important is the question of what the generator needs to know to choose the particular linguistic elements to make up its utterances. This is really a two-part question: what kinds of choices does the generator make, and what does it need to know in order to make them? In the broadest sense, the first answer is that it has to choose among the entire range of possible (i.e. grammatical) utterances. But this is too broad an answer, since the concern here is with what kind of *linguistic* choices the generator must make. In linguistic terms, the generator must choose between various words, phrases, syntactic constructions, discourse structures, etc., selecting the ones that do the best job of communicating what it is trying to express.

The problem of linguistic choice has generally been treated as straightforward, particularly in lexical choice. For example, the NIGEL grammar handles lexical choice by simply attaching words to the nodes in its semantic network [Sondheimer 86]. If it wants to express a concept that doesn't have a word attached to it, NIGEL simply combines the words associated with the parent concept and distinguishing feature(s) into a phrase.[2] There is no real "choice" involved; every concept has a unique word or phrase associated with it in the network. A similar scheme was proposed for MUMBLE's lexicon [McDonald 79] (although it doesn't seem to have ever been used). In general, MUMBLE handles linguistic choice by simply listing all possible realizations for each domain concept together with any linguistic constraints; the first option whose constraints are satisfied is used [Meteer 87]. There is no real linguistic knowledge being applied here; whatever linguistic knowledge the system has is compiled into ad hoc associations between domain terms and specific linguistic structures.

This absence of general linguistic knowledge is shared by discrimination-net-based systems such as Danlos's [Danlos 87] or BABEL [Goldman 75]. While the lexical choice in these systems may ultimately be based on sophisticated linguistic information, that information has been compiled out; the generator simply has a fixed set of decision criteria. The system can't apply the knowledge underlying the discrimination net to another situation, nor can it detect when other criteria ought to override the discrimination net.

Even generators that do have and use a body of general linguistic knowledge are concerned more with finding an acceptable choice than with using their knowledge to support the generator's goals. The PHRED generator [Jacobs 85] represents its linguistic knowledge as a set of pattern-concept pairs that provide

---

the user produces. This is problematic, since ideally the same grammar will handle both analysis and generation. Ultimately, given a complete and accurate grammar for English, the problem will disappear. In the (no-doubt lengthy) interim, a reasonable approach would be to indicate uncertainties in the grammar and let the generator be conservative and the analyzer be liberal.

[2]The NIGEL grammar does devote a lot of effort to dealing with *syntactic* choices. But the systemic approach centers mainly around organizing the various options and the connections between them. The issue of how to make the choices is pushed off into the "choosers", which appear to be essentially ad hoc.

general connections between linguistic structures and domain concepts. Unlike MUMBLE, these PC pairs are not in any way tied to the specific structure or workings of the generator; in fact, the same PC pairs were also used in PHRAN to parse and interpret input from the user. This general representation was not used, though, to support a more flexible approach to constructing utterances; PHRED simply uses the first acceptable utterance it can find. PHRED doesn't use its representation to expand the kinds of reasoning it does about linguistic choices, but merely to give a more principled basis to the choices it makes. The same is true of KING, which builds on PHRED's approach to generation [Jacobs 87]. KING uses a knowledge representation formalism (ACE) that allows a concept to be viewed as an instance of another concept, e.g. "John kissed Mary" could be viewed as an instance of transfer from John to Mary, allowing "John gave Mary a kiss" to be generated. As in PHRED, though, this representational sophistication expands the *range* of linguistic choice, but not the *criteria* for choice. KING simply finds the most specific acceptable way of viewing a concept, and then selects and instantiates a pattern in a fashion similar to PHRED. Similarly, COMET uses information about collocations to guide lexical choice; whenever a lexical item is chosen, all the words it frequently co-occurs with are immediately brought out for consideration [Smadja 90]. This is an automatic procedure, though; COMET has no awareness of the consequence of using (or not using) a collocational pair. In fact, if a sentence contains two collocations that enforce conflicting constraints on a particular word choice, the system has no way to recover. In none of these systems is there any reasoning about which choices would better serve the overall task of the generator. All linguistic choices not ruled out are considered equally suitable, and if *all* the choices are unsuitable, the generator has no way to choose a "least bad" option.

All of these systems work, of course, in the sense that they produce comprehensible output that says the right thing. In fact, some of them can generate quite sophisticated and fluent text. Yet they miss an important aspect of how language is used. In natural speech and writing, linguistic choices serve not just to transmit information in acceptable (i.e. grammatical) form, but also to support the speaker's or writer's aims in more subtle ways. The sensitivity of linguistic choice to more than just the literal information being expressed can be seen in some examples from a (randomly chosen) newspaper article [Rossi 87]. The article discusses an Indian chief who is leading opposition to a highway project because it will disturb Indian graves. While it's not possible to explore how particular choices were made in the construction of the article, since it was presumably revised and edited to some extent, the way that particular linguistic choices support larger goals is apparent.

Consider the following examples: (all italics added)[3]

[3]The discussion of these examples will occasionally refer to Indian beliefs. I make no claim that these references are at all correct; they are intended rather to reflect the popular conception of what Indians believe. I assume that, like most things, Indian tradition is more complex and more varied than the popular conception.

(13) He [Roy Crazy Horse] is one of three chiefs the state *routinely* deals with. [p. 44]

(14) If you talk to Wyandaga, it's pretty obvious there's *bad blood* between them. [p. 44]

(15) On a Sunday afternoon in *late spring*, a small group of Indians waits for something to happen. Wyandaga sits quietly at a picnic table. [p. 45]

Why is the word "routinely" used in (13) rather than, say "normally" or "usually"? The article contrasts Wynadaga (its main subject) with Chief Crazy Horse; Wyandaga is rebellious and outside the system, whereas Crazy Horse co-operates and works within the system. Using "routinely" supports this contrast by suggesting that Crazy Horse is part of the "routine" of the state government; "normally" or "usually" would not have this connotation, even though they would convey the basic intended meaning at least as well. In (14), "bad blood" conjures up images of blood feuds and primitive tribal conflict that draws on and reinforces an image of "primitive" Indian life in a way that alternatives such as "a lot of anger" or "a long history" wouldn't. "Late spring", in (15), is a vague term. Why not give the actual date, or if the author doesn't want to suggest that the particular date has some significance, at least the month? The term "late spring", though, ties the time to the season, reflecting Indian concern for nature and the flow of the natural world, and setting up the subsequent description of an Indian ceremony in the woods. In all of these examples, the what makes the actual choice superior to the alternatives isn't that it expresses the intended meaning better; the alternatives do that just as well. Rather, the actual choice's advantage is that it better reinforces some other aspect of what the article is trying to portray.

This can be seen more directly in cases where the article describes the same thing in more than one way:

(16) a. If the archaeologists suspect there is a *burial site*, . . . [p. 44]

b. He added that "if it were a *burial ground*, I would be there." [p. 44]

(17) a. . . . he [Wyandaga] is forced by his *religion* to oppose them [p. 44]

b. "he mixes a touch of *myth*, a touch of tradition, maybe a touch of archaeology, and it's sort of a mixture of everything." [p. 44]

(18) The chief's voice *echoes* in the trees; a dozen voices *follow*. About a hundred yards off, on the main road, there is a *sound*. At first it is a distant *roar*, impossible to identify, but as it closes in, it becomes obvious: It is the *cry* of a car without a muffler. [p. 45]

In (16)a, the term "burial site" is used when discussing the actions of archaeologists, reflecting the clinical, professional attitude they have towards the graves. On the other hand, Chief Crazy Horse describes the same place as a (purported) "burial ground", reflecting the more nature-centered Indian view. An even stronger contrast is seen in (17)a, where what Wyandaga sees as his "religion" is viewed by an archaeologist as merely "myth"; what one sees as a true way of approaching spirituality the other sees as merely an interesting collection of stories. What's

at issue here isn't what's being talked about; after all, the graves are the same whether they're called a "site" or a "ground", and Wyandaga's beliefs are the same regardless of whether they are called "myth" or "religion". The issue is how the reader is going to *conceive* of the referent and relate it to other things discussed in the article. Thus it is even possible to use several different terms to describe the same thing in rapid succession, as in (18). Here several terms are used in two consecutive sentences to describe the same noise. First it is a "sound", leaving it uncertain and mysterious. Next it is a "roar", making it seem ominous and perhaps a little frightening. Then it becomes a "cry", making it identifiable as the call of an animal. This succession of terms turns the car, a machine, into a kind of stalking animal, more in keeping with the nature-oriented Indian view of the world adopted here. It also serves a more local function of tying the sentences together into a complete little story, with the introduction of a mystery, rising tension, and a climax that relieves the tension and clears up the mystery. This dramatic structure would still work without the varied terminology, but the choice of words reflects and supports that structure. This use of linguistic choice to support the local discourse structure can also be seen in the first sentence in (18), where the choice of "echoes" and "follows" suggests that the other voices are also "echoes".

The kind of choice in these examples is beyond the capability of most generators because of their assumption that linguistic (especially lexical) choice involves locating the most appropriate option in a static space of possibilities. The only relevant factors are the information to be expressed by the choice and the set of available options. But in fact, as these examples show, the generator needs to consider many other factors, such as the purpose(s) driving the generation, other things being talked about, how the current choice fits into the overall structure of the utterances being generated, the knowledge, attitudes and beliefs that the system holds regarding the subject under discussion, and what the system knows about the hearer's knowledge, beliefs, and attitudes. The set of factors relevant to the generator's decisions about how to talk about comprise the system's *perspective* ; i.e. the way it currently perceives the world. The notion of perspective is a kind of extension of the common notion of a "discourse context" as including everything directly mentioned or associated with something directly mentioned in the discourse. The perspective includes not just the discourse context but also whatever ideas and beliefs the system currently has about whatever is being discussed. Only by being sensitive to the entire perspective will a generator be able to make the kinds of choices shown here.

There is, in fact, at least one generator that does seem to be sensitive to perspective. PAULINE [Hovy 88a] explicitly considers pragmatic issues such as formality, social interaction, and time pressure in constructing its utterances. For example, when generating a description of an anti-apartheid protest, PAULINE can say "officials removed the shantytown" if it supports the officials, and "the shantytown was destroyed by officials" when it supports the protesters [Hovy 88c]. But PAULINE doesn't provide any guidance on how to design a *representation* for a generator, because its realization component is essentially ad hoc; it consists

of a set of independent specialists that are not constrained in any way. While PAULINE does use Conceptual Dependency [Schank 75] as an underlying representation, the information that it uses to realize pragmatic effects is encoded separately. Thus we are still confronted with the problem of designing a representation that can provide the generator with everything it needs to know.

The obvious solution would be to simply incorporate all the factors making up the system's perspective into the structure of the space of available choices. Every element of the representation could be marked to indicate what the system's current attitude was towards it, whether it was currently being talked about, how it relates to the system's current goals, etc. This isn't possible, though. The range of relevant factors is unbounded, and thus cannot be compiled out in advance of the need to use them. The examples discussed above demonstrate that linguistic choice is potentially dependent on a particular system of understanding and describing the world. The terms "burial site" and "burial ground", for example, are equally good for explaining what sort of place is being discussed; the difference is in what role such places are to play in the world. Similarly, "late spring" is actually less useful than "May" in describing the time of the Indian meeting. But it encourages the reader to think of the passage of time in natural terms, as the Indians do, rather than in terms of an artificially structured calendar, as Western civilization does. The particular view of the world from which the system is speaking can affect any of the choices it has to make, in arbitrary ways, because it determines which aspects of the things the system is talking about are important and which are unimportant. But these determinations are not absolute; if precision is very important, then "May 6th" is a better choice than "late spring". Furthermore, the system must be able to adapt to different world views as it deals with different situations and different people.[4] So there is no way to arbitrarily select a fixed set of possible relevant factors; the generator must be able to consider the effects of various different perspectives, and also be able to override them when appropriate.

Even if it were possible to figure out all the different ways the system might need to view the world, it still wouldn't be possible to work out all the relevant factors in advance. First, the amount of information involved would be overwhelming; every bit of knowledge that the system has could potentially be relevant to every linguistic choice. The more serious problem, though, would be that the system would still need to take on other points of view besides its own. There are two reasons for this: the system may need to talk about other people's views, and it may need to adapt to the needs of the person it is communicating with. Note that the phrases "burial ground" and "burial site" both occur in the same article. The different phrases are used because the writer is presenting the views of different people, and thus writes from their perspective. A generator that is part of an intelligent system will need to be able to produce descriptions of the reasoning, beliefs, and desires of other people; to do this properly, it must be

[4]Of course, a special-purpose system might be fine with just one world view encoded into its linguistic and conceptual knowledge. But a generator that is part of a truly intelligent system must be more flexible and general.

able to adopt their perspective. Similarly, the generator may need to work from the user's perspective in order to successfully achieve the goals it is working on. Referring to the user's religious beliefs, for example, as "myths" or "fables" could lead to trouble. Thus the linguistic choices the generator makes must be capable of being sensitive to any possible perspective.

The inherent fuzziness and inconsistency of an intelligent agent's model of the world also prevents the perspective from being compiled out in advance. It has often been noted that many concepts have "fuzzy" edges; it's not possible to precisely delimit their meanings (see, for example, [Labov 73,Rosch 75,Lakoff 72]). How big does a rock have to be to be "large", for example? Or what is a "toy"? A doll house is presumably a toy, even if it's part of an adult's collection of doll houses, and a computer presumably isn't, even if a child plays games on it. But what about a baseball? Is it a toy if a child uses it, but not if it's used in a professional baseball game? What if it's used in a professional game and then given to a child to play with? No matter how well-specified a concept may seem, there are always cases where its applicability is uncertain. Thus it is impossible to compile out in advance a complete specification of the concept in every possible perspective; there would always be further perspectives to consider that differed in relevant ways.

Similarly, an intelligent agent's world model will be inherently inconsistent. This problem has often been noted in the case of inheritance networks with exceptions (e.g. [Brachman 85]). For example, if we know Clyde is an elephant, and we know that elephants are gray, then we can infer that Clyde is gray; if we also know that albino elephants are elephants, and that albino elephants are white, and that Clyde is an albino elephant, then we can also infer that Clyde is white, producing a contradiction. This kind of inconsistency is not limited to property inheritance, though. For example, a hypothetical Mitch Marcus might know that the fastest way to get to work is to take Wissahickon Drive. But he might hear on the radio one morning that there's been a terrible accident and Wissahickon Drive has a two hour backup, from which he would hopefully infer that Wissahickon Drive is *not* the best way to get to work today. Inconsistencies like these run rampant through the world model; it would be impossible to resolve them all in advance.

In fact, it would be a mistake to try to compile out the fuzziness and inconsistency of the world model, because the system can take advantage of them to be more flexible and expressive. For example, strongly connotative words such as "jerk" or "blunder" work by being extended over the person or action described; they are used primarily for the implications they make, not for classification. Their fuzziness is what makes them useful; if they had sharp edges, they could only be used in cases where their connotation was already clear. Similarly, the inconsistencies in the world model allow for ignoring irrelevant details; the system need only believe that "elephants are gray" or "Wissahickon Drive is the best route" rather than "elephants that aren't albino elephants are gray" or "Wissahickon Drive is the best route unless there is an accident or it's closed for construction or...". People generally only worry about the inconsistencies in their

knowledge when they bump into them explicitly, and then they just deal with the immediate consequences. Furthermore, inconsistencies can be handled just as easily by revising the categories and inference principles involved as by detecting a mistaken inference or an invalid piece of evidence; or the inconsistency may just be left intact if it's not important enough to worry about. The appropriate response depends on how the inconsistency came up, what the implications of each possible resolution are, what the system is planning to do with the information, etc.; in other words, the appropriate response to the detection of an inconsistency depends on the perspective. Thus it's not possible to build a complete map of the system's conceptual space to use to look up the options in, because the conceptual space is fluid and incoherent.

On the other hand, a map is just what is needed. If all the generator needed to do was look through some options and find one that matched its intentions, then it would be sufficient to simply evaluate each option and get a yes or no answer. Unfortunately, as discussed in Section 2.1, the generator can't assume there will be any options that precisely match its intentions. In order to choose between options, the generator must be able to determine how well the options match its intentions, and precisely how they differ. This would be simple if each linguistic choice were positioned at a particular point in a fixed map of semantic space, as (for example) in N GEL [Sondheimer 86]; then the difference between the option and the system's intention would simply be the path between them indicated by this map. For example, if the generator wanted an expression for the concept "house", the map of conceptual space would indicate that the word "house" was an exact match and that the word "cottage" added the information that the house is small. This could be simply read off the semantic map, in which "cottage" would be a sub-concept of "house" with the additional attribute "small". The generator could then decide whether it wanted to include the additional information, and choose between the options on that basis. Similarly, options that leave out information or overlap with the intended meaning would be simple to detect; "house" is just "cottage" without the indication of smallness, and "mansion" is like "cottage" except that it's large instead of small. All of this can be determined immediately from a map of semantic space; without some sort of map it becomes difficult or impossible. It's not enough to have a reasoning system that can tell whether the intended meaning follows from the proposed expression; the system would also have to determine what else follows from the proposed expression, and, if the intended meaning doesn't follow, why it doesn't. This would require a meta-reasoning system, a notoriously difficult problem.

Thus we are confronted with a dilemma: the generator needs a conceptual representation that it can read like a map, yet this can't be provided without compromising the flexibility the generator needs. The resolution of this dilemma is clear; what is needed is a representational framework that can indicate the differences between concepts that a map would show without actually containing a fixed map of the conceptual space. Unfortunately, the construction of such a representation is beyond the scope of this work. The best that can be done is to sketch out what would be required for a representation to be a solution to this

dilemma, and then try to build a representation that captures enough of this for the generator to work properly. The representation's design will sacrifice efficiency and adequacy for other tasks such as general reasoning and learning in order to fulfill the generator's needs. Thus the generator will work as if it could really on a truly general representation, even though the representation will simplify or ignore many of the important issues involved in representing knowledge for general use.

3.2 Requirements on the Representational Framework

The principal difficulty in designing a representational framework is that it must be sensitive to perspective; that is, the linguistic and conceptual representations must be able to "represent" different views of the world. A change in perspective can, in effect, change both the meanings of words and the state of the world. The representational framework must therefore be able to shift automatically between various partially overlapping models of the the world. Furthermore, these models must be created and updated dynamically, because the system may adopt new perspectives that couldn't be anticipated.

The system's perspective on an object or concept can be affected by almost anything, but most of the relevant factors fall within a few categories:

- "Properties" of objects and concepts, i.e. what the system knows about them. There's nothing very surprising or exciting about the fact that this affects how the system views things, but it's important not to overlook it.

- Other related objects and concepts. Anything that an object or concept might be connected with or compared with can affect how the system views it. This could involve simple local changes such as whether a person is "tall" depending on who they are standing next to; or it could involve large-scale effects such as different domains using completely different vocabulary to describe the same object.

- Purposes and attitude of the system with regard to the object or concept. If an object has no connection to what the system is currently doing, and has no intrinsic importance, very little information about it will be in perspective; in fact, the object may not be visible at all to the system. Similarly, the system's perspective on an object that it needs to perform some task will change significantly if the system finds a better way to do the task; the object may change from being desirable to being a nuisance (e.g. if it is bulky and in the way).

Perspective controls how much of the system's knowledge about an object is visible at any given time; it can change both the amount and the choice of information that is available. The effects of perspective show up in a number of ways:

Specificity Objects and concepts can be thought of and talked about at differing levels of detail. This shows up in "basic level" effects, in which there is a basic level of detail which people will naturally tend to choose [Rosch 76]. For example, if a person sees a dog running down the street, they're more likely to call it a "dog" than to refer to it as an "animal" or a "beagle"; and they're very unlikely to call it a "creature" or a "long-haired beagle-terrier mix". It's not that the person doesn't *know* that the dog is a beagle; rather the concept "dog" is a more basic descriptive term. The information that it's a beagle is simply further detail describing the object. What constitutes a "basic" level of description varies with the perspective, though. When talking to a breeder at a dog show, for example, the same dog would be more likely described as a "beagle" rather than as just a "dog". Thus the perspective affects which concepts are focused on as the most basic ones.

Perspective also affects whether details are visible or "forgotten" temporarily. Thus, for example, I might think that my local convenience store is always open, even though I know it's closed on Christmas and New Year's; those details are just not relevant if I'm just trying to decide where to get a newspaper. Or a financial advising system might know that municipal bonds are a good investment in certain cases, oblivious to the fact that they're not very good for non-U.S. citizens who won't get the tax exemption on them.[5] Hopefully these details will be brought into perspective if it is New Year's or if the user is a foreigner; in the normal case, though, they can be safely filtered out by the perspective.

Domain Different domains will view things in different ways, and hence will use different concepts to describe and reason about them. For example, the same building might be thought of as "home" when planning leisure activities, a "property" when considering real estate taxes, an "investment" or a "tax shelter" when considering financial planning, or an "eyesore" when considering aesthetics. The range of concerns relevant to the domain affect which properties of the object are relevant; thus a change in the domain can change the range of conceptual vocabulary that makes useful distinctions. For example, when thinking or talking about kitchen supplies, if the concern is for household expenses, the various items might be characterized on the basis of how they were obtained, using concepts such as gift, purchase, prize, reward, or loan. On the other hand, if the concern is for what to use at different kinds of meals, they might be characterized as junk, disposable, utility, fancy, or elegant items. And if the concern is just to find the right implement for a particular task, they might be characterized simply as plates, cups, bowls, knives, forks, spoons, trays, and so on. This kind of sensitivity to the domain-related perspectives is essential to working in multiple domains, because otherwise the knowledge used in each domain has to be kept completely separate; there is no way to keep the irrelevant

[5]This is only an illustration; don't take it as actual useful advice!

aspects of other domains from appearing and distracting the system.[6]

"Lexical" Presuppositions Words and concepts include implicit background assumptions; if these don't hold, then it's generally not possible to apply the concept. For example, the concept of a "lie" presumes a whole theory of basic cooperative behavior [Sweetser 83]. An utterance can be considered a lie only if it's in a situation where that theory applies; thus someone acting in a play, or reading a story, or making a joke wouldn't be lying, even if what they said was false and was believed by the hearer. Similarly, "steal" presumes notions of property (otherwise the objects are merely being "taken"); "tax" assumes that the organization is a government (otherwise it's "extortion" or "theft"); "bachelor" implies that the person is subject to the normal process of courting and marriage (and hence isn't, say, a priest). The presuppositions are implied even if the term is being denied. Thus the perspective must support the presuppositions in order for the concepts to be available. When reasoning or talking about actors in a play, the concept of a "lie" shouldn't be available; when talking about the Pope, the concept of "bachelor" shouldn't be available.

Reinforcing Factors If some concept is of questionable applicability, it can be reinforced (or invalidated) by the presence (or absence) of independent factors commonly associated with the concept. For example, eating a bagel may not count as "breakfast" if I'm by myself, but if my wife is sitting with me and having a bagel, eggs, and some cereal, then it probably is "breakfast". This distinction is entirely independent of anything I do. And this is a real distinction, because I'm likely to respond differently in each case if someone asks me if I've had breakfast today. Similarly, if I go to a conference for 3 days, skip all the sessions except one, and spend the whole time sightseeing, it's still not a "vacation", while if I go to the same city on my own, and attend a linguistics talk at the local university that I happened to hear about, and spend the rest of the time sightseeing, it is a "vacation". The distinction is not in what I did, but rather in how I fit it into the overall pattern of my life. Thus "vacations" are part of my personal life, and "conferences" are part of my professional life, and this is the determining factor.

If the meaning of concepts can change with the system's perspective, then what happens to the notion of the definition of a word or concept? The answer is simple: it disappears. There cannot be any distinction between "definitional" knowledge and "factual" knowledge; there is neither a coherent way to make the distinction nor a difference in how the two kinds of knowledge are used by the system. Of course, the representation will still contain the information that makes up the "definition" of its concepts; it could hardly function without it.

[6]McCoy's ROMPER system [McCoy 85] uses a notion of "object perspective" to focus on the most salient parts of its domain model; see Section 3.5 for a comparison of the notions of perspective in ROMPER and the current work.

There should even be a representation of the concept of a "definition". But this information will have the same status and be represented in the same way as the "factual" information that isn't considered part of a definition.

The reason for this is, paradoxically, that separating definitional and factual knowledge makes it impossible to modify the definition of a concept independently of what the system knows about the concept. This can be seen in the attempts to give a rigorous interpretation to the concept taxonomy in KRYPTON [Brachman 83]. Brachman, Fikes, and Levesque argue that concepts should be replaced with new concepts rather than modifying their definitions. For example, they argue against correcting the (mistaken) notion that whales are fish by simply detaching the whale concept from the fish concept and attaching it to the mammal concept. Since definitions simply equate terms with combinations of primitive predicates, changing the definition would amount to changing the combination of predicates to some other combination. "In reality what has been achieved is a change in the structure of the Concept of a fish with properties a, b, and c to that of the Concept of a mammal with properties a, b, and c—not a terribly well-motivated move."[Brachman 83, p. 4]

So how could the system correct its mistake? The only possibility in KRYPTON would be to create a new whale concept with the correct definition. The problem with this approach is that there is no way to relate the new and old concepts. Any previous inferences about whales used the old whale concept. Thus there are only two options: throw out everything the system knows about whales except the definition, or assume that anything inferred using the old concept applies to the new one as well. The former option throws out a lot of useful information (e.g that whales are the largest animals, or that whales are in danger of extinction); this doesn't seem reasonable. Why should correcting a misconception about whales make the system forget everything else it knows about them? The latter option, though, would maintain the belief that whales are fish, since that's something the system could infer, thus making it impossible to get rid of the misconception at all! What the system really needs is not to replace its whale concept at all; it really does need to modify it from "a fish with properties a, b, and c" to " a mammal with properties a, b, and c".

As Brachman et. al. point out, though, modifying concepts in this way is incompatible with a definitional knowledge representation. So the notion of a definitional representation will have to be abandoned. This shouldn't really be surprising, though; after all, what can be known about concepts independent of knowledge about the world is essentially nothing. The only knowledge that is independent of facts about the world is tautological, i.e. vacuous. Definitions, as used in mathematics and in KRYPTON, are not really a kind of knowledge at all; they are merely convenient abbreviations for complex expressions. Thus the attempt to separate the knowledge representation into definitional and factual components is unworkable.

The attempted distinction between between definitional and factual knowledge is very similar to the traditional philosophical distinction between analytic and synthetic statements (see, for example, [Carnap 37,Carnap 47]); Brachman et. al.

even describe the terminological component of KRYPTON as able to "answer question about analytical relationships among [its] terms."[Brachman 83, p. 6] The problems with the analytic/synthetic distinction have been shown by Quine (among others, see for example [Quine 53] and [Quine 60]). In fact, Quine's critique of analytic philosophy underlies much of the approach taken here in analyzing intelligent behavior; in particular, the representation developed here has been modeled after some of the ideas in **The Web of Belief** [Quine 70] (although the emphasis and the details are quite different). Quine's critique proceeds on theoretical grounds; the basic problem (oversimplified) is that there is no way to define analyticity without having a metalanguage to discuss ordinary language in, and there is no such metalanguage. The problem with the definitional/factual split here is a more practical one; there is no way to maintain the split and still allow the kind of knowledge revision that the system needs.

This criticism only applies if the terminological component is interpreted definitionally; if it is taken as merely asserting certain specific kinds of relations between concepts, then the problems with modifying definitions go away. Such an assertional terminological component might still be useful as a convenient and/or efficient way of representing "definitional" information. Even this approach won't work, though, because the information that is considered definitional varies with the perspective. "Spring", for example, might be the time between the vernal equinox and the summer solstice to an astronomer; the time when the trees and flowers bloom to a botanist; and the time when the weather gets nicer to a layman; or it might be all three to the same person at different times. Similarly, "water" might be a compound of hydrogen and oxygen to a chemist, and a clear, tasteless beverage to a waiter. Which information goes in the (assertional) terminological component and which in the factual component would have to change to reflect these differences. Thus there can be no fixed division into these two components.

This doesn't mean the representation must be completely uniform, though; in fact, the representation proposed below will be built with several different kinds of pieces. What is important is that the knowledge base present a uniform interface to the rest of the system. The internal organization of the knowledge base can't reflect fundamentally different kinds of knowledge, because perspective shifts can change which category information fits into. A particular piece of information could be considered a definition in one perspective, a fact in another, and a reasoning principle in yet another. Thus the representation can't have those distinctions built into it.

Instead, the knowledge base must use perspective as a kind of filter to present the current view of (part of) the world. As the system looks up information in the knowledge base, the perspective helps pick out which parts of the fuzzy and inconsistent model will be visible to the system. Thus the knowledge representation provides the illusion of a coherent and consistent world model, a kind of dynamically varying, on-demand partial map of semantic and conceptual space; which is just what the generator needs. At least, it would in an ideal world; the representation developed below will only partially meet these goals.

3.3 Sketch of the Representational Framework

The representational framework used in this work will not fully meet the requirements laid out in the previous section. Building a representation that did satisfy them completely would be a difficult task, beyond the scope of the current work's focus on the generator. Instead, the goal will be to construct a representational framework that adequately supports the generator's needs; in particular, the handling of perspective will be somewhat oversimplified compared to what a full intelligent system would need. The generator, however, will get the same kind of information that it would get from a more sophisticated representational framework, so it will still be able to provide the services an intelligent agent would need.

The representational framework will be kept fairly simple, since the aim is to merely meet the needs of the generator. More complex mechanisms that might be needed to support a complete system's needs can be left out, making the construction and use of the representation simpler and more straightforward. In addition to its practical advantages, keeping the representation simple will allow for complexity to emerge only where it is needed. As the generator is built, it may prove necessary to augment the representational framework; thus the representation arrived at in the completed system will indicate the simplest possible framework that can do the job. Furthermore, a simple representation framework will also improve the system's flexibility; anything built into the representation framework is fixed for all time. Things merely *represented* using the framework, on the other hand, are easily modified as needed.

In line with this approach, as little information and reasoning as possible will be built into the framework; instead it will be simply represented as part of the system's knowledge whenever possible. For example, property inheritance will not be explicitly built into the system. Instead, there will be an "isa" link marking subconcept relations and an inference rule saying that when two concepts have an "isa" link, one of them can share the properties of the other.[7] Many other things that might plausibly be built into a knowledge representation (e.g. the figure/ground distinction, spatial position and motion, and quite a few other things) will also be handled by explicit modeling, at least initially. The only information that will be hard-wired into the system is the idea that there are concepts and relations between concepts; all other information can be modified as the system changes its perspective. The reluctance to build in special-purpose mechanisms to handle these might seem surprising, given that the representation is admittedly not expected to meet the ideal requirements. Avoiding special-purpose mechanisms is extremely important, though, because it will allow the system's knowledge to be sensitive to its perspective. This also allows almost anything to be learned by the system (although this won't be taken advantage of here). Since almost all of the system's knowledge is represented using the framework, it can all potentially

[7]This is an oversimplification; there will actually be several kinds of links marking relations such as subconcept, subset, and subpart that will have inheritance rules. See section 3.6.1 for more details.

be learned by the system; any knowledge built into the framework would have to be "innate", i.e. present in the system from the beginning. This doesn't mean that that the system has to start off not knowing anything, of course; it just means that whatever "innate" knowledge the system has is encoded by the representation, not built into it.

The prospect of adding machinery as it seems necessary raises the specter of the representational framework becoming completely ad hoc, particularly since it is already designed around the specific needs of the generator. This is important in both the evaluation of the completed system and the process of building it. Assessing the generality of a system is often problematic when there's no clear underlying theory to judge it against. It's possible, of course, to just try adapting it to a new problem or domain. But this may require a lot of work, perhaps as much as building the original system, and if it doesn't work, is the problem with the system or the adaptation? If it does work, did the adaptation really differ enough from the original? Nevertheless, a combination of inspection and appropriate testing of the system can at least detect obviously ad hoc features; this problem is discussed further in Section 4.3. More difficult still is the problem of preventing ad hoc features from creeping in as the representational framework is modified. The only apparent remedy for this, aside from caution and good intentions, is that any change to the representational framework will be applied consistently throughout the system. Thus, for example, if a special mechanism is added to handle figure/ground distinctions because it's needed to properly annotate a particular linguistic option, it will be used wherever figure/ground distinctions occur. This will keep the mechanism a general one, rather than one used just for a particular case. Sticking to this principle while building the system will help ensure that the representational framework doesn't become a collection of ad hoc machinery.

The representational framework consists of three elements: a semantic network representing the system's world knowledge, perspective weights on the elements of the network representing the system's current perspective, and inference rules over the network that the system uses to reason about the world. The generator uses the semantic network, filtered by the perspective, as a map of its conceptual space that indicates differences in meaning between linguistic options and the planner's requests. The inference rules are used to help locate requests and options on the map. Thus the representational framework lets the generator compare an option with the corresponding request and determine how their meanings are related (in the current perspective), which is precisely what the generator needs to know.[8]

[8]The entire framework is also used by the communicative planner to support constructing plans and evaluating the consequences of particular actions. This use of the representation, however, is less central to the focus of this work, since it's really a matter of general issues of reasoning and representation. This will limit the power of the planner, of course, but the planner has already been limited in order to concentrate on the internal workings of the generator (see Section 2.4). This further limitation just means that it won't work as well as it should, not that it will work differently.

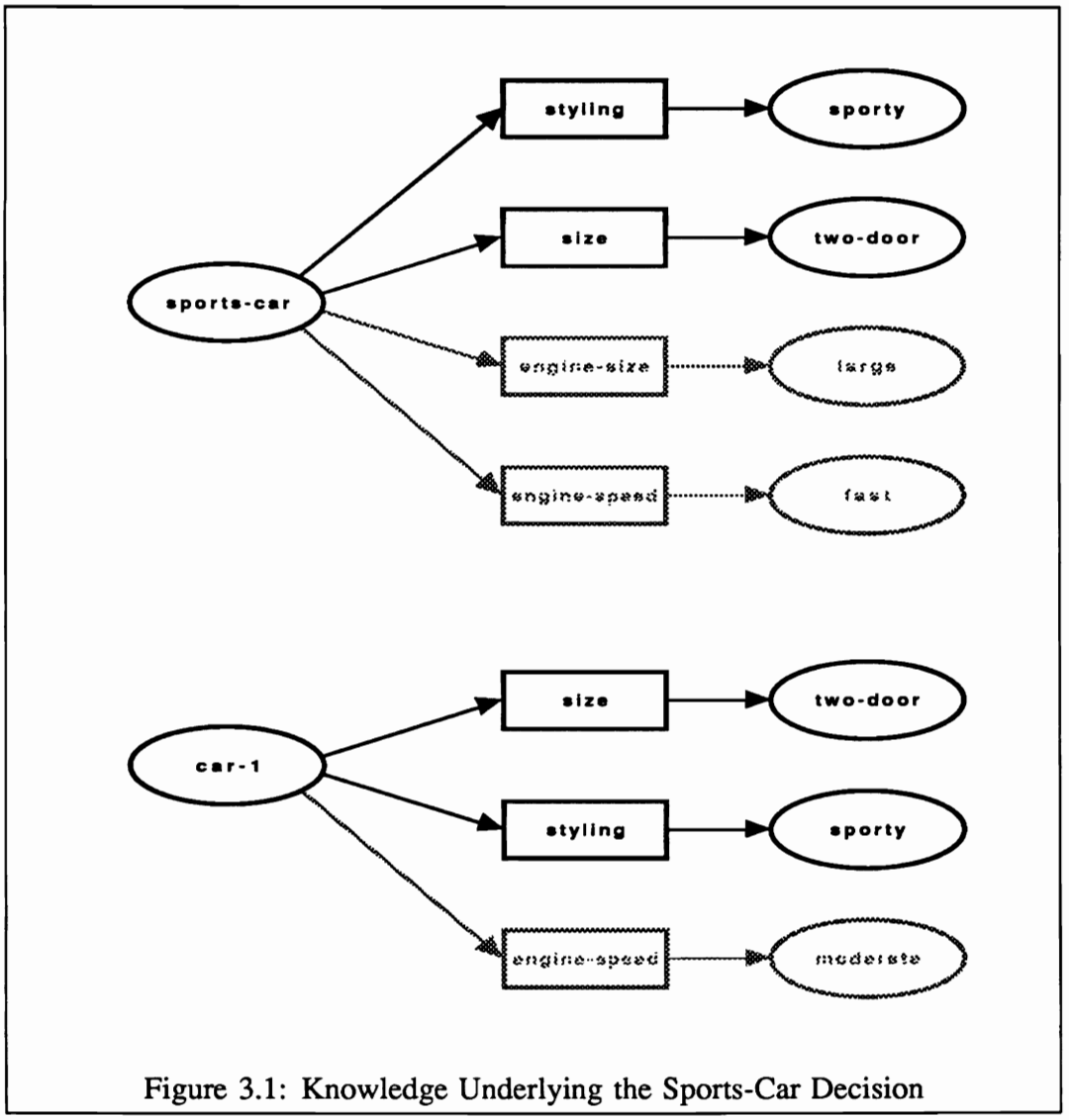


Figure 3.1: Knowledge Underlying the Sports-Car Decision

For example, suppose the generator is trying to decide whether a particular car **car-1** can be described as a "sports car". Figure 3.1 shows (some of) the knowledge used to figure this out; the shaded nodes and links are not in the current perspective. The system will use general principles of inference to decide that **car-1** does in fact fit in the category "sports car"; more precisely, it will deduce that there is an **isa** relation between **car-1** and **sports-car**. The knowledge about engine size and speed would block this inference, of course, but it's not in perspective, and is therefore not noticed. If the concepts of engine size or speed should come up either in the conversation with the user or in the system's reasoning (which is likely to happen if there is a continued discussion of cars) then these concepts will be brought into perspective. Once this happens, the system will no longer consider **car-1** to be a **sports-car**; instead it would have to find some other way to describe it. Thus the perspective shift leads the generator to produce a different utterance; this is precisely what the generator needs from the representational framework.

The various parts of the representation, in more detail, are organized as follows:

Semantic Network The basis of the representational framework proposed for the generator is a semantic network, in the original sense of representing various relations between concepts [Quillian 67], rather than providing definitions of terms as in KL-ONE or KRYPTON [Moser 83,Brachman 83]. In the traditional fashion, the network will be contain nodes representing concepts and individuals[9] and links representing relations between them. The links can connect an arbitrary number of nodes, although presumably most will connect two nodes, as is the case in most semantic networks. In addition, each node and link has a label; these labels are used to identify two nodes or links as representing (instances of) the same concept. For convenience, the label will generally be an English word or phrase corresponding to the (intended) meaning of the node or link, but the labels have no meaning to the system itself; the system can simply tell whether two labels are the same.[10] The links will also have weights associated with each node they connect; these weights are used in reasoning to control how strongly the link can be followed from the node. For example, the reasoning "if x is a fire engine, then it's probably red" is more likely and more reliable than the reasoning "if x is red, then it's probably (or possibly) a fire engine", both because it's more likely to be true and because it's more likely to be useful.

[9]The distinction between concept and individuals, and how to represent it, is discussed in Section 3.4.3.

[10]Actually, it would be better to use the labels simply as conveniences for the system designer that are invisible to the system itself. Unfortunately, they are needed for technical reasons; see Section 3.4.4. This is the first instance we've encountered of being forced to add machinery to the representational framework, since we could avoid the need for labels if we didn't need to have property inheritance.

Perspective Weights Each node and link in the network has a weight (in addition to the permanent weights on the links) representing its visibility in the current perspective. Nodes and links with higher weights are noticed sooner when the system is scanning through its knowledge to either follow chains of reasoning or look up information. Thus, for example, if the system is looking for properties to describe a fire engine, it will come up with "red" long before it comes up with "75 feet long". Sufficiently low perspective weights can ensure that a concept or relation will never be noticed (unless the perspective changes); thus the fire engine would never be described as "having Delco spark plugs" in most perspectives.

Variable weights, of course, raise the question of how they are varied. That is one of the issues to be explored in building the generator, but part of the answer seems clear already. Any time a concept or relation is accessed, that is, reasoned with or noticed by the system, its perspective weight must be increased (by how much is unclear). Also, nodes and links can have perspective activations associated with them, so that when they are accessed they can automatically alter the perspective of other nodes and links. This is clearly not enough though: the massive shifts in perspective that accompany a change in domain need to be handled somehow. Also, there must be some way that things can fall out of perspective if they're not accessed for a while; perhaps the perspective weights should gradually decay if the node or link is not accessed.

Inference Rules The semantic network only represents the static knowledge of the system. In order to reason about what it knows, the system must have some sort of inference mechanism; this is what the inference rules are for. Inference rules can be associated with particular nodes or links that they reason about, in which case they are only used when the nodes or links are accessed, or they can be generally available. Inference rules are subject to perspective because they are available only if the node they are attached to is accessible. In addition, it may prove necessary to put perspective weights on the general rules; this will not be done initially, though.[11] Furthermore, inference rules can have simplified versions that are activated when the details involve concepts that aren't in the current perspective. This allows for certain kinds of default reasoning; the simpler but less accurate version of the rule is used unless a concept involved in an exception is active.

The rules consist of an antecedent and a consequent, each of which is a small piece of semantic network. The rules operate by matching the antecedent against the system's knowledge base; if a match is found, the consequent is then added to the system. The consequent may already be there, of course, in which case the rule's effect is merely to increase its

[11]One alternative might be to not allow general rules but instead attach them to appropriate nodes or links that are almost always in perspective. The rule for property inheritance, for example, might be attached to the "subconcept" node.

perspective weights. The rule may also have nodes or links with variable labels; these are treated as universally quantified, and thus match anything. Variables in the consequent must also be in the antecedent, so there will never be a problem of trying to add variable-labeled nodes or links to the knowledge base. Arranging to use these rules in an efficient and useful fashion is a difficult problem which, fortunately, doesn't need to be solved here. The main rule that is needed for the generator is one allowing property inheritance, which is constrained to following "isa" links. Rules that are attached to particular nodes or links will, of course, be used only when those nodes or links are accessed. If the reasoning becomes too unwieldy, a simple time-limited breadth-first search can be used.

3.4 The Semantic Network

The semantic network is the base that the whole representational system is built upon. It records the system's beliefs about the world, indicating both the conceptual categories that the system uses to describe the world and the particular information about the world that the system believes to be true. With the perspective fixed, it resembles the usual kinds of semantic networks such as KL-ONE or KRYPTON [Moser 83,Brachman 83,Sondheimer 86].

There is a significant difference, though, between the semantic network here and KL-ONE or KRYPTON with regard to the intended interpretation of the network. In both KL-ONE and KRYPTON, the network is merely "terminological", that is, the nodes and links in the network merely serve to *define* concepts in terms of an fixed set of primitive concepts and relations [Moser 83,Brachman 83].[12] As discussed in Section 3.2, though, this approach will not work for an intelligent system. Instead, the structures in the network should be interpreted as making assertions about the system's beliefs about the world. The presence of a node for a concept means that the system believes that such a concept is a meaningful and useful way of categorizing things in the world; the presence of an individual node means that the system believes that such an object exists in the world.[13]

3.4.1 Nodes and Links

In knowledge representation systems such as KLONE and KRYPTON, the nodes and links represent concepts, individuals, and relations (or properties) in the world. Thus a "corporate stock" node represents the concept of corporate stock, and a "Mitch Marcus" node represents an actual person named Mitch Marcus. Here, in contrast, the nodes and links are the loci of the system's knowledge and belief

[12]Other systems may have different intended interpretations, of course, but it's often not clear what the intention is. KL-ONE and KRYPTON are noteworthy for being explicit about this. Also, KRYPTON's framework is actually defined in terms of these definitional relations, although these correspond in straightforward ways to elements of a network representation.

[13]See Section 3.4.3 for a discussion of individual vs. generic concepts.

about concepts and relations. Thus a "corporate stock" node or a "Mitch Marcus" node would simply be hooks where everything the system knows about corporate stock or Mitch Marcus is stored. In a sense, it's a misnomer to call the semantic network (et. al.) a "representation" framework, because it doesn't inherently *represent* anything. Rather, the network actually *embodies* the system's knowledge and belief. Thus the system's knowledge about the "corporate stock" concept consists of the various links that connect it to the rest of the net (together with any inference rules that mention it), and the presence of the "corporate stock" node in the network means that the system has a concept of "corporate stock" available for reasoning. This distinction isn't actually important for the generator's workings, but it is necessary to make sense of the notion of perspective in general and the handling of contradictions in particular.

The nodes and links, then, provide the basic vocabulary for the system's knowledge and belief about the world. Nodes are discrete, atomic entities. Links, on the other hand, connect nodes together. In general, links connect two nodes, but they can connect any number of nodes as appropriate. For example, a **between** link could connect three nodes together. As this suggests, there must be a way to distinguish which node is which for any given link. That is, of the three nodes connected by a **between** link, the system must be able to distinguish which node is between which other two nodes. This need arises even with two-node links; a **greater-than** link is useless without an indication of which node is greater than which.[14] In practice, this becomes important when comparing two links; the system needs to know which nodes correspond to which. This will be handled by numbering the nodes connected by a link from 1 to n. These numbers will *only* be used to pair nodes when comparing links; they are not intended to have any inherent meaning, nor will they be directly accessible to the system. There will be no particular meaning to being the "first" or "second" node for a link. Thus, for example, the "first" node of a **greater-than** link might be the larger or the smaller item; what matters is just that all **greater-than** links use the same ordering. Furthermore, **greater-than** and **smaller-than** links might both make the "first" node larger, and **larger-than** links might make the "second" node larger without any problem.

Actually, links can connect other links as well as nodes. These links provide the system with a way to reason about and generalize over relations. For example, there could be a **same-relation** link between two **greater-than** links, a **similar** link between a **greater-than** link and a **larger-than** link, or an **opposite** link between a **larger-than** link and a **smaller-than** link. Links can also connect a node to a link; in Figure 3.1, for example, there could be a **describes-appearance** link between the **car-1** node and the **styling** link. In fact, there can be one or more links between any link and the nodes (or links) it connects that indicate the role(s) the node (or link) plays in the relation indicated by the (main) link. This has the potential for infinite regress, of course, but in practice this won't be

[14]Note that in systems like KL-ONE and KRYPTON, the analogue of links (roles) are attached to particular concepts and this implicitly distinguishes between the two linked concepts.

a problem. These links are only potentially there; in general the system won't be interested in pursuing the regress very far, so they won't ever be created.[15] These "role" links could be used to distinguish the various nodes connected by a link, eliminating the need for numbering them. This would require, though, that there be a full set of role links associated with every non-role link. (The two ends of a role link can be easily distinguished, since one is a node and the other is a link.) This would perhaps be reasonable in a full intelligent system, which would necessarily have a large, broad, detailed network. In the present system, though, the network will be much simpler, and the numbering will be used (at least initially) for simplicity and efficiency.

Allowing links to connect both nodes and links, though, seems to weaken the distinction between nodes and links. And since links can connect a variable number of nodes or links, then perhaps nodes should simply be considered links of "arity" zero, i.e. that don't connect anything. The immediate objection to this suggestion is that nodes represent general concepts and links represent specific connections between concepts, but this is really a question of individual vs. generic concepts (as discussed below). In fact, there will be nodes for specific objects (e.g. the **Mitch-Marcus** node) and generic links that embody a relation abstracted away from any particular concepts it relates. There is a real question, in fact, of whether it is useful or meaningful to have all three of a concept **red**, a single-node link for the property **is-red**, and the concept of a **red-object**. Resolving this question, however, is not crucial for the workings of the generator, so the issue will not be tackled here. Nodes and links will simply be considered different kinds of objects (whenever it matters), and the nature and existence of zero-node links will be postponed to some later project.

3.4.2 Link Weights

Each link has a weight (between 0 and 1) associated with each node (or link) that it connects. These weights are intended to represent the directionality of connections between concepts. For example, the fact that fire engines are red is more about fire engines than about the color red. So the **color** link between **fire-engine** and **red** would have a much higher value for **fire-engine** than for **red**.

The link weights help control the running of inference rules; rules depending on links with high weights are more likely to be run, and the conclusions of the rules will be given more attention. This is in contrast to the perspective weights, which control whether the system even sees a link (or node) at all. The details of how this will work are discussed in Section 3.6 (although they are far from clear).

The link weights are not really motivated by any need of the generator. The inferences needed by the generator will usually be driven by the need to compare two concepts, and thus will be fairly well focused already. It seemed unreasonable,

[15] See Page 69 for a discussion of how to deal with a case where the infinite regress does cause problems.

though, not to provide some way to distinguish between properties or relations that are very central to an object (e.g. a fire engine being red, or an elephant having a trunk) and those that are incidental (e.g. red being the color of a particular house, or Knuth's **The Art of Computer Programming** being the first book I read as a child.[16]). A full intelligent system, of course, would have to make such distinctions. Still, since the generator doesn't really need them, the link weights will be eliminated if they prove to be difficult to implement and use reasonably.

3.4.3 Individuals vs. Generic Concepts

The discussion so far has glossed over the distinction between individual objects and generic concepts. The discussion of Figure 3.1, for example, ignored the distinction between **car-1**, a particular object, and **sports-car**, a generic concept. This distinction is an important one, since it affects what kinds of reasoning and belief about the concept are possible, and it affects how the concepts are used. The distinction has been made explicit in many (if not most) knowledge representation schemes. In KL-ONE, for example, there are two different kinds of nodes for generic and individual concepts [Moser 83]. In KRYPTON, the TBox (which is the semantic network-like part of KRYPTON) contains only generic concepts; individuals appear only within the ABox [Brachman 83].

It would be possible to build the notion of individuals into the system. This would require adding a second type of node (and link) that would be used for individual objects and relations. Then the **car-1** node in Figure 3.1 would actually be an individual node, and most of the links would be individual links. There would presumably have to be some adjustment of other parts of the system to accommodate this change. Link weights might work differently with individuals; inference rule antecedents might indicate whether they match only individual nodes, only generic nodes, or both; and so on. All of this would complicate the representational framework even further.

Instead, the generic/individual distinction will be represented within the network. There will be an **individual** node that will be connected by an **instance** link to each node that is an individual object.[17] Any inferences which depend on whether a node is an individual or a generic node can simply check for this link to find out.

The basis for this approach is the idea that the generic/individual distinction, while real, is part of the perspective. That is, whether a particular concept is generic or individual can depend on the system's current perspective on that concept. For example, **common-stock** certainly seems like a generic concept; it can have subconcepts (e.g. **IBM-common-stock** or **common-stock-that-pays-dividends**), and there's no particular object in the world that can be pointed to as *the* common stock. On the other hand, statements such as "there are two kinds of

[16]Not true.

[17]Individual links will be instances of either this node or of a corresponding generic **individual-link** link. The question of whether there should be **instance** links between nodes and links, and what to make of such links, is as yet unexplored.

stock: common and preferred" or "common stock was invented in the late Middle Ages " seem perfectly reasonable. In these statements, the concept of common stock is being treated either as a discrete entity within a set or as an object that has been created. Fortunately, the question of how to properly understand the generic/individual distinction need not be settled here. The distinction will simply be treated as part of the perspective unless it proves to cause too many complications or inefficiencies, in which case it will be built into the framework as a fixed property of nodes and links.

3.4.4 Labels

Diagrams of semantic networks usually show the nodes and links with labels like "**car**" or "**isa**", and the diagrams here are no exception. This raises the question of what these labels actually mean. There are basically four possibilities:

1. The labels are just conveniences for the reader and don't exist in the system at all.

2. The labels are purely atomic entities within the system; they have no meaning beyond being attached to particular nodes or links in the network.

3. The labels are built from a base set of uninterpreted atomic labels, but they can be combined into various composite structures.

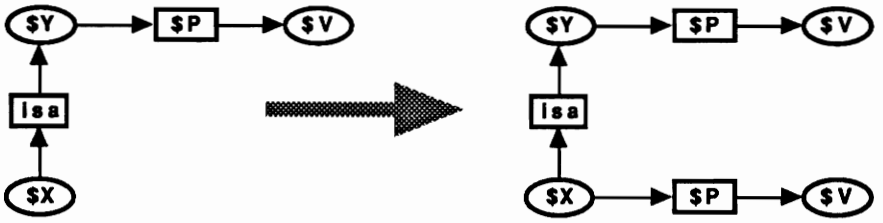
4. The labels carry meaning that the system can understand and use.

The last possibility, of course, just pushes the problem of how the labels come to have meaning off to another part of the system. In fact, this would imply that the semantic network isn't really serving as a representation formalism at all; instead whatever was providing the meanings for the labels would be the real repository of the system's beliefs. So this approach has to be rejected. The third possibility must be rejected for the same reason: there must be some other part of the system that provides an interpretation for the composite labels in terms of the atomic labels. Without such an interpretation, there would be no real connection between the composite labels and the atomic labels they contain, and the "composite" labels would effectively be atomic.

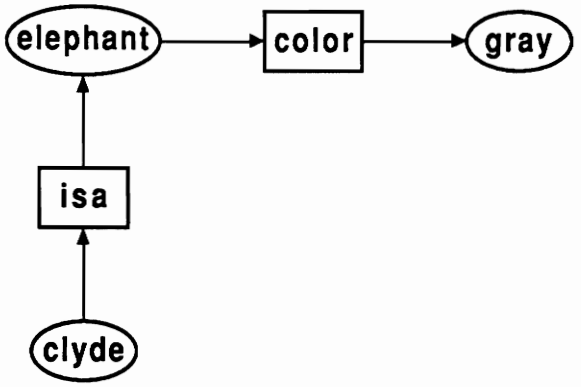
The ideal approach would be the first one: the labels are simply for the convenience of the reader (and designer), but have no role in the actual system. Only reasoning based entirely on the structure of the network would be legitimate. Any description, explanation, or justification of the system's behavior that relied on particular labels would be immediately seen as an instance of reading meaning into the network. Thus the kind of problems pointed out by McDermott [McDermott 81] could be avoided (or at least recognized) easily.

Unfortunately, this approach puts severe limitations on the inference rules. In particular, it is impossible to write an inference rule for property inheritance. Such a rule should look something like Figure 3.2a. But consider trying to apply it to

a) Property Inheritance Rule



b) Target for Inheritance Rule



c) The Problem: Infinite Regress of **isa** links

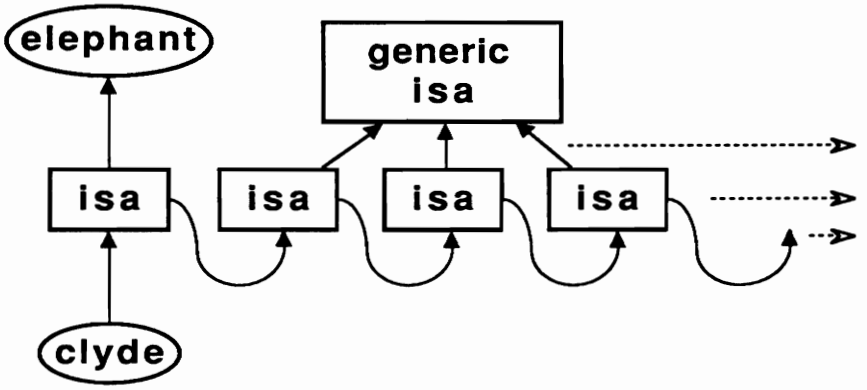


Figure 3.2: Inheritance Without Labels

Figure 3.2b: the problems is how to recognize that the **isa** link between **clyde** and **elephant** is in fact an **isa** link. The only indication of this is the presence of an **isa** between the link and the generic **isa** link. But this just postpones the problem; now the new link must be recognized as an **isa** link. And thus we end up with an infinite regression of **isa** links which must be followed to its (non-existent) end before the rule can be applied. The only way out of this regress is to make the labels visible within the system, so that the **isa** links can be immediately identified by their labels.

Thus the second approach is the one that will be adopted here: the labels will be visible to the system, but will have no structure or meaning to the system. The only thing the system can do with labels is compare them to see if two nodes (or links) have the same label. This makes the property inheritance rule tractable. The antecedent **isa** link can now be immediately matched with the link between **clyde** and **elephant** because it has the same label, and the rule can be run.[18] But the labels still don't have any meaning to the system; the meaning of a node or link depends entirely on how it fits in with the rest of the network and the inference rules.

3.5 The Perspective Weights

Every node and link in the network will have a weight between 0 and 1 representing its prominence in the system's current perspective. The higher the perspective weight on a node or link, the more likely the system is to notice it. A node with weight of 1 would be immediately noticed by the system; a node with a weight of 0 would never be noticed by the system (unless its weight changed). The collection of weights on all the nodes and links in the semantic network embodies the system's current perspective; that is, how the system currently conceives of all the objects and relations it knows about. for example, the difference in perspective that allows a particular house to be described as either a hovel or a cottage could be modeled by differing perspective weights on **instance** links to **hovel** and **cottage** nodes. More importantly, this perspective difference could result from different perspective weights on related nodes such as **size, location,** or **state-of-repair**. If the **location** and **size** nodes has a high perspective weight but the **state-of-repair** node has a low weight, then the system might infer that **cottage** appropriately describes the house, since the factor that might block the description is not noticed. If the weights were reversed, on the other hand, the opposite conclusion would be reached.

The perspective weights control the "noticing" of nodes and links by ordering the system's search through the network. The system searches the semantic

[18]Note that it's still possible to allow individual **isa** links to have other labels without undercutting inheritance. All that would be needed is an inference rule that says: any two nodes joined by a link that **isa** **isa** link also have an **isa** link between them. This amounts to following up the (infinite) chain of **isa** links far enough to identify it. This technique won't be needed in the current system, though; all the **isa** links will be appropriately labeled.

network in three ways: it can simply scan the network to see what elements are connected to a particular node or link (e.g. to construct a description, or to search for information to help with some other goal it is working on); it can compare nodes or links; and it can check for matches to inference rule antecedents (see Section 3.6). In each of these cases, the perspective weights determine which nodes and links the system examines and in what order. Thus when scanning the network the system looks first at the connections with the highest weights; connections with low weights will usually never be reached at all. When comparing nodes or links, the system will look first at the connections that have the highest weights; connections that are completely out of perspective will be ignored. Finally, when running an inference rule, the system will try potential antecedent matches with the highest perspective weights first.[19] In all three cases, nodes and links with high perspective weights are "noticed", i.e. used by the system, sooner than nodes with low weights.

There are a couple of difficulties with this model of perspective. The first is with the notion that the perspective weights order the search through the network. Since the system's perspective varies constantly, there is no way to pre-compute a list of (say) links connected to a node, ordered by perspective. Instead the system has to compute how perspective orders network elements on demand whenever the system needs to search the network. This means that the system has to actually scan through all the links, including the ones that aren't in perspective, in order to determine which ones to "notice". Ultimately, this paradox suggests that a massively parallel implementation of the network is needed; this is, of course, a natural approach for a large network anyway (given appropriate hardware). Even with a sequential implementation, though, it's important to remember that perspective is a central part of the representation design, not just some sort of feature of the implementation. So even if the implementation is forced to check nodes and links that are out of perspective, the system will still see the search as guided by the perspective; the low-level details of the search are not visible.

The other difficulty is with the notion of being "out of perspective". A node or link with a low perspective weight should not be noticed by the system, but how is this to be arranged? A node or link with zero weight can simply be ignored, but small non-zero weights are a problem. In some cases the normal ordering of search by the perspective weights will ensure that nodes with low weights will never be reached; in constructing a description, for example, the system will often find enough information from nodes and links with high weights. This cannot be guaranteed, however; in particular, a search for potential rule antecedents could easily find a match with a low perspective weight if there are no matching nodes currently in perspective. What is needed is some sort of cut-off value below which nodes and links are ignored just as if they had zero weights. Unfortunately, the choice of a cut-off value is completely arbitrary, and the consequences of a particular choice are unclear. Ultimately, there should probably be a variable cut-

[19]The perspective also affects the order in which rules are tried and the content of the rules; see Sections 3.6.3 and 3.6.5.

off range depending on how hard the system is looking (a notion which would also need to be made more precise, but see Section 3.6.5). For the current project, a fixed cut-off will be used, and the effect of setting it at various values will be explored.

The model of perspective described here is similar to the one in the ROMPER system [McCoy 85], in which a perspective consists of salience values for object attributes in a KL-ONE-like concept network. "It is these salience values which dictate which attributes are highlighted and which are suppressed."[McCoy 85, p. 68] Thus the salience values function much like the perspective weights. There are, however, a number of differences:

1. In ROMPER, there is one (current) set of salience values for the entire network; two concepts cannot have different salience values for the same attribute. In contrast, two nodes can have different weights for the same relation (i.e. for links that are instances of the same generic link); there is no particular connection between them.

2. ROMPER allows a small fixed set of precomputed perspectives, in contrast with the unbounded set of dynamically computed perspectives possible here.

3. ROMPER only uses perspective when judging the relative similarity of object classes; perspective here is integrated with every operation of the knowledge representation.

4. ROMPER's perspectives control how much weight is given to each attribute when making comparisons. The perspective weights here are used only to control whether the system *notices* a node or link; once noticed, a node or link is given equal importance regardless of its perspective weight.[20]

3.5.1 Setting the Perspective Weights

The perspective weights must initially be set to model the system's current perspective and subsequently be modified as the system's perspective shifts. Determining initial values for the perspective weights will be tedious but poses no major difficulties. Since the system will be starting off with no particular idea of what it is expected to do, the perspective should include high weights on fairly general concepts and on concepts related to helping the user (assuming the semantic network is sophisticated enough to include such concepts, which is unlikely in the present project). Medium weights should be put on basic concepts in whatever domain(s) the system knows about, and low weights should be put on everything else. If there are a small number of particular domains or problems the system is expected to be used for, then concepts relevant to them can have their perspective weights increased. Experience in building and testing the system will undoubtedly

[20]This is a slight overstatement, since reasoning is directed towards nodes and links with higher weights. Any operation such as comparison that depends on the presence or absence of a node or link, though, gives equal importance to all in-perspective nodes and links.

suggest ways to improve this general outline. The basic idea, though, is that the system's initial perspective should focus on its general knowledge without too much detail, on helping the user, and possibly on the likely domain(s) it will be working in.

A more complicated issue is how the weights are modified to reflect shifts in the system's perspective. The perspective shifts can't be worked out in advance by experimentation like the initial weights since they depend on the particular circumstances and situations the system finds itself in. The knowledge representation must be able to detect when the system's perspective should shift and respond by modifying the weights appropriately. Thus the weights will dynamically model the system's perspective as it develops and changes.

One trigger for perspective shift is external effects such as perception. For example, seeing a red object will bring the color red into perspective. Similarly, touching something that's very hot will bring the concept of heat (and pain as well) into perspective. Even a computer system without any perception of this sort can still receive input from the user, which will have similar effects on its perspective. Any concepts that the user mentions would of course be "noticed" by the system and should therefore be brought into perspective. This is precisely how the system can focus on whatever questions or problems the user is interested in. Since the current project is primarily concerned with the generator, though, there won't be any opportunity for this kind of perspective shift. A full intelligent system, though, would need to handle it.

Perspective shift more often arises from the workings of the system itself. As the system reasons with and explores its body of knowledge, it will tend to bring into perspective concepts related to the ones it is using. In the most extreme case, using a concept in a new domain can bring the whole body of knowledge the system has about that domain into perspective. The shift can be much less dramatic, however. Thinking about a rainbow, for example, might bring into perspective the various colors of the spectrum; it might also suggest related concepts such as prisms, or rain, or even romantic feelings. Or if the system were thinking about a car and noticed that the door was long, it might bring into perspective concepts such as the distinction between two-door and four-door cars, the ease of getting into the back seat of a car, and the amount of leg room in the back seat. These are cases of shifting a whole body of concepts and knowledge into perspective, but on a much smaller scale than the usual notion of domain, which involves a reasonably complete and self-contained body of knowledge. Perspective shift can involve even smaller effects than this, though, down to altering weights on only one or two nodes or links. Thinking about "today", for example, brings into perspective a time scale of a few days centered on the current day; this would involve only the node for the time scale and the link between it and the **today** node. Thus the normal use of the representation by the system must be able to trigger perspective shift on any scale from an entire domain down to just a few nodes and links.

This kind of perspective shift will be handled by associating with each node and link a set of perspective weight adjustments to be made when the node or link

is used by the system. These adjustments indicate which other nodes and links should be brought into perspective, and how much they should be shifted, i.e. what their new perspective weights should be. It might also make sense to have the new weights depend on the old weights and the weight of the node or link inducing the adjustment, but it's not clear what the dependence should be. The new weights will therefore just be set to the values indicated in the adjustments, except that if the old weight is higher than the adjustment value it won't be changed, because that means the node or link was already in perspective.

Whenever the system uses a node or link, it will carry out any associated adjustments, thus effecting the appropriate perspective shift. "Using" a node or link means one of:

- Using it as part of a triggering match of a rule antecedent.
- Asserting it as part of a rule consequent.
- Comparing it to some other node or link.
- Checking its connections while scanning through the network.

Any of these events will trigger the associated perspective weight adjustments. In addition, using a node or link (in this sense) will cause the node or link itself to be brought into perspective if it isn't already, since it has certainly been noticed by the system. Similarly, using a link will bring the nodes or links it connects into perspective, since thinking about a relation naturally brings the related concepts (or individuals) to mind. It might seem reasonable to extend this to activating every network element connected to a used node or link. This would eliminate the entire effect of perspective, though, since it would mean that whenever the system looked at a node or link, it would see everything connected to it regardless of the current perspective. So this extension will not be made. Using a node or link will bring into perspective only the node or link itself, the nodes or links it connects (for links), and the nodes and links indicated by the adjustments associated with it.

Managing the system of perspective weights poses some difficulties. Any use of the knowledge representation can shift many perspective weights, so there is a danger of spending more time adjusting weights than actually using the representation. More critically, bringing too many concepts into perspective can overwhelm the system. There are really two problems here: how to constrain bringing things into perspective, and how to move things back out of perspective so that everything doesn't end up with a high weight. The problem of shifting concepts out of perspective is very important, but I don't currently have any well-developed insight into its solution (hence its conspicuous absence from the discussion above). There are some obvious possibilities, such as having weights decay over time, or having downward as well as upward adjustments associated with nodes and links, but there are problems with these approaches. Fortunately, the generator will not depend on a solution to this problem, as it will operate over

relatively short time spans in which concepts would rarely fall out of perspective anyway.

The problem of constraining bringing concepts into perspective is important, though, because the generator depends on the perspective to guide its reasoning and search and to evaluate particular linguistic options. The solution is simple, though: perspective shift will be induced only when the system uses a node or link with a low perspective weight. Since the majority of the nodes the system uses will have high weights, perspective shift will occur only a fraction of the time, and the set of nodes and links currently in perspective will expand slowly. This is a reasonable constraint even independent of these efficiency concerns. If a node or link has a high perspective weight, that means the system is already aware of it; actually using it in reasoning or thinking won't bring anything further to mind. Only when the system uses a concept it was only slightly aware of will the system also bring into perspective other related concepts. Thus nodes and links with high perspective weights will not induce perspective shifts.[21]

This constraint presumes a well-defined distinction between high and low weights; it's not clear how that distinction should be drawn. It must be different from the one used to determine whether concepts are completely out of perspective; nodes and links with weights below that cut-off point will never be noticed, so they can't induce any perspective shift. There are still many questions; should it be a single cut-off point? Should it depend on precisely how the node or link is being used? Should different kinds of perspective shift be handled differently? Should there be different sets of adjustments depending on the node or link's weight? Since I don't have answers for these questions, the simplest approach will be taken: a single fixed value will distinguish "high" weights from "low" ones. As with the cut-off for being in or out of perspective, the effects of choosing particular values will be explored as the generator is implemented.

3.6 Reasoning and the Inference Rules

In addition to merely having a set of current beliefs, an intelligent agent must be able to reason with those beliefs to figure out new information that it needs to accomplish its current goals. The inference rules provide the mechanism that supports this reasoning. The inference rules can vary from very general ones such as the property inheritance rules to very specific ones such as "if the jewelry box is on the floor, then the cat must have jumped on the dresser". The rules provide mappings from existing information (i.e. a set of nodes and links currently in the network) to new information (i.e. new nodes and links). They can be used in both backward-chaining and forward-chaining fashion, i.e. in attempting to deduce a particular piece of information or to see what follows from the system's current beliefs. This corresponds respectively to the system trying to determine

[21] One exception: using a link with a high weight will still bring the nodes it connects into perspective, because they are really part of the relation the link represents. For activating the node or link itself the question is moot: if it has a high weight, then it's already fully in perspective.

if something is true or determine the details of some information, e.g. to find the particular color of some object, and to considering the consequences of something it believes is true.

Instead of having explicit inference rules, there could simply be an **acceptable-inference-pattern** node for the concept of an inference that the system is willing to make. Each individual inference "rule" would then be an instance of this node. This would avoid the need for special representational machinery for the rules. It would also make the knowledge expressed in the inference rules themselves available to the system for introspection or reasoning. On the other hand, knowledge of acceptable inferences is *used* differently than other kinds of knowledge, so it seems reasonable to represent it differently. It would also be awkward and inefficient to represent inference rules within the semantic network; some mechanism for picking out a subnetwork with "meta-nodes" and "meta-links" would be needed. Furthermore, there would still have to be special machinery to recognize and use the inference rules. Thus we might as well expand that machinery to include a special, efficient, representation for the rules. This does mean that the system will not have direct access to the information in the rules, a consequence with significant psychological and cognitive implications. None of its implications, though, seem relevant to the generator, so the issue will simply be ignored.

The rules will consist of an antecedent and a consequent, each of which are small networks (perhaps as small as a single node or link). The rule is run by trying to match the antecedent against the main network; a match consists of a subset of the main network that has the same pattern of nodes and links as in the rule antecedent with all the labels matching. If a match is found, then the consequent can be asserted in the network also. This assertion can be temporary, i.e. available only for immediate subsequent inference, or permanent, in which case the consequent is actually added to the network.[22] The antecedent and consequent may share nodes and links; this allows the rules to assert information about existing nodes. Any nodes or links in the consequent that are not in the antecedent are added to the network as new nodes or links. Rules can also contain nodes or links with variable labels. Variables in the antecedent can match any node or link (with the same number of connections); variables in the consequent (which must also occur in the antecedent) are filled in with their value from the antecedent. These variables allow rules to make generalizations; without them, rules could only apply to a fairly specific set of nodes and thus wouldn't be very useful.

There are a number of possible variations on the rule structure, such as allowing variability in the network configurations or allowing restrictions on the range of the variables, that a full intelligent agent might need to support its reasoning. The primary goal here, though, is to support the expected needs of the generator. What the generator will need most is the ability to locate the relative positions of

[22]Deciding whether to add the consequent to the network involves issues of memory and learning that will not be addressed here; all the reasoning done for the generator will be temporary.

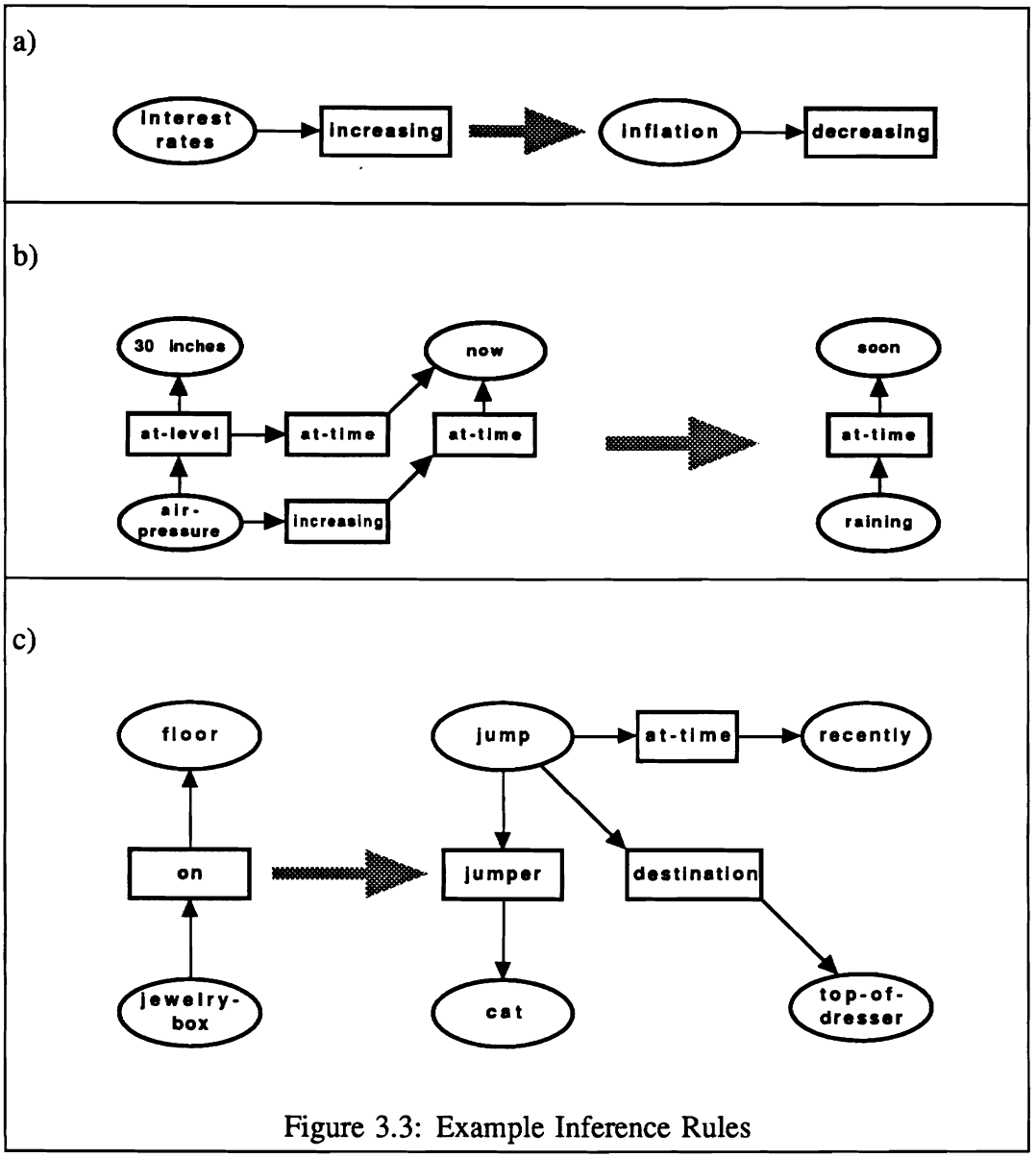


Figure 3.3: Example Inference Rules

various concepts within the system's belief space. This requires mainly rules for inheritance and simple rules to determine whether particular properties and relations hold over particular objects and concepts, i.e. rules that build up complete descriptions that can then be compared. The current formalism appears adequate to meet this need (with one exception, discussed in Section 3.6.2). Thus variations on and extensions of the rule structure should not be necessary.

Some examples of rules are given in Figure 3.3. Rule a represents the belief that rising interest rates lead to lower inflation. Rule b represents the belief that if the air pressure is at 30 inches and rising, it will rain soon. Rule c shows how the rule about jewelry boxes and cats mentioned above could be represented. These rules are intended just as illustrations of the formalism, of course; without a broad background of nodes, links, and rules within which to interpret them, they're not

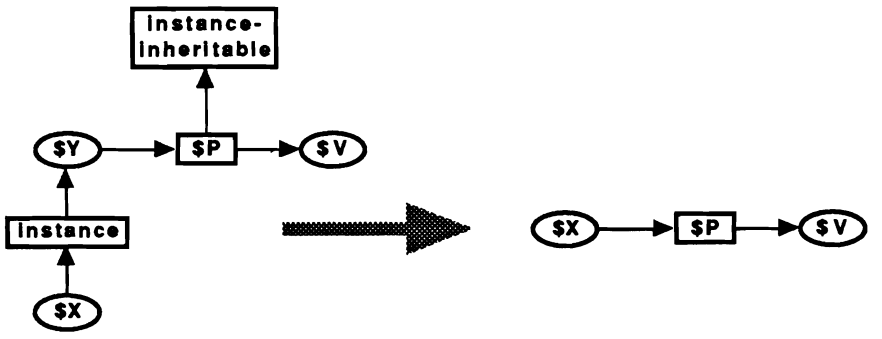


Figure 3.4: An Inheritance Rule for Instance Links

really very meaningful.

3.6.1 Inheritance Rules

One particularly important set of rules are the ones providing for property inheritance. These rules allow the system to apply general knowledge to more specific cases, and are thus crucial to the workings of the generator. There need to be several rules because there are actually several relations that license inheritance. Many of the examples so far have used **isa** links for simplicity, but in the actual system, **instance**, **subset**, **subconcept**, and **part** links will each license inheritance, subject to appropriate constraints.[23] The constraints limit the inheritance to properties and relations that can appropriately be inherited. A **color** link would reasonably be inherited across an **instance** link or a **subconcept** link, for example; an **invented-by** link would not. Figure 3.4 shows an inheritance rule for **instance** links. The **instance-inheritable** link enforces the constraints on the rule; the system must believe (or infer) that the link has the property of being inheritable across **instance** links in order for the rule to run.[24] In the actual system, these constraining links will be omitted, since the system will only be doing limited reasoning. If the system does use invalid inheritances, then constraints will be added.

The rule in Figure 3.4 is actually only one of several rules to handle inheritance across **instances**. It will only handle inheritance of links that connect two nodes; it won't handle inheritance of links that connect a different number of nodes

[23]The difference between the **subset** and **subconcept** links is that the former indicates an accidental relation, e.g. between **taxi** and **yellow-car**, while the latter indicates an inherent relation, e.g. between **PhD-dissertation** and **document**. If this distinction proves to be problematic, a **contain** link could be added that generalizes **subconcept** and **subpart**; this link would also require an inheritance rule.

[24]Inheritance on **instance** links also requires a way to inherit **instance-inheritable** links onto the particular links being inherited. This threatens to create an infinite regress similar to the one discussed in Section 3.4.4; fortunately, the solution only requires adding a rule saying that all **instance-inheritable** links are themselves **instance-inheritable**.

or inheritance of links connected to other links. The form of the rules forces there to be a different rule for each possible link configuration. In theory this would require an infinite number of rules (since links can connect arbitrarily many nodes), but in practice links are unlikely to ever connect more than a few nodes, so the inheritance rules can be safely limited to links of arity five or less, say. This multiplication of rules is a defect of the representational scheme; it is obviously missing a basic generalization. It could be eliminated by modifying the representation of the inference rules to allow for variable-arity links and elements in the antecedent that can match either a link or a node. At present, though, this complication will be avoided. Experience with the implementation, though, may indicate that other rules could take advantage of it; in that case it will be adopted, allowing for the combination of the inheritance rules into a single rule for each licensing relation.

3.6.2 Applying Descriptions

The rule formalism described so far is adequate for most of the rules the generator will need. It can handle property inheritance and simple inferences. Unfortunately, there is one other type of inference needed by the generator that the existing formalism cannot express; this is the inference that something that meets the description of some concept (i.e. that matches all the links connected to it) is an instance or subconcept of it. This is related to the process of "classification" in systems such as KLONE or KRYPTON [Lipkis 82,Schmolze 83], except that in classification the goal is to find the minimal concept that subsumes the new concept. Here the goal is just to determine whether a particular concept "subsumes" (i.e. describes) another concept. The representation needs to have this ability in order to match (the meaning of) linguistic expressions against the information the planner wants to express. The generator will need to know whether an expression describes a particular concept even if (as will generally be the case) there are no links already in the network indicating the relationship between them.

This kind of inference cannot be represented with the machinery developed so far, because it requires quantifying over network elements. The necessary rule is something like "if all the links connected to **$n1** have matching links connected to **$n2** then **$n2** is a subconcept of **$n1**" (with appropriate qualifications to distinguish between subconcepts and instances). The antecedent of a rule, though, can only be a fixed subnetwork; there's no way to match against a varying set of links. One solution, of course, would be to expand the rule formalism enough to express this. This would be a significant extension, though; it would require not only quantification over links but also conditionals within the antecedent. Adding this much power to the rules would make them much more difficult and inefficient to use, in return for allowing one small class of rules to be expressed.

Instead, the ability to apply descriptions will be built directly into the rule formalism. Any rule will be able to specify in its antecedent that one node must describe another node. The "classification" inference rule(s) will then be straightforward to write; the antecedent will simply contain **$n1**, **$n2**, and an

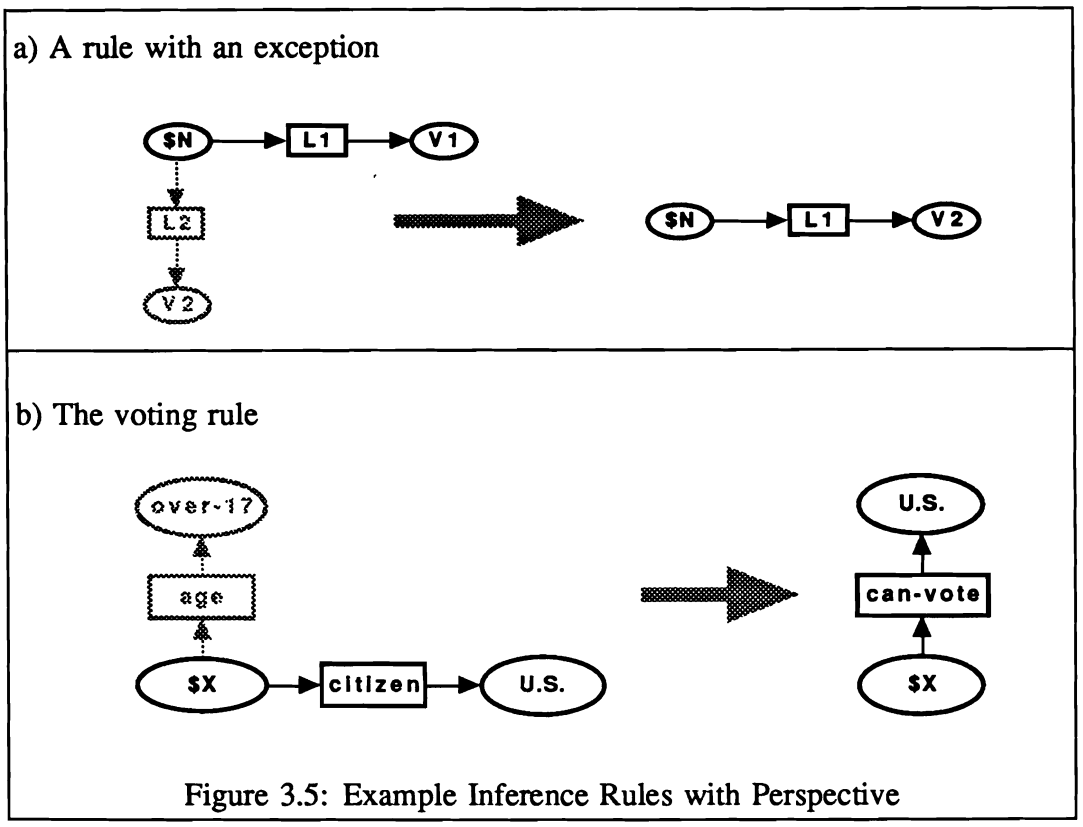


Figure 3.5: Example Inference Rules with Perspective

indication that one describes the other. Other rules can also use this feature, although there's little reason to do so. Any rule that specifies description can be replaced by one indicating an **instance** link between the two nodes. Thus the ability to apply descriptions gives the rule formalism just enough added expressive power without requiring that the additional mechanism be restricted to specific cases.

3.6.3 Rules, Perspective, and Default Rules

Reasoning can be affected by the current perspective in two ways. The first is fairly simple: each inference rule will itself have a perspective weight. These weights will be used the same way as the weights on the nodes and links: rules with very low weights will be completely out of perspective and not used, and rules with higher weights will be used sooner. The detailed interaction between rules' perspective weights and the use of the rules is discussed in Section 3.6.5.

A more subtle effect arises from the interaction of perspective and the rules' antecedents and consequents. Since a rule's antecedent and consequent are actually small semantic networks, the nodes and links they contain all have perspective weights. These weights are used and manipulated in the same manner as the perspective weights in the main network. In particular, nodes and links in the rules can be out of perspective, effectively changing the content of the rule. For example, the rule in Figure 3.5a would require only an **L1** link to a **V1** node in

the current perspective; if the **L2** link came into perspective, the rule would then require both links in order to fire.

This effect provides a way to encode general rules with exceptions without the exceptions overwhelming the general rule. Consider the well-known problem of whether Tweety can fly. This involves a general rule that all birds can fly with the exception that penguins (which are birds) can't fly plus the knowledge that Tweety is a bird. The difficulty is how to encode both the general rule and the exception and still be able to infer that Tweety can fly. If the general rule and the exception are encoded separately, the system can infer both that penguins can't fly and (since they are birds) that they can, a contradiction.[25] The alternative would be to encode as a rule that birds that are not penguins can fly. This, though, would make it impossible to infer that Tweety can fly, because there's no way to prove that Tweety isn't a penguin. This sort of dilemma has provided much of the motivation for non-monotonic reasoning systems such as default reasoning and circumscription [Reiter 80,McCarthy 80].

The Tweety dilemma would not actually be handled by a "default rule" in the current framework, since it is a case of property inheritance. Similar sorts of default reasoning can be represented using rules and perspective, though, as in Figure 3.5b. The rule here captures the general notion that every citizen of the U.S. is eligible to vote. Children are an exception, of course, but normally in a political discussion that exception would be irrelevant and the **age** link would be out of perspective. If the system *did* start talking about a child or children, the **age** link would be shifted into perspective. Thus the system can normally conclude that anyone can vote without worrying about their age; when children are being discussed, the system will notice that age is relevant and check it.

3.6.4 Contradictions

One apparently troublesome consequence of the representation's design is that differing perspective can lead the system to reach contradictory conclusions. Given the network in Figure 3.1, for example, the system can either conclude that **car-1** is or isn't an instance of **sports-car**, depending on the perspective. Perspective shift can also cause contradictory attributes to apply to the same object; for example, a grey sock might be "dark" in comparison to white socks but "light" in comparison to black socks, or a book might be "sophisticated" when considered as a children's book but "simple" when considered as adult reading matter. Even worse are properties such as "interesting", "entertaining", or "irritating", whose applicability can vary based on factors entirely independent of the object they are applied to or the situation in which it is being described.

As long as the contradictions are between information in different perspectives, though, there's no real problem. The whole point of perspective is to make visible just the information that is relevant and useful for the system's current needs. So

[25]Of course, contradictions are not a serious problem for the representation here (see Section 3.6.4). Still, it would be better to avoid them whenever possible, since resolving contradictions can take a lot of work and disrupt normal reasoning.

when the system is thinking about how cars look, it *should* consider **car-1** a **sports-car**, and when it's thinking about how cars drive, it should consider **car-1** not to be a **sports-car**. This doesn't cause any problems; the contradiction only comes up here because the shift in perspective has changed the system's concept of a **sports-car** (by changing the links attached to the **sports-car** node).

The real difficulty comes up when contradictory information shows up within the current perspective. This can happen if the system decides to record the result of an inference permanently in the net.[26] The system may then "remember" the information even after the perspective has changed in a way that makes the inference no longer valid. For example, the system could infer an **isa** link between **car-1** and **sports-car** while considering cars' appearances, and the link would remain even after a shift to considering how cars drive. In the new perspective, the system's beliefs imply that **car-1** isn't a **sports-car**, contradicting the **isa** link. The system doesn't know what to believe! This could be avoided by putting the **isa** link only in the first perspective, but there's no way for the system to do that. The only way the system could know precisely what perspective(s) to limit inference results to would be to test the inference in every possible perspective. Since the set of perspectives is potentially unbounded, this is not possible. Even if it were possible, subsequent changes elsewhere in the network could change what's in a particular perspective. So there is no way in general to prevent information added to the system from causing contradictions with other information that's not currently in perspective.

Why exactly is this a problem, and what can be done about it? In a system based on first-order logic (or most variants of it) contradictions are deadly, because any statement follows from a contradiction. Since the inference system here is not intended to be sound or complete (if there's even any relevant notion of soundness or completeness), that won't happen. As long as there are no rules with contradictory antecedents, nothing will be inferred directly from a contradiction. On the other hand, both contradictory statements will be available for inference, and the results of these inferences will presumably often be contradictory themselves. For example, the system might infer that **car-1** is fun to drive and is worth buying since it's a **sports-car**, and also that it isn't fun to drive and isn't worth buying since it isn't a **sports-car**. Over time, the system would acquire more and more contradictions and be unable to reliably or consistently believe anything.[27]

The real danger, then, is contradictions within a perspective that are used for inference over a period of time. Contradictions between perspectives, or within a perspective where at most one of the contradictory statements is noticed by the inference rules, are not a problem. These simply indicate that the system

[26]Of course, this could also result from faulty inference or wrong information given to the system by users (or programmers). But this would be handled simply by trying to locate and correct the mistake or the wrong information. The issue here is with contradictions that should be kept in the system; the problem is just to keep them in separate perspectives.

[27]Note that this isn't really a problem for the generator, since it won't be adding assertions to the network; but the theoretical problem must still be resolved if the framework is to have a meaningful interpretation.

has contradictory beliefs, which is not an inherently problematic state.[28] If the dangerous contradictions are the ones that drive inferences, though, then they are also the ones that the system is most likely to notice, since they involve links or nodes that the inference rules are explicitly looking for. So the ultimate solution to the problem (which will not be implemented in the current project) will involve dealing with contradictions when the system notices them. The system will have to examine how it came to hold the contradictory beliefs, what follows from each of them, and how well each is supported in the current perspective, and then either eliminate one of them from the current perspective or mark them as an unresolved contradiction and avoid using them in inferences. There are a lot of details to be resolved, of course (e.g. how broad a range of perspectives is the contradiction removed from, how is the reasoning leading to the contradiction traced, how does the system decide which belief to remove), but the general outline is clear: when the system notices it holds contradictory beliefs, it decides which belief it finds more compelling and settles on that one.

3.6.5 Using the Inference Rules

In order to use the inference rules, the system must decide when to run them and which ones to run. The simplest approach would be to always run any rule whose antecedent matches something in the network; then the system would always infer as much as possible. This process, though, would never terminate.[29] Actually, that's not so bad, given that the system is functioning in real time; the inference rules could simply go on inferring things while the rest of the system continued in parallel. The real problem is that it would infer things in the wrong order. The vast majority of inferences would be information the system had no need for or interest in, while inferences the system needed would often be put off for a long time, possibly forever. What is really needed is an inference strategy driven by the system's current needs and interests.

The inference strategy used here is designed around two principles: inference is driven from particular nodes and links in the network and is guided by the current perspective. Inference could proceed by collecting all the rules that meet some criterion for potential runnability (e.g. all the non-variable nodes in the antecedent have matches in the network) and running them. This would spread the reasoning out over the entire network, though; instead, the reasoning will focus on particular nodes and links that the system wants to explore further. This will help ensure that the inference will be directed towards what the system needs to know. The perspective will provide further direction, concentrating inference on concepts that are currently within the system's notice.

[28]Of course, only one of the contradictory beliefs can be *correct* (in the current perspective), but that's beside the point. The network is only representing the system's *beliefs*, not the actual state of the world.

[29]For example, the system would infer that $2 > 1$, $3 > 2$, and so on. Even without any mathematical knowledge, there would be many such infinite chains of nodes it would infer, e.g. Bill's son, Bill's son's son, etc.

Inference is invoked when the system wants to know either what follows from a particular node or link or whether a particular node or link can be inferred. The first case occurs when the system decides that some concept or relation may be connected to other information it would find useful; the inference rules will then be invoked in a forward-chaining fashion to explore the implications of the concept. This kind of reasoning is normally invoked from outside the inference system itself. Automatic invocation of inference may be useful in some circumstances, though, such as when a node or link is added to the network or shifted into perspective; the consequences of this idea will be explored as the system is implemented. The latter case of invoking inference occurs when the system wants information that isn't in the network (as filtered by the current perspective). The system could simply be looking for a particular node or link, or it could be trying to complete a partial subnetwork. The simplest (and probably most common) case of this is trying to find which nodes a link connects. For example, the system might want to know what color a particular car is; the question here is not whether there is a **color** link attached to the car's node, but rather what particular node the link connects the car's node to. When invoked this way, inference proceeds by backward-chaining, looking for rules whose consequent fills out the subnetwork the system is interested in.

However inference is invoked, the general procedure is the same. The system first collects all candidate rules and then uses perspective to select and run one of them. If the rule succeeds, new candidate rules are added to the set; this procedure is repeated until either there are no more candidate rules (not likely to ever happen) or enough rules have been tried (see below). The details of how to implement all this differ somewhat for forward- and backward-chaining inference. Of particular importance, though, is that candidate rules need not actually be usable; that would require trying all the rules first to see which ones ran successfully. Instead, candidate rules are intended to be ones that might be usable; non-candidate rules are ones that can be immediately (and simply) dismissed as useless.

For forward-chaining inference, a "candidate rule" is one whose antecedent contains nodes and/or links matching the piece of the network the system is reasoning forward from. For example, Rule c in Figure 3.3 would be a candidate when reasoning forward from the **jewelry-box** node. There needn't be a match for the entire antecedent; as long as the relevant part of the network is matched, the rule will be a candidate. Once all the candidates are collected, the rule with the highest combined weight will be chosen. This weight is formed by combining the perspective weights of the nodes and links in the rule's consequent, the weight of the rule itself, and the weight of the node or link the system is reasoning from. Precisely how to combine these weights is an open question; the simplest thing would be to add them all together, but that would probably undervalue the weight of the rule. The link weights might also be involved here; the various possibilities will be explored further as the system is implemented.

Once a rule is selected, it will then be run by attempting to match its antecedent against the network. If the system can find a match that includes the node or link

driving the reasoning, the rule will succeed and its consequent will be added to the network; if there is more than one such match, then all the instantiations of the consequent will be added. The nodes and links that are added to the network will also be able to drive inference, so additional candidate rules that match them will be collected and put in the candidate set. After the rule has been run, it will be removed from the candidate set (whether it succeeded or failed), and the system will select another rule to run. This process will repeat until the candidate set is used up or until otherwise stopped, as discussed below.

With backward-chaining inference, the procedure is basically similar, except that the matching and running goes in the opposite direction (naturally). The candidate rules are those whose *consequent* contains nodes and links matching the ones in the subnetwork the system is trying to infer or fill in (the "goal"); that is, those rules that would answer the system's question if their antecedents matched against the network. Selecting a candidate rule is the same as in forward-chaining except that the perspective weights of the (non-variable) nodes and links in the *antecedent* rather than the consequent are considered. The selected rule is handled somewhat differently, though. First, the consequent is matched against the goal subnetwork. If it doesn't match, then the rule is removed from the candidate set and the system goes back and picks another rule (as in forward-chaining). If it does match, then the antecedent is filled in with any instantiations made during the match and is itself matched against the network. If a successful match is found, the result is propagated back up to the original goal, and the inference is complete. If no match is found, then the antecedent becomes an additional goal; the candidate set is expanded, and the process repeats.

Unfortunately, the procedure outlined so far will not terminate except in the case where a backward-chaining inference finds a match with the network or in the unlikely case that all the candidate rules are used. The inference strategy must therefore also have a way to stop inference after a period of time. Whenever inference is invoked, therefore, the system will have to give an indication of how much effort to spend on the inference, based on how much importance the system attaches to the inference. For simplicity, this will be represented by an integer value representing (roughly) the number of inferences to make before quitting. Each driving node or link (for forward-chaining) or goal sub-network (for backward-chaining) will have an "effort" value associated with it. Whenever a rule is attempted, the effort value of its associated node, link, or goal will be decreased by one. If a rule succeeds, the new driving node(s) and/or link(s) or the new goal will take a portion of the old driving node, link, or goal's effort; the division of effort will depend on the relative perspective weights of the rule and the next highest candidate rule and the perspective of the new node(s), link(s), or goal. When the effort associated with a particular node, link, or goal reaches zero, it can no longer drive inference, and the rules collected because of it are removed from the candidate set. This approach ensures that inference will always terminate; that the amount of time spent will depend on the importance of the inference; and that the allocation of effort will concentrate on the concepts and rules that are most in perspective and are thus most easily noticed by the system.

3.6.6 An Example: Is car-1 a sports-car?

The example in Figure 3.1 is repeated here in Figure 3.6 in more detail. The generator wants to know whether **car-1** can be described as a "sports-car", so it is searching for an **instance** link between the **car-1** node and the **sports-car** node. Since the consequent of the "classification" rule in Figure 3.6 matches the goal sub-network, it will be included in the candidate set and (eventually) run. If the **engine-size** and **speed** links are out of perspective, the antecedent will match against the network and the rule will succeed; the result will then be propagated back to the goal, verifying the presence of the **instance** link in question. The generator will then go ahead and use "sports car" as a description of **car-1**. On the other hand, if the **engine-size** and **speed** links *are* in perspective, the antecedent won't match, so the system will continue trying other rules. Eventually the system will use up the allotted effort and fail, the **instance** link will remain unverified, and the generator will have to find some other description for **car-1**.

3.7 Using the Representation to Support Generation

The generator uses the knowledge representation in two different ways. The communicative planner uses the representation to support instantiation of and reasoning about actions, and the linguistic specialists use the representation to construct the linguistic options they present to the planner. The planner's use of the knowledge representation is fairly straightforward: checking for objects that meet some description, or whether a precondition is satisfied, or whether an effect of an action violates a goal, and so on. This kind of use of the representation is not specific to the generator; other components of the (hypothetical) intelligent agent would need to make similar requests. Furthermore, the planning aspects of the communicative planner have already been simplified in order to concentrate on the connection between the planner and the linguistic specialists.[30] The planner's use of the representation is therefore not crucial to the current work and will be kept as simple and limited as possible.

More interesting is the linguistic specialists' use of the representation. The specialists deal with the connection between language and meaning, that is, the connection between the (lexical, syntactic, and pragmatic) vocabulary of language and the elements of the representation. Thus they must (as we shall see) make use of the representation in a manner specific to the generator. The specialists use the representation not for reasoning but rather to find and characterize linguistic options for expressing particular information. Finding options depends primarily on how the linguistic knowledge is indexed; characterizing options involves computing annotations. Thus the central issues in using the representation are the nature of the indexing of linguistic knowledge and the nature and derivation of the annotations.

[30] See Section 2.4

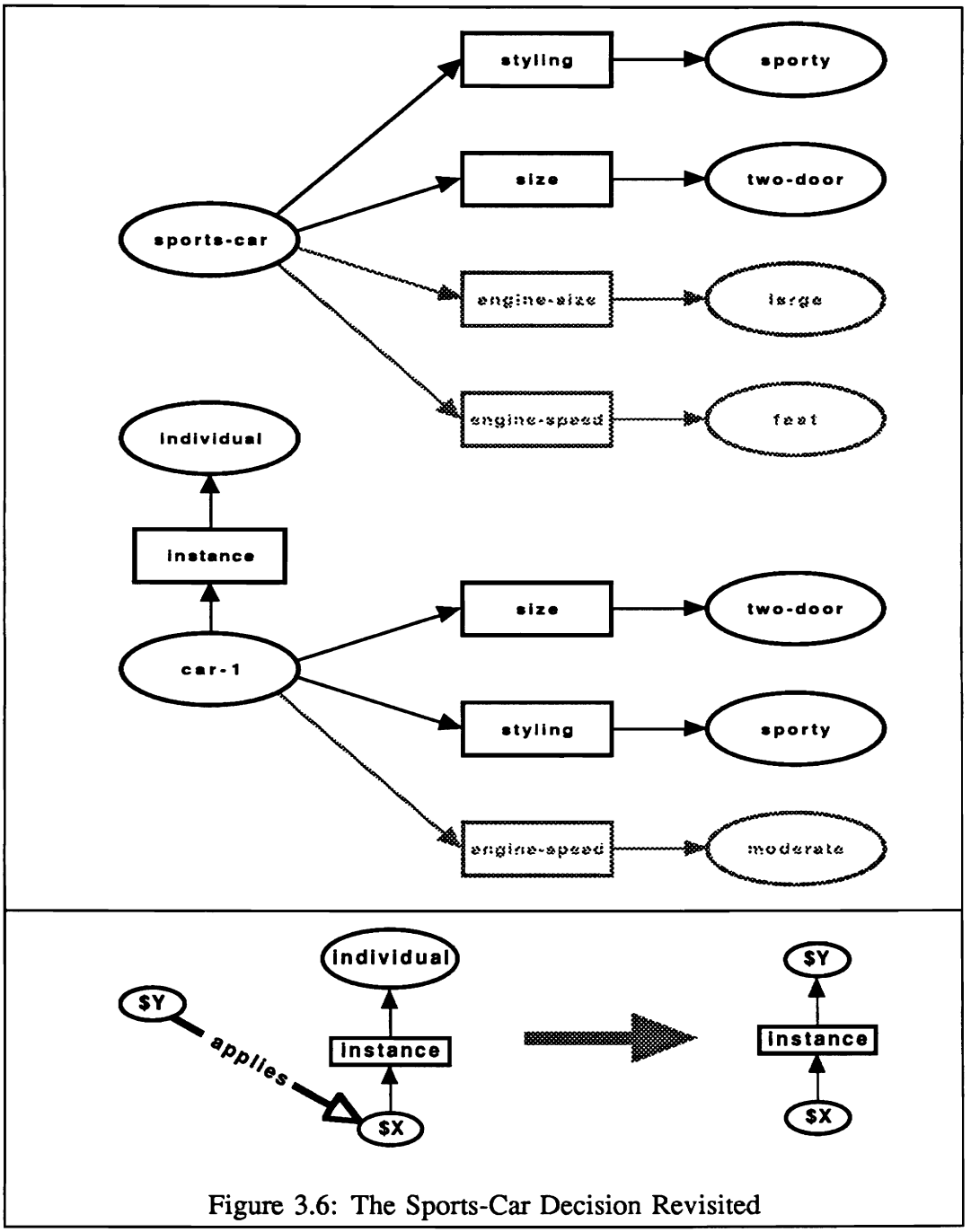


Figure 3.6: The Sports-Car Decision Revisited

3.7.1 Indexing

The first job of the linguistic specialists is to find linguistic expressions that correspond more or less to the information the planner wants to communicate. Each expression in the system's vocabulary (words, phrases, or constructions), wil have a link to some element in the semantic network.[31] The obvious solution would be to simply pick an expression that's linked to whatever node or link the planner proposes. This will only work if the node or link has a linguistic option directly connected to it, though, and not all of them will. Also, the linguistic specialists must be able to suggest a single expression for planner requests that contain multiple nodes or links. Even in cases where there is an expression directly linked to the planner's request, the generator should have the flexibility to consider alternate expressions. So there must be a more flexible indexing scheme linking semantic elements to linguistic expressions.

Fortunately, the indexing scheme doesn't have to be very precise, because inappropriate expressions will be filtered out later. Any indexing scheme that manages to include all (or most) of the reasonable options will do. The one used here will work as follows: the system will scan through the network, starting from the node(s) and/or link(s) in the planner's request, guided by the perspective weights as described in Section 3.5. Whenever the system scans through a node or link connected to a linguistic expression, that expression is added to the list of options. Many of these will be wildly inappropriate, of course, but the annotations placed on them will reflect that, leading the planner to give them very low evaluations. The most appropriate options will generally be included, though, since the procedure will focus on ways of expressing concepts closely connected, in the current perspective, with the concepts in the planner's request.

There are a number of ways that this procedure can be fine-tuned. For example, should the options be considered (and thus presented to the planner) in the order that they are encountered, or sorted according to perspective weight? Is it adequate to allow annotation and evaluation to filter out clearly inappropriate options, or should the linguistic specialists do some amount of pre-filtering of the options? (For that matter, what sort of options are "clearly inappropriate"?) The major problem, though, is that this procedure can be very inefficient, since it could involve a lot of searching through the network. The solution to this is to limit the search; experience will hopefully indicate a reasonable compromise that provides enough options without taking too long. Even in the (very efficient) limit of no search at all, this approach will simply reduce to using expressions directly linked to the planner's requests, so it would still be useful.

3.7.2 Annotating

Since the planner can't understand linguistic expressions themselves, the specialists must provide the planner with descriptions of the options they build that it can

[31] These links are distinct from the links within the semantic network; they are just simple fixed links connecting an expression to its meaning.

understand. These descriptions consist of annotations that the specialists attach to the options placed in the workspace. The annotations are based on a small fixed set of predicates that describe the effects of using particular choices, both in terms of the information they express (or omit) and in terms of how they fit that information into the evolving discourse context. Some annotations are associated specifically with particular linguistic expressions such as idioms and some syntactic constructions. Others can be read directly off the network (as viewed from the current perspective). Annotations can also be derived from the effects on the perspective of using a particular expression; still other sources of annotations may arise as the system is implemented. Additional *types* of annotations may also prove necessary if the planner needs more information.

The annotations consist of a predicate, possibly with one or more arguments, that describe the option in some way. The arguments will most often be nodes or links in the semantic network, taken either from the planner's request(s) or from parts of the network examined by the specialists. The arguments may also consist of pointers to nodes (or links) connected by links in the planner's request(s). For example, <**arg** `over-time-span` **2**> would indicate the number 2 argument of an `over-time-span` relation in the planner's request. (When the relation is clear from context, this will be written more simply as <**arg 2**>.) This kind of argument allows the annotations to describe effects related to pieces of the planner's request not covered by the current option.

The annotations produced by the linguistic specialists will include:[32]

3.7.2.1 Annotations Read off the Network

makes-explicit Indicates what information is being expressed explicitly by the linguistic option. Thus "hit" expresses the notion of hitting something explicitly, whereas, say, "assaulted" doesn't. This information may seem trivial, but the planner still needs it because it can't interpret the linguistic expressions directly. Every option will therefore have a **makes-explicit** annotation indicating the node or link it is connected to.

makes-implicit Indicates information that is not explicit in the linguistic option but is conveyed implicitly. Thus (as above) "assaulted" conveys implicitly the information that someone hit someone (or something), or "assassinate" conveys implicitly that someone was killed. A **makes-implicit** annotation annotation will be generated whenever there is a chain of **instance** and/or **subconcept** links[33] connecting the planner's request and the node or link connected to the linguistic option.

[32]For convenience, the annotations are classified here as they would be computed by a simple specialist using only a fixed set of linguistic expressions. Other specialists might compute the annotations differently.

[33]Note that the other types of "isa" links (**subset** and **part**) cannot be used in these chains. The **makes-implicit** annotations indicate when the option actually describes the request, but implicitly. Although **subset** and **part** can license property inheritance, they don't indicate that the "parent" concept carries the "child" concept as part of its meaning.

indirectly-suggests Indicates information that is not directly conveyed by the option but is still strongly suggested by it. For example, "assassination" doesn't actually say that someone was shot (as opposed to strangled or suffocated, say), but it suggests it. Similarly, "weather" suggests things like the temperature or precipitation. An **indirectly-suggests** annotation annotation will be generated whenever the planner's request has a direct connection in the network to the concept linked to the linguistic option unless the connection is an **instance** or **subconcept** link (in which case a **makes-implicit** annotation will be generated).

missing-info Indicates information in the planner's request that is not conveyed at all by the option. If the request and the option's associated node or link are connected by a chain of **isa** links (of any type) then **missing-info** annotations will be generated for any links that occur only on the request.

As described here, these annotations all simply indicate structural relationships within the network, and hence might better be called **matches-request**, **isa-chained-to-request**, and so on. The actual names, though are used because they reflect how the planner (which doesn't understand the network structure) uses the annotations.[34] Furthermore, as the system is developed, more sophisticated linguistic specialists may construct these annotations in different ways that don't directly reflect the network structure.

Two potential modifications to the use of these annotations are immediately apparent. The distinction between the **makes-implicit** and the **indirectly-suggests** annotations is not entirely clear and may need to adjusted. Also, as described so far, the annotations will only refer to the planner's request and the node or link associated with the linguistic option (except for **missing-info** annotations). It might be useful to add annotations about other nodes. For example, if the planner requests options for **shoot** and "assassinate" is produced as an option, the system might add an **(indirectly-suggests politically-motivated)** annotation, even though the **politically-motivated** link isn't the actual request.

3.7.2.2 Annotations Computed from Perspective and Discourse Context

These annotations assume a notion of a "global focus space" or "attentional state" containing the concepts that are currently active in the discourse (as in, for example, [Grosz 81,Grosz 85]). The generator doesn't currently have any such mechanism for tracking the discourse context, but it would be straightforward to add one. Thus these annotations will be used if and when such a mechanism is added. Actually, the **from-context** annotation is the only type that *requires* a context model to be useful; the others will be useful even if a context model is never developed. Furthermore, the planner's use of these annotations doesn't

[34] What the annotations are called is, of course, irrelevant to how the system actually uses them. Nevertheless, the names indicate the annotations' intended role in the system, so it's important that they not be misleading.

depend on there actually being a context model; it can still reason about how an option's effects on the context fit into its overall plan. Thus it may be possible to use these annotation types even without adding a context model.

activates-in-context Indicates concepts that will be brought into the discourse context by the use of this option. These annotations are added for every concept that the node or link associated with the option shifts into perspective. This assumes that the user's perspective shifts will mirror those of the system, which is of course not reasonable in general. It's a reasonable approximation, though, for a system with no model of the user. This process also assumes that the concepts added to the context are (at least) the ones shifted into perspective, but this is a reasonable and natural assumption.[35]

from-context Indicates an item in the discourse context that the option depends on, either as part of the reasoning that identified it as a valid option or (for options associated with links) as something connected by the link. This planner can use this information to evaluate whether it wants to reinforce the item already in the context. These annotations cannot be generated by the system as here described, because there is no model of the context.

relates-to Used when an option describes a request in relation to some other concept. For example, comparatives (naturally) compare some property of the request to something else. Or a specialist might propose "next to the door" as an option for a particular physical location. This kind of relative description is most likely with something in the discourse context, but it's possible with any concept, so these kinds of annotations can be generated even without a context model. The question of when the linguistic specialists should generate relative descriptions is not yet clear, though, so these annotations will also not be generated initially.

3.7.2.3 Other Annotation Types

tone

simple-construction

awkward-construction These annotations all give the planner a sense of how the option will be perceived as a use of language; that is, what sort of language the use will see the system as using. Since there is no particular theoretical model of language style being used here, these annotations will simply be directly associated with particular expressions as seems appropriate.[36]

[35]That doesn't mean it's correct, of course, just that it's a reasonable way to build the initial system. Looking at anaphoric expressions in actual discourse to see what gets into the context could help refine the creation of these annotations. It could also suggest what sort of perspective shifts are appropriate.

[36]"Awkwardness" often depends as much on how the options are combined as on the particular options; for this reason the utterer will have a rudimentary (ad hoc) model of awkwardness. Initially this will just amount to enforcing grammaticality as much as possible; see Section 2.6 and Page 46.

Linguistic Annotations There will have to be annotations to indicate how options affect the linguistic pragmatics of the utterance, i.e. issues such as the "local focus" or "center" [Grosz 83,Grosz 86a] or the distinction between given and new information [Chafe 76].

Request Partitions There will have to be annotations that indicate which parts of which requests a particular option covers. For the simple type of specialist discussed here these are not needed, because the option exactly matches one request. In general, though, specialists can produce linguistic options that only cover part of a request, or that cover (parts of) more than one request. The planner (and utterer) will need to know when this happens.

3.7.3 An Example In Detail

With the machinery developed in this chapter, we can now see how the linguistic specialists produce the annotated options in the example in Section 2.8, where the planner has requested options for:

a) `temperature within-range`[37]

b) `<60°F 80°F>`

c) `over-time-span`

d) `<September 25 1987>`

The linguistic specialists respond to these requests using the information shown in Figures 3.7 and 3.8; each part of the figure is used for one part of the request. (Note that the hollow arrows are pointers from linguistic expressions into the network, not part of the network itself.) The system's semantic network will be much larger than what is in these figures, of course, but they show the portion of the network that the specialists will draw on for this example. The options discussed here will all be produced by the simple network search technique discussed in Section 3.7.1. In the actual system, there will presumably be other options (and other annotations), but these will be produced and used in the same way as the ones discussed here.

For `temperature within-range` the options are:

- "The temperature be": This option is directly connected to the **temperature** node and the **be-located** link, so it will generate **makes-explicit** annotations for both of them. It will also generate a **makes-implicit** annotation for **within-range**, since it is in the request and connected by a **sub-concept** link to **be-located**. So the complete annotated option will be:

  "The temperature be": `(makes-explicit temperature)`

[37]These two pieces of the request are combined to simply the example; the options and annotations constructed by the linguistic specialists would be essentially the same if they were considered separately.

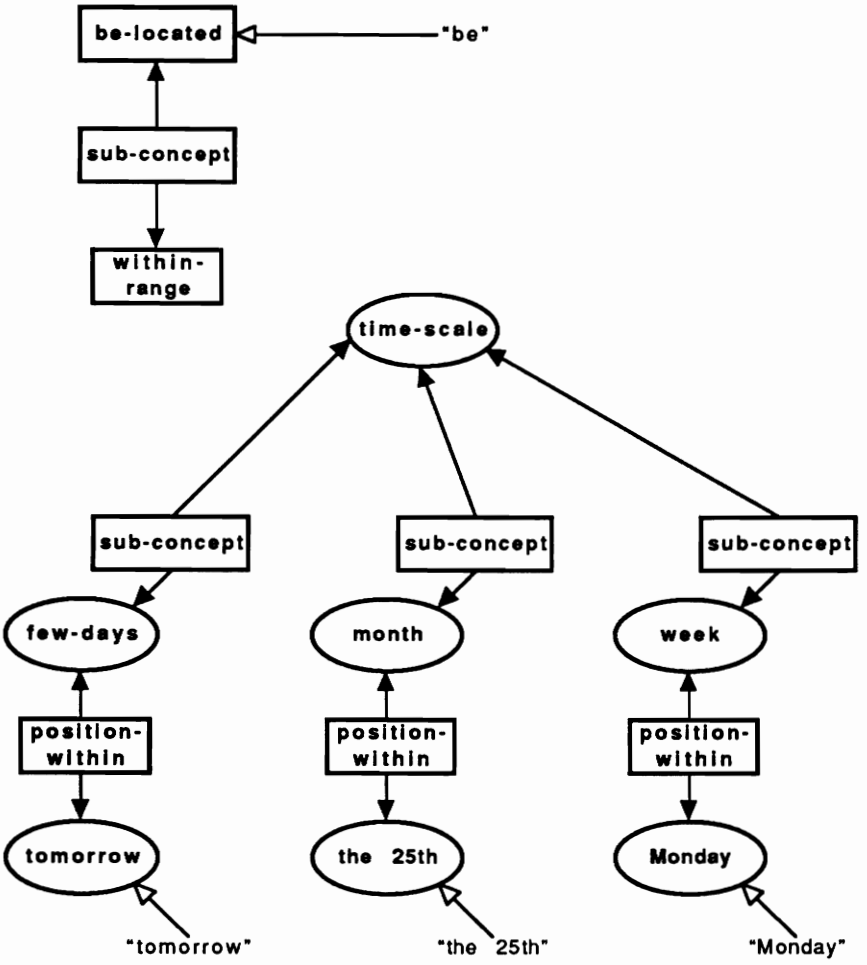


Figure 3.7: Some Semantic Network Fragments Used by the Linguistic Specialists

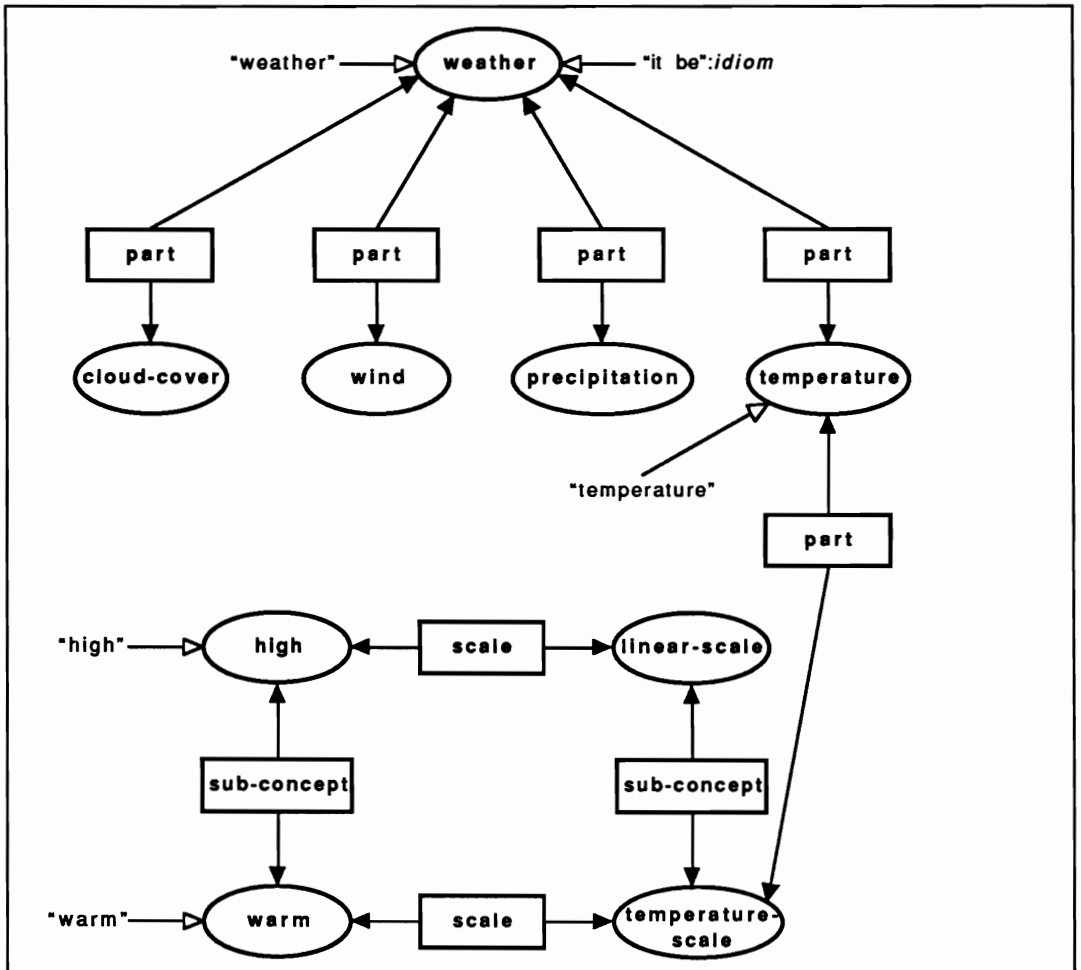


Figure 3.8: More Semantic Network Fragments Used by the Linguistic Specialists

(makes-explicit be-located)

(makes-implicit within-range)

- "The weather be": This is similar to the previous option, except that it is connected to **weather** rather than **temperature**. Thus instead of the **makes-explicit** annotation for **temperature** there will be one for **weather**. Since **temperature** is connected to **weather** by a **part** link, there will also be an **indirectly-suggests** annotation for **temperature**, so the complete option will be:

  "The weather be": (makes-explicit weather)

  (indirectly-suggests temperature)

  (makes-explicit be-located)

  (makes-implicit within-range)

- "It be": This item is an idiom, so it has some special processing. It is annotated as if it were connected to a "weather-it" node that has **sub-concept** links to everything the construction can be used for. There isn't actually any such node, though; the specialist simply acts as if there were. Thus there will be a **makes-implicit** annotation for **temperature** without a corresponding **makes-explicit** annotation, since the "explicit" node is the "weather-it" pseudo-node. "Be" is still connected to the **be-located** link, though, so there will be annotations for both **be-located** and **within-range**. In addition, this expression has a **simple-construction** annotation associated with it. The resulting annotated option is thus:

  "It be": (makes-implicit temperature)

  (makes-explicit be-located)

  (makes-implicit within-range)

  simple-construction

The specialists' response to <60°F 80°F> is similar. Here, though, verifying that these options do in fact describe the request depends on reasoning about positions within linear scales. (That's how the specialists know to suggest "warm" rather than, say, "cold".) This reasoning will be handled either by encoding a model of position in the network and inference rules or, if that proves too difficult, simply building it into the representation. Here, we simply assume the reasoning has been done in some way. The options include:

- "warm": Since there is an (inferred) **instance** link between the **warm** node and the request, the specialist simply generates a **makes-explicit** annotation for **warm** and a **makes-implicit** annotation for the request.

- "high": This will generate a **makes-explicit** annotation for **high** and a **makes-implicit** annotation for the request. Since the request has a **scale**

link to **temperature-scale** and **high** doesn't, the option will also have a **missing-info** annotation for that link.

- "warmer": This option would not be generated by the current machinery, as it depends on having an explicit context model and on specialists that can produce options relating the request to items in the context. If a context model and a specialist that can use it are developed, though, it would need to produce (in addition to appropriate **makes-explicit** and related annotations) a **relates-to** annotation annotation for the temperature item in the context it is comparing the request with. This should be straightforward, since the specialist will have to explicitly relate the two. Similarly, the specialist should be able to construct **from-context** annotations for the cooler temperature item.

For `over-time-span`, the options will be evaluated by the planner primarily on the basis of perspective-based annotations, so the discussion here will ignore the annotations like **makes-explicit** that are read off the network. (These annotations will still be generated, of course.)

- tense=future: Since tense is compatible with any time-scale of any size, it doesn't induce any perspective shift, so there won't be any annotations.
- "during":
- "over": Both of these options are connected to relations that locate something within a span or range, as opposed to at a single position. Thus they shift into perspective an **instance** link between that argument and the generic **linear-span** node. Thus they will both generate **(activates-in-context (instance <arg 2> linear-span))** annotations.
- "on": This option is connected to a relation that locates something at a particular position; this is just the opposite of the previous options. As with "over" and "during", it shifts into perspective an **instance** link connected to its position argument, but here the new link connects it the generic **linear-position** node rather than the **linear-span** node. The resulting annotation is therefore **(activates-in-context (instance <arg 2> linear-position))**.

Finally, for `<September 25 1987>`, the options are all connected to nodes that shift into perspective the time-scale within which they are defined. Thus the options have the following perspective-based annotations:

- "tomorrow": `(Activates-in-context few-days-time-scale)`
- "Monday": `(Activates-in-context week-time-scale)`
- "the 25th": `(Activates-in-context month-time-scale)`

As they are computed, these annotated options are placed in the workspace. Once there, they are used by the planner and the utterer as described in Section 2.8 to produce an appropriate utterance.

Chapter 4

Proposed Research

4.1 Proposed System

The ideas presented in the preceding chapters will be tested and refined by building a generator that will talk about the weather. The generator will be driven by various kinds of goals, including not only presenting specific information but also cooperative goals such as "warn the user about the rain" and conversational goals such as "drag out the conversation" or "say something". The effects of varying the perspective will also be explored to see how the generator's output is sensitive to the system's perspective. If time allows, the generator will be extended to other domains and other kinds of motivations for producing utterances. The weather domain will be used initially because it provides a limited but useful amount of sensitivity to perspective. For example, the distinctions between "warm" and "hot" or between "drizzle" and "rain" are largely matters of perspective, as is the difference between "slightly cloudy" and "nearly clear skies". On the other hand, the weather domain avoids the need to model anything about human psychology or sensibilities, as would be required in domains such as art, politics, or economics, which are highly sensitive to perspective.

4.2 Areas of Investigation

While many of the details of the generator design have been worked out in the preceding chapters, there are still a few questions that must be resolved before a working generator can be built. The form of the initial plans must be made more precise, for example, as well as the mechanism for deciding when and how to revise the plan. The criteria that the utterer uses to balance time pressure against fluency and appropriateness of options need to be worked out. More generally, the basic semantic network and set of inference rules that the generator will use must be built. These problems don't seem particularly serious, though; they will be worked out as the generator is implemented.

In addition, there are a number of areas where the effects of different choices can be explored. Examples include:

- What should the cutoff be for a node or link to be out of perspective?
- Should perspective shifts be sensitive to the old weight of the node or link being shifted and the weight of the node or link causing the shift? If so, how?
- How should nodes and links be shifted out of perspective?
- How low should a node or link's weight be in order for its use to induce perspective shift?
- How should reasoning effort be split when there are several possible paths to reason over?

There are also possible extensions to the design that could be investigated:

- More sophisticated linguistic specialists. In particular, the specialists might be given more freedom to diverge from the planner's requests, given a sufficiently descriptive set of annotations.
- More flexible inference rules. The rules might be allowed to have variable structures in the antecedent, or to have variable nodes (or links) that are restricted to match some range of nodes.
- A way to resolve contradictions. The knowledge representation will sometimes produce contradictory information (see Section 3.6.4). This should not be a problem for the generator, but if the representation is to be used for more general reasoning, there must be a way to revise the system's beliefs to eliminate contradictions (or at least to limit their consequences).
- New annotation types. The annotation types discussed in Section 3.7.2 are only an initial set; they are not expected to be adequate for everything the linguistic specialists will need to indicate. As the linguistic specialists are improved, they will undoubtedly need a richer set of annotation types.
- Context model. Some of the annotation types discussed in Section 3.7.2.2 assume some sort of model of the discourse context. The generator currently has no such model. Eventually, though, a context model will be necessary.
- Hearer model. It's not clear whether there should be a separate model of the hearer, or whether there should simply be information about the hearer within the general world knowledge. Either way, though, the generator will need to be able to take this information into account when constructing its utterances.

4.3 Criteria for Evaluation

The minimal goal of the proposed research is to develop a generator along these lines that works (or to understand why it won't work). Given that the design has made a number of compromises for practical reasons, and that further modifications are likely during implementation, it's important to remember what it means for a generator to be "along these lines". What's important is that the generator follow the general principles outlined in Chapter 1. In particular the central constraint on the generator is that choices at all levels are (at least potentially) sensitive to the goals the generator is working towards. As long as the generator meets this constraint, it will be using language as a tool with which it can act, rather than a code into which it must encode messages.

The extent to which the generator actually meets the criteria laid out in Chapter 1 can be tested in a straightforward fashion. Simply run the generator on an example, remove some or all of the linguistic options used and run it again on the same input. The generator should still be able to produce a meaningful and appropriate utterance. This shows that it is really making choices on the basis of an understanding of what the language means and what the goals are, rather than on some kind of direct mapping between the generator's input and linguistic structures. If more and more linguistic options are removed from the generator, its output should become progressively less fluent but still appropriate given its goals.

Another way to test the generator is to vary the system's perspective and see how the generator's output varies. The generator should tend to talk more about things that are more in perspective and to describe things as seen in the current perspective; but shifts in the perspective should not alter the generator's output so much that it no longer achieves its goals. The generator should demonstrate appropriate sensitivity to both the perspective and the goals it is working towards.

Beyond the minimal goal, there are more ambitious possibilities. The generator could be extended to work in other domains. If this were just a matter of building a new lexicon and modeling the new domain in the knowledge representation formalism, it wouldn't really be very interesting. The intent here, though, is to have a single world model that can be used across all domains. Extending the generator to other domains would involve identifying the core of its world knowledge that is common to the domains and using perspective to filter out the information unique to each domain. Thus the generator would be able not just to work in several domains, but even to work across several domains, i.e. to deal with multiple domains in the same utterance.

The generator might also be extended to handle a wider range of types of goals, particularly the kinds of interpersonal goals such as "be reassuring" or "be aggressive" that are usually background goals. This will also test the limits of what the representation and architecture can handle.

These more ambitious goals will be pursued if time is available. Even the minimal goal, though, would establish the feasibility and utility of this work's principal aim: to develop a natural language generator that is conceived of as an

integral part of an intelligent system.

Appendix A

Focus of the Proposed Research

This proposal, and this project, deal with a broad range of issues. The central focus, though, is the production and use of the annotations on the linguistic options. This can be seen by considering the basic argument underlying the proposal, which runs roughly as follows: A generator for an intelligent system will need to respond to a wide range of pragmatic concerns involving the situation in which and purpose for which it is generating utterances. Since the knowledge on which the generator depends must be sensitive to the widely and dynamically varying perspective, it's not possible to organize the generator's linguistic options in advance around the various pragmatic factors. Instead, the generator must be able to compare the effects of particular choices against the various pragmatic concerns underlying its current generation. This is only possible if there is a way to mediate between the generator's linguistic knowledge and its goals; this is what the annotations do. The annotations make it possible to separate out the work of planning the utterance from the detailed construction of the utterance; it is the central claim of this proposal that this can be done without sacrificing the generator's sensitivity to perspective and to high-level goals.

The annotations are thus critical to the success of this work. The other parts of the generator can safely be simplified or compromised to some extent, because they are both less critical and less controversial than the annotations. The planner, for example, has been simplified in certain ways (see Section 2.4), but this is not a serious weakness because a more sophisticated planner would fit into the generator in the same way. Furthermore, the notion of planning in generation is widely (though not universally) accepted; so compromising on the implementation of the planner doesn't weaken the novel claims of the proposal. Similarly, the phenomena identified as effects of perspective will clearly have to be included somehow in the knowledge supporting generation, so the fact that the mechanism here is only a partial solution is not critical. On the other hand, developing a reasonable and practical set of annotation types is crucial, both because the annotations are a novel technique not used in other work and because if the annotations are not adequate, then the project as a whole will have failed to achieve its aims.

The central focus of this proposal is thus to develop an adequate and appropri-

ate set of annotation types. There are basically two constraints on the annotations: they must not include any purely linguistic information, and they must provide the planner with whatever information it needs to evaluate the options. That is, they must maintain the separation of the linguistic and planning components but provide enough of a bridge between the components to allow them to work together.

Keeping purely linguistic information out of the annotations is fairly simple, assuming an appropriate definition of "linguistic" as opposed to "conceptual" knowledge. One can certainly quibble about this, but as a rough guideline I assume that anything that depends on the particular language used is "purely linguistic", while anything that is independent of the particular language is not. Thus the fact that "it be" can be used to talk about the weather is purely linguistic; whereas the concept of weather is not.[1] Aspects of discourse structure, such as the local focus or center of a sentence or the use of cue words to indicate discourse segmentation are more problematical. I will simply assume that, to the extent the planner needs to know about them, they are aspects of communication rather than purely linguistic phenomena; hopefully this assumption can be played out without becoming implausible.

The main issue in designing the annotations is ensuring they provide enough information to the planner. The planner needs to know three things about the options:

1. What information does it express (and omit), and how directly? Without this, the planner wouldn't know what it was saying.

2. What other effects will the option have on the hearer? The planner needs to know how a particular option will affect the hearer's awareness of and beliefs about things beyond what's directly mentioned. The **activates-in-context** annotations, for example, serve this function. There might also be annotations indicating effects such as insulting, offending, or pleasing the hearer (to the extent that these effects are caused by purely linguistic factors, rather than by the information expressed by the option).

3. How does this option interact with other things that have been or may be said? Using a particular option may conflict with, depend on, or change the meaning of another part of the utterance; the planner needs to know this in order to evaluate the option.

The annotation types listed in Section 3.7.2 make a start at providing this information, but they are clearly not sufficient. The examples in Appendix B indicate a number of problems that will have to be solved, including:

[1]This ignores the fact that different languages lexicalize different concepts, or different perspectives on a concept. I am assuming here that since it is possible to explain, for example, how the Hebrew word "kadosh" differs from the English word "holy", that there is such a thing as conceptual knowledge that is distinct from linguistic knowledge. Without this assumption, it would be harder to state what the constraint on the annotations should be, but the generator would still work the same way.

- The distinction between explicit and implicit information needs to be more subtle. Currently there are only two levels of "explicitness" (or three, if you count the **indirectly-suggests** annotation type). There must be finer distinctions to capture the fact that, for example, "be" is a much better option for `consists-of` than "precipitation" is for `rain`.

- There must be a way to indicate interactions and constraints between different options. For example, "it be" can express `temperature consists-of` if it's followed by "warm", but not if it's followed by "pleasant". Also, in Example 5 in Appendix B, the planner must know that the same option must be used for both instances of `no-precipitation` in order for `repeat` to be realized.

- There must be some way to appropriately control redundancy in the utterance. For example, it's okay to say "the temperature will be warm", but not "the temperature will be a high temperature". It's not immediately clear what the relevant constraints are, but they clearly involve the particular options chosen (e.g. "it's raining a little a little" is horrible but "it's only raining a little" is fine), so the annotations will have to express whatever information is relevant.

- There must be annotations describing how options affect discourse structure, indicating such things as which element will be the local focus and when options influence discourse segmentation.

Extending the set of annotation types to deal with these sorts of problems will be the main focus of the work outlined in this proposal.

Appendix B

Additional Examples

Example 1

There is one further point raised by the example in Sections 2.8 and 3.7.3. Consider the option "it be", suggested for `temperature within-range`. It gets a **(makes-implicit temperature)** annotation, which indicates that it using it will express the concept of temperature (although only implicitly). Unfortunately, whether it actually does this depends on what else gets said. In the present case, both "the temperature will be warm" and "it will be warm" express the notion of temperature. But suppose, instead, the choice is between "the temperature will be listed in the newspaper" and "it will be listed in the newspaper". Clearly "it be" *isn't* expressing the concept of temperature here.

There are two problems here. The first is that the idiomatic use of "it be" has some constraints that haven't been captured here. This isn't so troubling; this idiom already gets special handling, so the constraints could presumably be built in somehow. The real problem is that the meaning of the expression depends in part on what other expressions are being used. Thus "it will be warm" is fine, whereas "it will be high" isn't. The problem isn't the use of the word "high"; "the temperature will be high" is fine. The problem is that the constraints on the use of "it be" involve choices made about other parts of the utterance. The annotation set doesn't currently have any way to indicate this; it will have to be augmented to do so.

Example 2[1]

Continuing the previous example, the planner might intend to talk next about the rain, producing:

```
(precipitation consists-of rain) over-time-span
<September 25 1987>
```

For `precipitation consists-of` the options produced include:

[1]These examples depend in part on the semantic network fragments in Figures B.1 and B.2

"The precipitation consist of":

```
(makes-explicit precipitation)
(makes-explicit consists-of)
```

"The precipitation be":

```
(makes-explicit precipitation)
(makes-implicit consists-of)
```

"It be":

```
(makes-explicit precipitation)
(makes-implicit consists-of)
simple-construction
```

The planner will rank the first option high, since it explicitly mentions everything the planner wants mentioned. The second option will receive a lower rating, since it only implicitly mentions `consists-of`. The third option is more problematic. On the one hand, it also only mentions `consists-of` implicitly; on the other hand, it is marked as being a simple construction. So its ranking relative to the first option depends on which factor predominates. Since I don't have any particular basis to decide this, I will assume it is given the same rating as the first option.

For `rain` the options include:

"rain":

```
(makes-explicit rain)
```

"precipitation":

```
(makes-explicit precipitation)
(makes-implicit rain)
(missing-info (form rain liquid))
```

"rainy":

```
(makes-explicit rain)
```

"rains":

```
(makes-explicit precipitation)
(makes-explicit consists-of)
(makes-explicit rain)
(includes-other-request-part precipitation)
(includes-other-request-part consists-of)
```

"snows":

```
(makes-explicit precipitation)
(makes-explicit consists-of)
(makes-explicit snow)
(missing-info (form <request> liquid))
(extra-info (form <request> solid))
```

```
(includes-other-request-part precipitation)
(includes-other-request-part consists-of)
```

Here the highest rating goes to "rains", since it explicitly mentions the requested information, and also mentions some other information the planner wants to express. Next come "rain" (as a noun) and "rainy", both of which explicitly mention the request. Lower still is "precipitation", which only mentions the request implicitly and leaves out some information, and least of all is "snows" which is farthest of all from the request.

For `over-time-span` and `<September 25 1987>` the options are the same as in the previous example.

Ignoring for a moment the "rains" option, the utterer will assemble several possible utterances:

1. "The precipitation will consist of rain [over tomorrow]"
2. *"The precipitation will consist of rainy ..."
3. *"It will be rain ..."
4. "It will be rainy ..."

The second possibility is ruled out because it's ungrammatical ("of" needs a noun phrase, not an adjective). The third possibility can also be ruled out if we assume that the "it be" idiom requires an adjectival complement, which seems plausible. Even if that's not right, there must be some fairly specific criterion that distinguishes between the third and fourth option, which differ only in the syntactic category of the complement. The two remaining options have essentially the same ranking, so the utterer will choose either one.

Actually, this doesn't seem right; the "It be" option seems much better, even though it only mentions `consists-of` implicitly. It may just be that the idiom is strongly favored, and should automatically get a high ranking. There are two other possible relevant factors, though. First, using "be" for `consists-of` seems much less of a problem than, say, using "precipitation" for `rain`, even though they have the same structural relationship in the network. So perhaps there must be a finer-grained distinction than just the implicit/explicit one used here. Conversely, the issue may be that some parts of the request are more important than others. Here, for example, the most important concept seems to be `rain` (perhaps because `precipitation consists-of` can be inferred from it?); thus omitting it is more serious than omitting other parts of the request. Thus saying "it will be rainy" is much better than "the precipitation will consist of precipitation", even though the annotations suggest that it leaves out just as much information.

In the current example, this problem is moot, because the utterer will actually select "rains", the highest ranked option for `rain`; since this option also covers `precipitation consists-of`, it will be the only option selected. In this case, that works out fine, but there would be a problem if there were an option that

covered one part of a request (or requests) well but another part badly. Should the option be rated for the sum of what it covers, or separately for each part, or both? The obvious approach would be to always rank options for everything they cover, but this doesn't always give the right answer. For example, "later" might be a high-ranked option for `over-time-span` that could also cover the particular time span (`next-week`, say), but express the particular time very poorly. Thus the overall rank might be low, even though the best utterance might be "later" plus some further specification of the time span, e.g. "later in the month". So the planner must evaluate the options in response to the (parts of) requests that they respond to; perhaps it should provide overall evaluations as well.

The last two requests, `over-time-span` and `<September 25 1987>`, produce either "tomorrow" or "on Monday", as in the previous example. Here, though, this information is really redundant and should be omitted. This should probably be the responsibility of the planner, not the linguistic specialists or the utterer. It might be reasonable, though, for the specialists to be able to notice the repetition and suggest "and" as an option for these two pieces of the request. This would require some additional annotations to indicate that the effects of this option depend on which options are chosen for the previous request.

Given the current machinery, the resulting utterance will be either "it will rain tomorrow" or "it will rain on Monday".

Example 3

The generator is given the goal "get the user to take his umbrella", and constructs the following plan:

1. Goal: `(take user umbrella)`
2. Cause (1) by achieving sub-goal: `(want user (take user umbrella))`
3. Cause (2) by achieving sub-goal: `(want user (protect user rain))`
4. Cause (3) by achieving sub-goal: `(know user (precipitation consists-of rain)`
5. Achieve (4) by saying `(precipitation consists-of rain) at-time now`; this request is now placed in the workspace.

The planner might also explicitly say "take your umbrella", and only use the information in (5) to motivate this, but that's irrelevant to the example. Note that the request placed into the workspace is similar to the one in Example 2, but it fits into a very different plan.

The options for `precipitation consists-of` are the same as in the previous example, and again "It be" and "The precipitation consist of" get the highest rating. For "rain", all the options in the previous example show up, but some other options are also relevant: (These options will show up in Example 2, also, but they are down-rated straightforwardly there)

"drizzle":

```
(makes-implicit rain)
(makes-explicit drizzle)
(extra-info (strength <request> gentle))
```

"drizzles":

```
(makes-explicit precipitation)
(makes-explicit consists-of)
(makes-implicit rain)
(makes-explicit drizzle)
(extra-info (strength <request> gentle))
(includes-other-request-part precipitation)
(includes-other-request-part consists-of)
```

"pours":

```
(makes-explicit precipitation)
(makes-explicit consists-of)
(makes-implicit rain)
(makes-explicit pour)
(extra-info (strength <request> hard))
(includes-other-request-part precipitation)
(includes-other-request-part consists-of)
```

Since the planner is expressing this information to convince the user he needs protection from the rain, the option that suggests that the rain is heavy (and hence more of a problem) is preferred, and the planner selects "pours", even though it's not the best match to the information.

The options for `at-time` include:

tense

```
(makes-implicit at-time)
(makes-explicit <tense>)
simple-construction
```

tense=present

```
(makes-implicit at-time)
(makes-explicit <tense=present>)
simple-construction
(includes-other-request-part now)
```

and for `now`:

"today":

```
(makes-implicit now)
(makes-explicit today)
(activates-in-context few-days-time-scale)
```

"now":

```
(makes-explicit now)
(activates-in-context time-scale-including-present)
```

There are a couple of problems with these annotations. First, **time-scale-including-present** is a rather ad-hoc concept; there really needs to be a general model of temporal concepts that this fits into before this can be taken seriously. More importantly, the **(makes-implicit now)** annotation for "today" can't simply be read off the network the way the other **makes-implicit** annotations are. "Today" isn't a subconcept of "now", nor is "now" a subconcept of "today". Instead, it seems that the specialists need to draw on some sort of reasoning to determine the relationship between the concepts. Perhaps this should be an effect of the perspective; if the system is only interested in what's currently going on, a **subconcept** link between **now** and **today** might shift into perspective. This one example doesn't seem strong enough motivation to make that move, though; as more cases of this problem turn up, the proper way to deal with it may become more apparent.

The planner will prefer <*tense=present*> over <*tense*> since it covers more parts of the request. As for "now" and "today", since the planner isn't concerned with any particular time scale here, the **activates-in-context** annotations are ignored (in contrast to Examples 1 and 2). Instead, the choice depends on whether the planner wants to mention `now` explicitly. This in turn depends on whether the planner thinks that adding a sense of urgency will be useful: if so, it prefers "now"; if not, it prefers "today". In the latter case, though, the planner will prefer to have the <*tense=present*> option cover `now`, so both "now" and "today" will receive lower rankings.

Finally, the utterer will assemble the highest ranking options, producing either:

- "It's pouring"

  or

- "It's pouring now"

Example 4

The generator is given the goal "make the user happy", and develops a plan to do so by discussing the weather:

1. Goal: happy(user)
2. Achieve (1) by:
   (a) Goal: Downplay unpleasant information
   (b) Goal: Emphasize pleasant information
3. Achieve (2a) by saying: `(minimal-significance (precipitation consists-of rain))`

4. Achieve (2b) by saying: `(temperature within-range <60°F 80°F>)`

5. Achieve (2b) by saying: `(no-precipitation at-time this-afternoon)`

There are a number of difficulties in constructing such a plan, of course. For example, how is the notion of "happy" being modeled, and how does the planner know what sort of information is pleasant or unpleasant? These questions don't have anything to do with generation specifically, so I will ignore them for now and just assume the plan has been built somehow.

Starting with (3), the options for `precipitation consists-of rain` will be the same as in Examples 2 and 3, but they will be evaluated differently here. Here "it be" is clearly favored for `precipitation consists-of`, since it only mentions implicitly some of the information (which the planner is trying to downplay). For `rain`, the preferred options are now "precipitation", which omits some of the unpleasant information, and "drizzle" and "drizzles", which include extra information that mitigates the unpleasantness. When syntactic constraints are figured in, the utterer will choose "drizzles" to cover both parts of the request. (Another plausible option would be "there be precipitation", but I haven't figured out yet how to handle this construction.)

Of course, the information could be even further downplayed by simply not mentioning it. The planner might, in fact, decide to do this, either in the initial plan or when it evaluates the options. In the present case, though, the planner has decided to mention the information in order to explicitly downplay it saying it has `minimal-significance`. The options for this concept are:[2]

"a little":

```
(makes-explicit little)
(makes-implicit minimal-significance)
```

"only":

```
(makes-explicit only)
(makes-implicit minimal-significance)
```

Here there's no way to distinguish the two options, so the planner rates them both as high.

The utterer will thus select either:

- "It's only drizzling"

  or

- "It's drizzling a little"

[2]Handling these options properly would require a more careful analysis of the pragmatics of the expressions. "Only", for example, implicates that its argument is below some threshold in some scale, where the threshold and the scale are contextually determined (or perhaps inferred from the utterance). The present approach will suffice for this example, though.

Actually, if the specialists were a little cleverer, they might have suggested "rains a little" or "a little rain" as possible options for `rain`. These would be given a high rating just like "drizzles" (although perhaps not quite as high, because they do mention `rain` explicitly), allowing the utterer to produce "it's only raining a little". The system would have to avoid saying "it's raining a little a little", though, so the utterer would have to be able to avoid repeated use of the same expression, unless explicitly approved by the planner.

Going on to (4), the options and rating for `(temperature within-range <60°F 80°F>)` will be the same as in Example 1, except that options such as "mild" or "pleasant" may be preferred over "warm", since they reinforce the planner's goal. Thus the utterer will produce either:

- "It's warm"
  or
- "It's mild"
  or
- "It's pleasant"

The last of these options demonstrates the problem discussed above under Example 1; the "it be" idiom for temperature requires a predicate that indicates temperature is being discussed. There's no way currently to indicate that constraint in the annotations.

Finally, the specialists produce options for (5).[3] I will just assume that "it be clear" is chosen for `no-precipitation` and that *<tense>* is chosen for `at-time` in a straightforward manner. This leaves `this-afternoon`, for which the options include:

"this afternoon":

```
(makes-explicit this-afternoon)
```

"later":

```
(makes-explicit later)
(makes-implicit this-afternoon)
(missing-info (scale <request> one-day-time-scale))
(missing-info (between <request> noon six-pm))
(extra-info (after <request> now))
```

"soon":

```
(makes-explicit soon)
(makes-implicit this-afternoon)
(missing-info (scale <request> one-day-time-scale))
(missing-info (between <request> noon six-pm))
(extra-info (close <request> now))
```

[3] I haven't completely worked out the representation for all the concepts in this request, so some of the annotations here will be speculative.

Here the planner prefers "soon", since it suggests that the pleasant event is not far away. Conversely, "later" gets the lowest rank, since it leaves the actual time of `no-precipitation` unspecified. "This afternoon" gets an intermediate rank, since it does indicate when the event will occur but doesn't give any sense of immediacy to it. So the resulting utterance is "it will be clear soon".

The final result is thus one of:

> It's drizzling a little. It's warm. It will be clear soon.
>
> It's only drizzling. It's warm. It will be clear soon.

The major problem with these is that the individual sentences are disconnected. There needs to be some way to make them flow more smoothly. Some of this could be handled by the planner. For example, the planner could request an indication of the rhetorical opposition between the first sentence and the other two. The linguistic specialists might then propose "but" as a way of expressing the opposition, leading to "it's drizzling a little, but it's warm". Also, the specialists might in general suggest conjoining sentences with a common element, and annotations might indicate the consequence of not doing so. This might make it possible to produce "It's drizzling a little, but it's warm and it will be clear soon", which is much better.

Example 5

Suppose the planner is trying to talk about the current drought, and its plan includes the following elements:

n. Say: `(continue drought)`

n+1. Support (n) by saying: `(no-precipitation over-time-span <today's date>)`

n+2. Support (n) by saying: `((repeat no-precipitation) over-time-span <tomorrow's date>)`

This is intended to produce something like:

> The drought is continuing. It's dry today, and it will be dry again tomorrow.

Much of this would be produced straightforwardly along the lines of the previous examples. There are a few particular choices that are worth examining.

First of all, note that `no-precipitation` needs to be expressed as "dry" here, whereas it was expressed as "clear" in the previous example. It's not too difficult to see how this would happen; "dry" would have a **(makes-explicit dry)** annotation, and the planner would presumably see how this reinforces the concept of a drought. This contrasts with Example 4, where the planner is trying to make the user happy. In that situation, the planner prefers "clear", which emphasizes

the lack of rain over "dry", which emphasizes the (possibly unpleasant) lack of any water. Thus the choice of particular words is affected by how they interact with the planner's high-level goals.

In (n+2), `repeat` could be expressed in a number of ways other than "again", such as "also", "as well", or "too". "Again" seems to be preferable, though, partly because it suggests a potentially indefinite sequence of repetitions (as opposed to only two), and partly because it suggests that the repetition is identical (as opposed to merely similar or related). Both of these factors reinforce the sense of the drought continuing over a long period of time. The annotations will have to be able to capture these distinctions; while it's not immediately clear how precisely to do this, it doesn't seem beyond the capacity of the current annotations.

It's important to note, though, that use of `repeat` seems to require that `no-precipitation` be realized the same way both times. That happens here, but only by coincidence; there's nothing in principle preventing the planner from trying to say "it's clear today, and it will be dry again tomorrow." Furthermore, the current machinery doesn't seem to provide any mechanism for expressing or enforcing this kind of constraint. This will need further work.

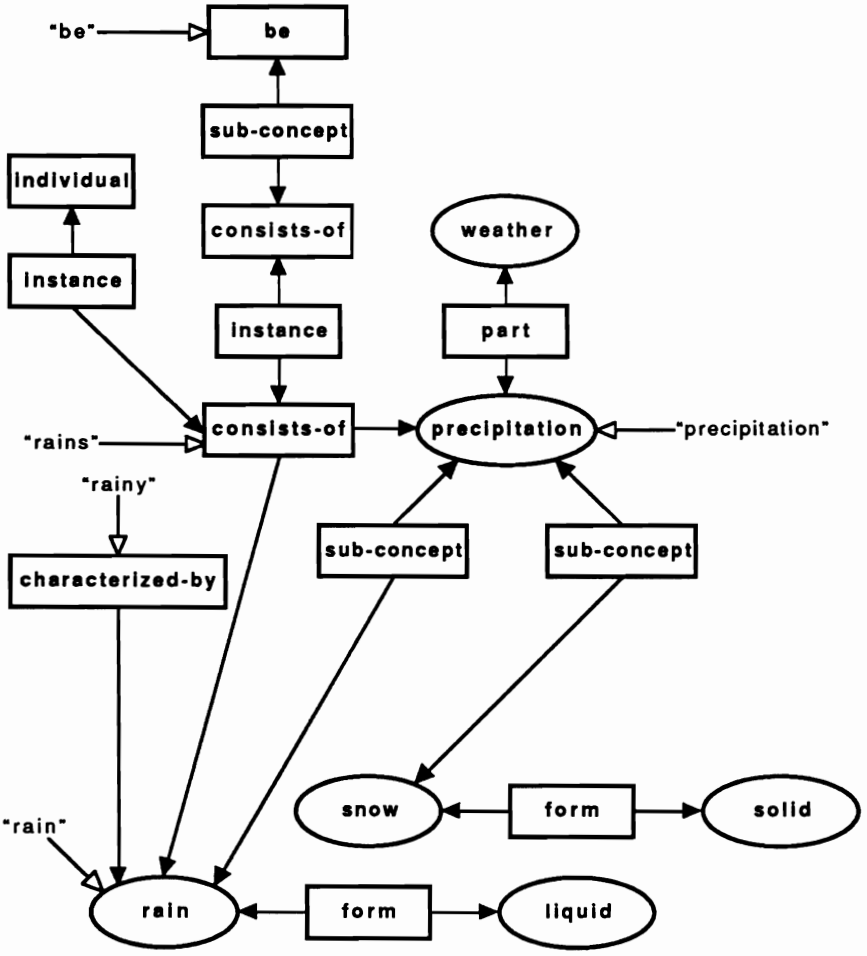


Figure B.1: Semantic Network Fragments Used in Additional Examples

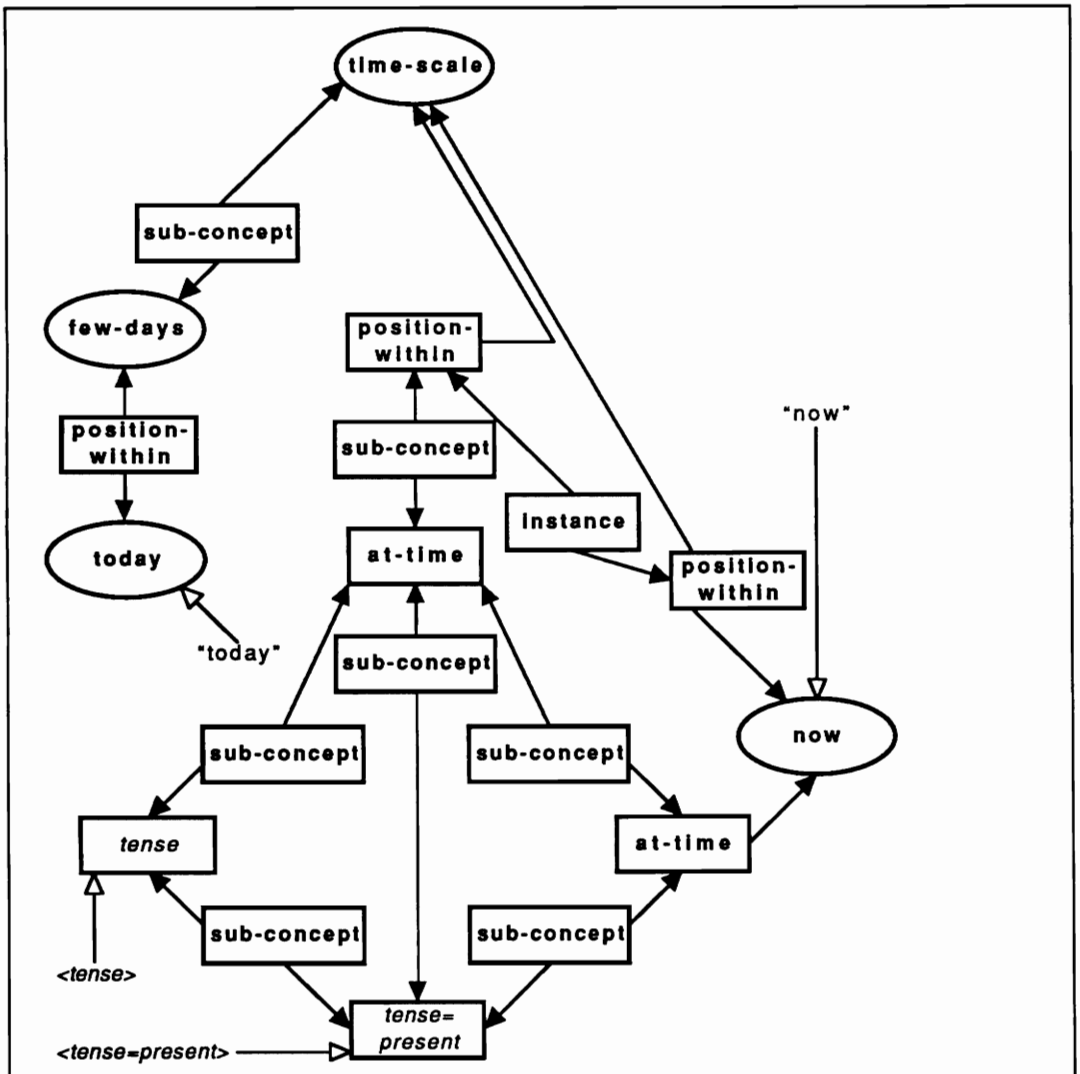


Figure B.2: More Semantic Network Fragments Used in Additional Examples

Bibliography

[Appelt 83] Appelt, Douglas E. TELEGRAM: A Grammar Formalism for Language Planning. In *Proceedings of the 21st Annual Meeting of the ACL*, Association for Computational Linguistics, pages 74–78. Cambridge, MA, June 1983.

[Appelt 85] Appelt, Douglas. *Planning English Sentences. Studies in Natural Language Processing*. Cambridge University Press, 1985.

[Brachman 83] Brachman, Ronald J., Fikes, Richard E., and Levesque, Hector J. *KRYPTON: A Functional Approach to Knowledge Representation.* FLAIR Technical Report 16, Fairchild Laboratory for AI Research, May 1983.

[Brachman 85] Brachman, Ronald. I Lied About the Trees. *AI Magazine* VI(3):80–93, Fall 1985.

[Carnap 37] Carnap, Rudolf. *The Logical Syntax of Language*. Harcourt, Brace, New York, 1937.

[Carnap 47] Carnap, Rudolf. *Meaning and Necessity*. University of Chicago Press, 1947.

[Chafe 76] Chafe, William. Givenness, Contrastiveness, Definiteness, Subjects, Topics, and Point of View. In Li, C. (editor), *Subject and Topic*, pages 25–55. Academic Press, New York, 1976.

[Chin 88] Chin, David Ngi. *Intelligent Agents as a Basis for Natural Language Interfaces*. Technical Report UCB/CSD 88/396, Computer Science Division, University of California at Berkely, Berkely, CA, January 1988.

[Clancey 81] Clancey, W. J. and Letsinger, W. Neomycin: Reconfiguring a Rule-Based Expert System for Application to Teaching. In *Proc. 7th Int'l. Joint Conf. on Art. Intelligence*, IJCAI. University of British Columbia, Vancouver, Canada, August 1981.

[Dale 89] Dale, Robert. *Generating Referring Expressions in a Domain of Objects and Processes*. PhD thesis, Centre for Cognitive Science, University of Edinburgh, 1989.

[Danlos 87] Danlos, Laurence. *The Linguistic Basis of Text Generation. Studies in Natural Language Processing*. Cambridge University Press, Cambridge, England, english translation edition, 1987. Translated by Dominique Debize and Colin Henderson.

[Goldman 75] Goldman, N. Conceptual Generation. In Schank, Roger and Riesback, C. (editors), *Conceptual Information Processing*. American Elsevier, 1975.

[Grice 75] Grice, H.P. Logic and Conversation. In Cole, P. and Morgan, J.L. (editors), *Syntax and Semantics III: Speech Acts*, pages 41–58. Academic Press, 1975.

[Grosz 81] Grosz, Barbara J. Focusing and Descripton in Natural Language Dialogues. In Joshi, A., Webber, B., and Sag, I. (editors), *Elements of Discourse Understanding*, Chapter 3, pages 84–105. Cambridge University Press, 1981.

[Grosz 83] Grosz, Barbara J., Joshi, Aravind K., and Weinstein, Scott. Providing a Unified Account of Definite Noun Phrases in Discourse. In *Proceedings of the 21st Annual Meeting of the ACL*, Association for Computational Linguistics, pages 44–50. Cambridge, MA, June 1983.

[Grosz 85] Grosz, Barbara J. and Sidner, Candace L. Discourse Structure and the Proper Treatment of Interruptions. In *Proceedings of the Ninth International Joint Conference on Artificial Intelligence*, pages 832–839. Morgan Kaufmann, Los Angeles, August 1985.

[Grosz 86a] Grosz, Barbara J., Joshi, Aravind K., and Weinstein, Scott. Towards a Computational Theory of Discourse Interpretation. 1986.

[Grosz 86b] Grosz, Barbara J. and Sidner, Candace L. Attention, Intentions, and the Structure of Discourse. *Computational Linguistics* 12(3):175–204, July 1986.

[Hovy 87] Hovy, Eduard H. Some Pragmatic Decision Criteria in Generation. In Kempen, Gerard (editor), *Natural Language Generation: New Results in Artificial Intelligence, Psychology, and Linguistics*, pages 3–17. Martinus Nijhoff Publishers, Boston, 1987.

[Hovy 88a] Hovy, Eduard H. *Generating Natural Language Under Pragmatic Constraints*. Lawrence Erlbaum, Hillsdale, NJ, 1988.

[Hovy 88b] Hovy, Eduard H. Planning Coherent Multisentential Text. In *Proceedings of the 26th Annual Meeting of the ACL*, Association

for Computational Linguistics, pages 163–169. Buffalo, NY, June 1988.

[Hovy 88c] Hovy, Eduard H. Two Types of Planning in Language Generation. In *Proceedings of the 26th Annual Meeting of the ACL*, Association for Computational Linguistics, pages 179–186. Buffalo, NY, June 1988.

[Jacobs 85] Jacobs, Paul S. PHRED: A Generator for Natural Language Interfaces. *Computational Linguistics* 11(4):219–242, 1985.

[Jacobs 87] Jacobs, Paul S. KING: A Knowledge-Intensive Language Generator. In Kempen, Gerard (editor), *Natural Language Generation: New Results in Artificial Intelligence, Psychology, and Linguistics*, pages 219–230. Martinus Nijhoff Publishers, Boston, 1987.

[Kroch 81] Kroch, Tony and Hindle, Don. A Quantitative Study of the Syntax of Speech and Writing. Final Report to the National Institute of Education Grant 78-0169, 1981.

[Labov 73] Labov, William. The Boundaries of Words and Their Meanings. In Bailey, C.-J. N. and Shuy, R. W. (editors), *New Ways of Analyzing Variation in English.* Volume 1. Georgetown University Press, Washington, DC, 1973.

[Lakoff 72] Lakoff, George. Hedges: A Study in Meaning Criteria and the Logic of Fuzzy Concepts. In *Papers from the Eighth Regional Meeting of the Chicago Linguistic Society*, Department of Linguistics, University of Chicago. Chicago, 1972.

[Lipkis 82] Lipkis, T. *A KL-ONE Classifier.* Technical Report 4842, BBN, 1982.

[Mann 83] Mann, William C. An Overview of the Nigel Text Generation Grammar. In *Proceedings of the 21st Annual Meeting of the ACL*, Association for Computational Linguistics, pages 79–84. 1983.

[McCarthy 80] McCarthy, John. Circumscription: A Form of Non-Monotonic Reasoning. *Artificial Intelligence* 13:27–39, 1980.

[McCoy 85] McCoy, Kathleen Filliben. *Correcting Object-Related Misconceptions.* PhD thesis, Moore School of Electrical Engineering, University of Pennsylvania, Philadelphia, PA, December 1985.

[McDermott 81] McDermott, Drew. Artificial Intelligence Meets Natural Stupidity. In Haugeland, John (editor), *Mind Design*, Chapter 5, pages 129–142. Bradford Books, Montgomery, Vermont, 1981.

[McDonald 79] McDonald, David D. Language Production: The Source of the Dictionary. 1979??

[McDonald 83] McDonald, David D. Natural Language Generation as a Computational Problem. In Brady, M. and Berwick, Robert (editors), *Computational Models of Discourse*, pages 209–265. MIT Press, 1983.

[McDonald 88] McDonald, David D. Modularity in Langauge Generation: Methodological Issues. In *Proceedings of the AAAI-88 Workshop on Text Planning and Generation.* St. Paul, MN, August 1988.

[McKeown 85] McKeown, Kathleen R. *TEXT GENERATION: Using Discourse Strategies and Focus Constraints to Generate Natural Language*. Cambridge University Press, 1985.

[Meteer 87] Meteer, Marie W., McDonald, David D., Anderson, Scott D., Forster, David, Gay, Linda S., Huettner, Alison K., and Sibun, Penelope. *Mumble-86: Design and Implementation.* Technical Report 87-87, Computer and Information Science, University of Massachusetts at Amherst, Amherst, MA, 1987.

[Meteer 89] Meteer, Marie W. *The "Generation Gap": The Problem of Expressibility in Text Planning*. PhD thesis, University of Massachusetts, Amherst, MA, December 1989.

[Miller 83] Miller, Perry L. ATTENDING: Critiquing a Physician's Management Plan. *IEEE Transactions on Pattern Analysis and Machine Intelligence* PAMI-5(5):449–461, September 1983.

[Moser 83] Moser, M. G., Schmolze, J., Israel, D., Vilain, M., and McAllester, D. *NIKL.* Technical Report 5421, BBN Laboratories, Cambridge, MA, 1983.

[Nii 86a] Nii, H. Penny. Blackboard Systems: Blackboard Application Systems, Blackboard Systems from a Knowledge Engineering Perspective. *AI Magazine* 7(3):82–106, 1986.

[Nii 86b] Nii, H. Penny. Blackboard Systems: The Blackboard Model of Problem Solving and the Evolution of Blackboard Architectures. *AI Magazine* 7(2):38–53, 1986.

[Nilsson 80] Nilsson, Nils J. *Principles of Artificial Intelligence*. Tioga Publishing Co, Palo Alto, CA, 1980.

[Nirenburg 88] Nirenburg, Sergei, McCardell, Rita, Nyberg, Eric, Werner, Philip, Huffman, Scott, Kenschaft, Edward, and Nirenburg,

Irene. *DIOGENES-88*. Technical Report CMU-CMT-88-107, Center for Machine Translation at Carnegie-Mellon University, Pittsburgh, PA, June 1988.

[Nogier 89] Nogier, Jean-François. *A Natural Language Production System based on Conceptual Graphs*. Technical Report F.146, Centre Scientifique IBM France, Paris, 1989.

[Quillian 67] Quillian, M. Ross. Word Concepts: A Theory and Simulation of Some Basic Semantic Capabilities. *Behavioral Science* 12:410–430, 1967.

[Quine 53] Quine, Willard Van Orman. *From a Logical Point of View*. Harvard University Press, Cambridge, MA, 1953.

[Quine 60] Quine, Willard Van Orman. *Word and Object*. MIT Press, Cambridge, MA, 1960.

[Quine 70] Quine, Willard Van Orman and Ullian, J. S. *The Web of Belief*. Random House, New York, 1970.

[Reiter 80] Reiter, Ray. A Logic for Default Reasoning. *Artificial Intelligence* 13(1,2):81–132, April 1980.

[Reithinger 90] Reithinger, Norbert. POPEL — A Parallel and Incremental Natural Language Generation System. In Paris, Cecile, Swartout, William, and Mann, William (editors), *Natural Language Generation in Artifical Intelligence and Computational Linguistics*. Kluwer Academic Publishers, 1990.

[Rosch 75] Rosch, Eleanor and Mervis, Carolyn. Family Resemblances: Studies in the Internal Structure of Categories. *Cognitive Psychology* 7:573–605, 1975.

[Rosch 76] Rosch, Eleanor, Mervis, Carolyn, Gray, Wayne, Johnson, David, and Boyes-Braem, Penny. Basic Objects in Natural Categories. *Cognitive Psychology* 8:382–439, 1976.

[Rossi 87] Rossi, Frank. The Curse of Route 55. *The Philadelphia Inquirer Magazine* 38–45, October 4 1987.

[Rubinoff 86] Rubinoff, Robert. Adapting MUMBLE: Experience with Natural Language Generation. In *AAAI-86*, American Association for Artificial Intelligence, pages 1063–1068. Morgan Kauffman, Philadelphia, August 1986.

[Schank 75] Schank, Roger and Riesback, C. (editors). *Conceptual Information Processing*. American Elsevier, 1975.

[Schmolze 83] Schmolze, J. G. and Lipkis, T. A. Classification in the KL-ONE Knowledge Representation System. In *Proc. IJCAI-83*. Karlsruhe, W. Germany, 1983.

[Scott 84] Scott, A. Carlisle, Clancey, William J., Davis, Randall, and Shortliffe, Edward H. Methods for Generating Explanations. In Buchanan, Bruce G. and Shortliffe, Edward H. (editors), *Rule-Based Expert Systems*, Chapter 18, pages 338–362. Addison-Wesley, Menlo Park, CA, 1984. Discusses strategies for constructing explanations in Mycin.

[Sibun 90] Sibun, Penelope. The Local Organization of Text. To appear in AAAI-90, 1990.

[Smadja 90] Smadja, Frank A. and McKeown, Kathleen R. Automatically Extracting and Representing Collocations for Language Generation. In *Proceedings of the 28th Annual Meeting of the ACL*, Association for Computational Linguistics. Pittsburgh, PA, June 1990.

[Smedt 87] Smedt, K. De and Kempen, G. Incremental Sentence Production, Self-Correction and Coordination. In Kempen, Gerard (editor), *Natural Language Generation: New Results in Artificial Intelligence, Psychology, and Linguistics*, pages 365–376. Martinus Nijhoff Publishers, Boston, 1987.

[Sondheimer 86] Sondheimer, Norman K. and Nebel, Bernhard. A Logical-Form and Knowledge-Base Design for Natural Language Generation. In *Proceedings of the Fifth National Conference on Artificial Intelligence*, American Association for Artificial Intelligence, pages 612–618. Philadelphia, August 1986.

[Swartout 83] Swartout, William R. XPLAIN: A System for Creating and Explaining Expert Consulting Programs. *Artificial Intelligence* 21:285–325, 1983.

[Sweetser 83] Sweetser, Eve E. The Definition of Lie: an examination of the folk theories underlying a semantic prototype. In Quinn, N. and Holland, D. (editors), *Princeton Conference on Folk Models*. May 1983.

[Thompson 77] Thompson, Henry. Strategy and Tactics: A Model for Language Production. In *Papers from the Thirteenth Regional Meeting*, Chicago Literary Society, pages 651–668. 1977.