June 1998

Generating Effective Instructions: Knowing When to Stop

Juliet C. Bourne
University of Pennsylvania

Generating Effective Instructions: Knowing When to Stop

Abstract

One aspect of Natural Language generation is describing entities so that they are distinguished from all other entities. Entities include objects, events, actions, and states. Much attention has been paid to objects and the generation of their *referring expressions* (descriptions meant to pick out or refer to an entity). However, a growing area of research is the automated generation of instruction manuals and an important part of generating instructions is distinguishing the actions that are to be carried out from other possible actions. One distinguishing feature is an action's termination, or when the performance of the action is to stop. My dissertation work focuses on generating action descriptions from action information using the SPUD generation algorithm developed here at Penn by Matthew Stone. In my work, I concentrate on the generation of expressions of termination information as part of action descriptions. The problems I address include how termination information is represented in action information and expressed in Natural Language, how to determine when an action description allows the reader to understand how to perform the action correctly, and how to generate the appropriate description of action information.

Comments

University of Pennsylvania Department of Computer and Information Science Technical Report No. MS-CIS-98-35.

GENERATING EFFECTIVE INSTRUCTIONS: KNOWING WHEN TO STOP

MS-CIS-98-35

Juliet C. Bourne

University of Pennsylvania
School of Engineering and Applied Science
Computer and Information Science Department

Philadelphia, PA 19104-6389

1998

GENERATING EFFECTIVE INSTRUCTIONS: KNOWING WHEN TO STOP

by

JULIET C. BOURNE

A DISSERTATION PROPOSAL

in

COMPUTER AND INFORMATION SCIENCE

at the

UNIVERSITY OF PENNSYLVANIA

June 29, 1998

Acknowledgments

First and foremost, I must thank my advisor, Dr. Bonnie Webber, for all of her guidance and patience in my quest for a dissertation topic and focus. It was her idea for me to use SPUD to implement my dissertation. So, secondly, I must thank Matthew Stone for writing SPUD and being a big help in general to me over the last several years. Many thanks as well to Dr. Mark Steedman, Dr. Norm Badler, and Dr. Martha Palmer for all of their help and encouragement throughout my graduate career. Finally, I'd like to thank everyone in the various groups I've participated in (DTOG, TOG/AMI, PAR, SPUD) for their helpful discussions out of which this dissertation proposal began to take shape.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

One aspect of Natural Language generation (hereafter simply referred to as generation) is describing entities so that they are distinguished from all other entities. Entities include objects, events, actions, and states. However, much attention has been paid to objects and the generation of their *referring expressions* (descriptions meant to pick out or refer to an entity). This is taken to involve iteratively including a property which distinguishes an object from the greatest number of remaining *distractors* (other objects that it could be confused with) until the description picks out only the intended object. For instance, if there are two blocks, one red and one blue, the referring expression for either of the blocks will have to include an expression of its color property to distinguish it from the other block. Generating referring expressions for other types of entities has not been explored as much as for objects. As such, the question of how to distinguish entities other than objects has not been addressed.

A growing area of research is the automated generation of instruction manuals. An important part of generating instructions is distinguishing the actions that are to be carried out from other possible actions. Thus, work needs to be done on the generation of referring expressions for actions, with a focus on representing action information, distinguishing one action from another, and building a generation system that can consider information provided at the clause level and can use multiple clauses to describe actions. Given an action representation which holds information about actions and a generation system which is able to produce multi-clausal referring expressions, what is needed is knowledge about how instances of the action representation are expressed in Natural Language text.

Such knowledge can be acquired through the analysis of naturally-occurring texts in terms of the linguistic constructions used to describe actions. Linguistic constructions include optional verb arguments (e.g. "Rotate *90 degrees*"), path prepositional phrases (e.g. "Walk *to the store*"), *until* clauses (e.g. "Turn lid *until it is loose*"), *etc.* All are used to describe necessary pieces of information about actions. An interesting set of linguistic constructions are those which describe when the performance of an action is to stop. All of the linguistic constructions mentioned above can be used to describe such *termination information*. Describing such information is an important part of generating *effective* instructions that will be sufficient for the hearer to understand and carry out the action correctly. If termination information is missing, then an instruction may be inadequate unless it is known that the hearer of the instruction can infer the correct termination information. Since termination information can be conveyed both through *termination condition* phrases and implicitly through phrases expressing other parts of the action instance, knowing how information is expressed is an essential part of generating effective instructions.

In my work, I concentrate on the generation of expressions of termination information as part of action descriptions. The problems I address include how termination information is represented, how to determine when an action description is adequate for the correct performance of the action, and how to generate the appropriate description for an action instance. In the next chapter, I discuss these problems in more detail and outline the work addressing these problems.

In Chapter 3, I discuss related work in the areas of action representation, lexical choice (how to choose between different linguistic constructions), and generation. In Chapter 4, I present a corpus analysis of naturally-occurring instructions, focusing on how termination information is expressed. In Chapter 5, I detail the implementation which addresses the problems mentioned above and, in Chapter 6, I propose the work to complete my dissertation.

Chapter 2

Effective Action Descriptions with Termination Information

An important goal of the generation of Natural Language instructions is to describe the actions fully and accurately so that they can be carried out correctly. This goal is particularly important to the generation of written instructions where the "speaker" (i.e. author) and the "hearer" (i.e. reader) are separated spatially and temporally. In the case of instruction manuals, the hearer does not have the opportunity to ask questions to clarify the action to be performed and the speaker likewise does not get any feedback from the hearer about the success of the instructions. Therefore, attention must be paid to the *effectiveness* of the instructions generated to be sure that they can be carried out correctly. Attention must also be paid to the efficiency or conciseness of the instructions. That is, all the necessary information should be included in an efficient in order to avoid confusion caused by extra information. Understanding how information about an action is expressed, which ways of expressing information are used for which purposes, *etc.*, is essential to generating instructions that describe actions both effectively and efficiently.

Expressing action information has been explored in several ways by other researchers. For instance, the issue of *lexical choice*, choosing the words (especially the verb) to describe an action has been addressed by a number of researchers including [Elhadad *et al.*, 1997]. Generation of *referring expressions*, i.e. descriptions of entities (mainly physical objects) at various points in a set of instructions, has been looked at by Dale ([Dale, 1992]), among others. Expressing the *purpose* of an action, i.e. "Do x to do y," has been examined by several researchers, including Di Eugenio ([Di Eugenio, 1993]) and Vander Linden ([Vander Linden, 1994]). As yet unexplored is the issue of expressing an action's *termination information*, or when to stop doing an action.

Before discussing how termination information is expressed, I should clarify the terms which I will be using to refer to actions and their descriptions in Natural Language:

Action refers to a whole action class, a set of actions with the same defining or main components. However, sometimes I will also use this to refer to more specific action classes incorporating generic objects. The context in which the term is used should distinguish whether I am referring to a general or specific action class, if it makes a difference.

Action instance refers to a particular action in the world, complete with the particular entities involved.

Action description refers to all of the linguistic expressions used to describe a particular action instance. These expressions do not necessarily have to be contiguous in the actual text; they can appear across multiple sentences.

Instruction refers to a single sentence with an imperative main clause which describes an action(s) the hearer is to perform. It need not contain a complete action description.

Instruction step refers to a set of instructions describing a single step of a task. The task step could involve multiple actions, and therefore could require multiple instructions to describe.

In terms of information about an action, an action description must convey not only information about the main components of the action (e.g. movement, change of state, *etc.*), its participants, its manner, *etc.*, but also the information about when to stop doing the action. Actions have different types of aspectual (temporal) structure and the type of an action can provide termination information. For instance, *culmination* (which is termination plus a change of state) can be inherent in some actions, such as *remove* and *break*. For these actions, just giving the main component of the action, i.e. the change of state, also provides the termination information. However, some actions, such as *turn*, do not have inherent culmination or termination information. Such actions, called *activities*, need to appear along with termination information to form an effective instruction step. Termination information can be explicit in the instructions or implicit in the interaction of the activity with other actions in the instruction step. However it is provided, termination information is necessary for the performance of actions that do not have an inherent end.

In Natural Language, information about an action is *realized* by many different linguistic sources. For example, the main component of the action is usually expressed through the verb. Verbs expressing actions reflect the different aspectual types of actions mentioned above. For instance, the verb *remove* is considered an *accomplishment* verb, which means, among other things, that it has inherent culmination. However, the type (and thus termination) of an action is determined by *all* of its information. Thus, linguistic expressions for other parts of the action, including the arguments to the verb (e.g. path information), and additional phrases such as purpose clauses and temporal clauses (e.g. *until*), contribute to the type of the action as well. Interactions among these linguistic expressions also affect the type of the action expressed and must be considered when deciding how to describe an action.

The variety of linguistic expressions of termination information provides several choices for expressing the termination of an action, each with different purposes and different implications in different contexts. Thus, it is important to characterize the choices made in naturally-occurring instructions and to determine how to make the same choices. In order to do this, a corpus of naturally-occurring instructions is obtained and analyzed. The results can then be used in a generation system to generate similar instructions. In the next section, I describe the characteristics of the instructions which I examine in terms of expressing action termination. These characteristics show the genre of instructions that I aim to generate.

2.1 Domain characteristics

The domain of the naturally-occurring data that I have examined is simple step-by-step maintenance instructions. The corpus consists of parts of a "do-it-yourself" book and a collection of *technical orders* (military instructions) for the maintenance of F-16 aircraft. I have only looked at the numbered step-by-step parts of the texts rather than the general discussion in the former and the notes, cautions, and warnings in the latter. In the corpus study done by [Hartley and Paris, 1996], step-by-step instructions are recognized as a sub-genre of instructions. Their analysis shows that step-by-step instructions have linguistic features which distinguish them from the other sub-genres (e.g. reference and tutorial texts). Thus, focusing on the step-by-step parts of corpora is linguistically-motivated and provides a manageable collection of contexts and linguistic features to study.

The actions in the domain include kinematic actions, that is, actions viewed as involving motion over time. Having kinematic actions, as opposed to state-space or change of state actions, means that some actions do not have intrinsic ends and thus need termination information. State-space actions, actions that are viewed simply in terms of changes in state (e.g. *switch*), also occur in the domain but are not a focus of the current study.

As I show in Chapter 4, termination information is usually explicit in each instruction (especially in the technical orders), either because the action has an inherent termination or because an expression of termination is used. However, termination is sometimes left to be inferred from world knowledge, as well as other knowledge the hearer is assumed to have about the domain. An action's termination can be inferred from its objects as well as from other actions, including the interactions with surrounding actions and with the overarching goal of the instructions. It is important for a generation system to be able to use not only information about the actions but also information about the domain, the hearer, and the discourse. With this information available, a generation system can generate appropriate and effective action descriptions.

2.2 Dissertation goals

In order to generate instructions which are effective action descriptions, I must achieve three interrelated goals in my dissertation work: representing actions, analyzing particular constructions used for expressing action termination, and generating instructions.

The action representation must support the variety of information about actions, including termination information. The representation should record the relationships between actions, such as sub-steps and purposes, which can be sources of termination information. It should be as *language-neutral* as possible; that is, it should not be structured in a certain way just for linguistic concerns. The action representation must be suitable for generating Natural Language from and yet not be tied to any particular language or linguistic theory; it must represent the correct level of detail and abstraction.

The constructions used for expressing termination, along with the semantic and pragmatic contexts in which they appear, must be characterized. To base this on actual language use, I coded the instructions in the corpora for the types of actions that occur, the constructions which appear in the action descriptions, the sources of termination information, and relevant world knowledge. From the coding, I analyze the instructions and draw preliminary conclusions about the use of constructions for expressing termination information. The characterization of the constructions must be compatible with and expressible in the generation system, which must be able to support the distinctions in meaning and function of the constructions.

The generation system must choose between the available constructions and construct an action description based on the same semantic and pragmatic contexts determined in the corpus analysis. It must also be able to use the option of spreading an action description over multiple clauses and sentences, since information about an action, especially termination information, can appear in multiple clauses and sentences. The generation system must be capable of interpreting the action representation in order to determine the best way to express the action information.

2.3 Overview of work

Each of the goals in the previous section entails its own set of tasks. I start with the action representation since it sets the foundation for the constructions and the generation. Next, I determine which constructions to look at and what contexts are relevant. Finally, I briefly describe the work needed for the actual generation of instructions. Each of these will be discussed in more detail in Chapter 5, which describes the implemented work.

2.3.1 Action representation

I use a *Parameterized Action Representation* (PAR) developed here at Penn as an intermediate representation that can support both the animation and Natural Language description of actions [Badler et al, 1997; Badler et al, 1998]. As an intermediate representation, PAR must represent actions at various levels of abstraction, from general action classes to specific action instances (i.e., sets of action performances). A PAR instance for an action consists of the various features of the action. These include the main semantic components of the action which identify its general action class. Other features include specific information about the action which distinguishes it from others in its action class. For the purposes of my work, PAR must be able to represent the following pieces of information, each of which can provide information relevant to the termination of an action:

- **core semantics** — the state-change, motion, and/or forces of the action
- **participants** — the agent of the action as well as the other entities involved in the action
- **direction/path** — for actions involving motion, the direction or path of the motion
- **purpose** — the purpose for which the action is done: to achieve a particular state, to generate another action, and/or to enable the next action
- **manner** — the way in which an action is performed (e.g. *quickly, carefully*)
- **termination** — explicit termination conditions (states of the world or events) unrelated to other aspects of the action
- **duration** — explicit timing of the action (e.g. *for 6 seconds*) or iteration (e.g. *between 5 and 6 times*); involves measured units (as opposed to general conditions)

Each of these slots can have counterpart, or realization, in an action description. For instance, the core semantics is usually realized as the verb, the participants as the subject and verb arguments, the path as a prepositional phrase, *etc.*, as will be shown in the following section.

For actions that are part of an instruction step or that contain sub-steps of their own, the action representation also needs to include information about the other actions in the instruction step. For instance, if an action has sub-steps, these should be given in the PAR for the action. Likewise, a sub-step action should have pointers to the action which it is a sub-step of as well as the other sub-steps. Therefore, the following slots are included in the PAR:

- **sub-steps** — elaboration of how to accomplish the action
- **previous-step** — link to a previous action
- **concurrent-step** — link to a concurrent action
- **next-step** — link to a following action
- **super-step** — link back to the parent action of which the action is a sub-step

These, too, can have linguistics counterparts. For instance, a concurrent action can be conveyed using a *while* clause. In the next section, I discuss the ways in which information in the PAR can be realized in Natural Language instructions.

2.3.2 Constructions

As noted above, termination information has many sources in an action description. These sources fall into the following groups:

Predicate-argument structure is the verb and its required arguments, denoting some of the participants in the action. The verb alone can have an inherent termination, as in

(1) Remove the access panel.

The verb with a certain argument type can give a termination, as in

(2) Cut the wire.

The arguments alone can provide a termination, as in

(3) Pour one cup of water.

Optional arguments of the verb, such as prepositional phrases for paths or locations, adverbial phrases for direction or manner, *etc.*, can also give termination information. For example:

(4) Rotate aerial refueling control *to full counterclockwise (off) position.* [USAF, 1988]

Without the prepositional phrase, the action description, i.e. "Rotate the aerial refueling control," does not express when to stop.

Additional clauses such as *until* and *when* clauses, purpose clauses (including purposive *and* clauses), etc., can provide the termination of an action. For instance ([USAF, 1988]):

(5) a. Depress system A reservoir dump valve *until accumulator gage[sic] indicates precharge pressure.*

b. Slide valve aft *and remove.*

c. Depress bleed valve sufficiently *to obtain stream of fluid flow.*

Interaction between an action and other actions, i.e. whether a generation or enablement relationship exists between two actions, whether one action is done for the purpose of another, whether the start of the next action implies the termination of the previous one, *etc.*, can give the termination of an action. Such sources of termination information all seem to require inference on the part of the hearer.

For any action, termination information can be combined from multiple sources as seen in this example from [USAF, 1988]:

(6) NOTE: *To remove actuator*, it will be necessary to lift actuator *slightly* and rotate actuator *90 degrees clockwise until sufficient clearance is obtained to disengage actuator splines.*[1]

[1]This example is not in the step-by-step subset that I use. It is shown as a good example of multiple sources and it could be paraphrased in the step-by-step style as

In my proposal, I focus on the termination information found in optional arguments and additional clauses, as they provide a wide range of constructions for an action description. However, in the full dissertation, termination information from the interaction of actions will be addressed as well.

To approach the question of which constructions appear in which contexts, I have developed a set of examples (see Figure 2.1). While these are constructed examples and do not cover the complete range of constructions, they provide minimal variation in the actions and should allow me to identify the contexts in which each construction is appropriate. In these examples, the actions to be performed (and thus their descriptions) are variations of *turning the knob*. The other action (or event) in the examples which affects the action performance is *opening the door* (or *the door opening*). Termination information is given by different constructions. For instance, in Example 8b, the purpose clause (*to open the door*) provides the termination for turning the knob, i.e. when the action of opening the door is done. In Example 9a, the prepositional phrase indicates the termination, i.e. when the knob reaches the "open" position. The *until* clause in Example 15a provides the termination, namely when the door opens. Choosing one of these constructions instead of another depends on the context.

The contexts in which these constructions are used involve both hearer and world models. Hearer models can differ in what the hearer is assumed to know about the world and what discourse has come before the current instruction. World models can differ in terms of object properties and states as well as relationships among objects and actions. Figure 2.2 shows the relevant *temporal* relationships that are possible between two actions where one action defines the endpoint of the other. Typical *causal* relationships between actions in instructions are those of *generation* and *enablement* [Di Eugenio, 1993]. An action *generates* another if doing the *generating* action (α) means that you also do the *generated* action (β). An action *enables* another if, after doing the *enabling* action (α), the *enabled* action (β) can be done. Notice that a generation relation means that the actions are coextensive and that an enablement relation means that the actions are sequential.

(7) *To remove actuator*, lift actuator *slightly* and rotate actuator *90 degrees clockwise until sufficient clearance is obtained to disengage actuator splines.*

(8) a. Turn the knob. Open the door.
b. Turn the knob to open the door.
c. Turn the knob and open the door.

(9) a. Turn the knob {90 degrees/to the "open" position}. Open the door.
b. Turn the knob {90 degrees/to the "open" position} to open the door.
c. Turn the knob {90 degrees/to the "open" position} and open the door.

(10) a. Turn the knob until you open the door.
b. Turn the knob so that you open the door.

(11) a. Turn the knob until you can open the door.
b. Turn the knob so that you can open the door.

(12) a. Turn the knob, opening the door.
b. Turn the knob {90 degrees/to the "open" position}, opening the door.

(13) a. Open the door by turning the knob.
b. Open the door by turning the knob {90 degrees/to the "open" position}.

(14) a. Turn the knob. The door will open.
b. Turn the knob {90 degrees/to the "open" position}. The door will open.

(15) a. Turn the knob until the door opens.
b. Turn the knob so that the door opens.

Figure 2.1: Set of "minimal pairs"

```
Coextensive           |←— α —→|
                      |←— β —→|

Delayed               |←— α —→|
                        |← β →|

Non-causation           |← α →|
                      |←— β —→|

Sequential        |← α →|
                        |← β →|
```

Figure 2.2: Temporal relationships between two actions where one (β) defines the endpoint of the other (α)

Of course, relationships which are not generation (in the strict sense) or enablement are possible, as shown in the "delayed" relationship. Figure 2.3 shows the possible world models (contexts) in which these relations could exist between the actions in the minimal pairs, namely *turning the knob* (α) and *opening the door* (β).[2] The appropriateness of the constructions can be examined in each of these world models (leaving aside any variations in the hearer model). Based mainly on intuition at this point, a preliminary characterization of which constructions are appropriate in which contexts is shown in Figure 2.4.

An important question that has yet to be addressed is what implications accompany each construction. Using a particular construction will cause the hearer to make some assumptions about the world and the action to be performed. The preliminary characterization shown in Figure 2.4 proposes the implications associated with each construction by noting world models in which constructions are felicitous and would not cause the hearer to make incorrect inferences. A related question is that, when termination information for an action is not explicit in the instructions, is an *expectation* raised that the termination is assumed to be known, inferable, or otherwise defaulted to by the hearer? The first question is addressed in Chapter 5, when the contexts appropriate for a few constructions are proposed. However, full answers to both of these questions must wait for the completed dissertation.

[2]I have not yet related this model of causation to any existing taxonomies. However, it should be compatible and the full dissertation will explore its relationship with other causal models.

1. Coextensive generation: Turning the knob opens the door, e.g. the knob controls the door hinges directly.
2. Delayed "generative" causation: Turning the knob a certain amount causes a physical link between knob and door hinges to be made. Turning the knob further opens the door.
3. Sequential "generative" causation: When the knob is turned sufficiently, the door opens automatically.
4. Coextensive "enabling" causation: Opening the door can be begin as soon as, and can continue as long as, the knob is being turned.
5. Delayed "enabling" causation: After a certain amount of turning, opening the door can begin and can continue as long as the turning continues.
6. Sequential enablement: Turning unlatches the door, allowing the opening of the door to begin.
7. Arbitrary causation: The door opens arbitrarily when the knob is being turned.
8. Non-causation: Turning the knob has no direct causal relationship to opening the door.

Figure 2.3: Possible world models for actions in the set of minimal pairs

World Model	Sentences
1 (Coextensive generation)	8(b-c), 9(b-c), 10, 12, 13
2 (Delayed "generative" causation)	8(b-c), 9(b-c), 10
3 (Sequential "generative" causation)	8(b-c), 9(b-c), 14(b)
4 (Coextensive "enabling" causation)	11(b)
5 (Delayed "enabling" causation)	11(b)
6 (Sequential enablement)	11
7 (Arbitrary causation)	14(a), 15
8 (Non-causation)	8(a), 9(a), 15(a)

Figure 2.4: Constructions appropriate in each world model

2.3.3 Generation

Generation is done using the SPUD (*Sentence Planning Using Descriptions*) Natural Language generator [Stone and Doran, 1997; Stone and Webber, 1998], which is described in detail in Section 5.2. SPUD forms descriptions of actions, events, states, and objects, by choosing lexical items from its Lexicalized Tree Adjoining Grammar which serve its communicative goals best. By virtue of being a lexicalized grammar, lexical items correspond to constructions (i.e., syntactic tree fragments). Lexical items are annotated with semantic and pragmatic information that SPUD can match against the information, e.g. about an action, it is trying to convey. Using this framework and extending it to handle multi-clausal sentences (and eventually multi-sentence discourse), instructions are generated from the proposed action representation and the semantic and pragmatic contexts determined empirically. (The action representation is outlined in Section 2.3.1 and described in Section 5.1. The semantic and pragmatic contexts are outlined in Section 2.3.2 and described in Chapter 4.) SPUD was designed to generate simple sentences consisting of predicate-argument structure and optional arguments. It is straightforward to make it generate multi-clausal sentences. Extending SPUD to generate a multi-sentence discourse, however, will take a bit more work and will be addressed in the completed dissertation.

Encoding constructions for SPUD consists of creating lexical items which specify their syntax as well as the semantic and pragmatic contexts in which they are used. An additional communicative goal is given to SPUD as part of a generation task in order to control when termination information must be expressed. To determine if a particular action description has termination information, SPUD needs rules that it can use to check whether the description provides termination information. Given all of this information, SPUD can be told to generate a description of a particular action in the form of an instruction. SPUD uses the given semantic and pragmatic context to determine the best description of the action, making sure that is adequate for the performance of the action. SPUD and the encoded constructions will be described in more detail in Chapter 5. In the next two chapters, I discuss some background material to provide a framework for my dissertation work and present a corpus analysis to motivate my encoding of constructions which express termination information.

Chapter 3

Background and Related Work

In this chapter, I review background and related work for the three goals mentioned in the previous chapter: action representation, linguistic constructions, and Natural Language generation. I draw on previous work to achieve each of the goals.

3.1 Action Ontology and Representation

As far back as Aristotle, philosophers and linguists have pondered the types of situations (events, actions, and states) evoked in language. Vendler [Vendler, 1967] proposed a typology of situations, distinguishing between *accomplishments*, *achievements*, *activities*, and *states*, each of which has its own temporal structure and properties. An activity, such as *pushing a cart*, has "no set terminal point," while an accomplishment, such as *drawing a circle*, has "a 'climax', which has to be reached if the action is to be what it is claimed to be" [Vendler, 1967, p.100]. Achievements, such as *reaching the top*, "occur at a single moment", whereas states, such as *loving*, "last for a period of time" [Vendler, 1967, p. 103]. Mourelatos [Mourelatos, 1981] proposed a similar typology, but he collapsed accomplishments and achievements together as *events* (see Figure 3.1, adapted from [Passonneau, 1987, Figure 1]). Moens and Steedman [Moens and Steedman, 1987] follow in the same vein, classifying situations into states and events. However, they make a finer and more systematic distinction between the kinds of events (and, therefore, actions). They are characterized along two dimensions — the extension of an event or action in time, or alternatively its ability to be decomposed into sub-events or sub-actions, and the existence

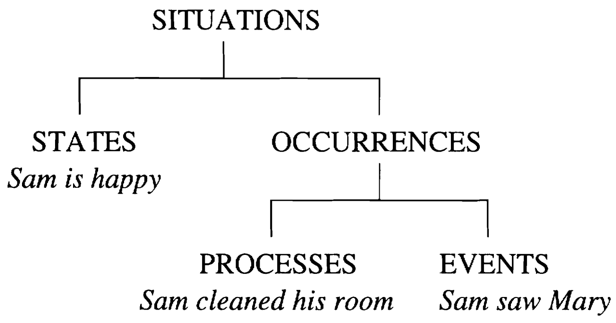


Figure 3.1: Mourelatos' typology of situations

	EVENTS		STATES
	atomic	extended	
+conseq (telic)	Harry broke the window (**achievement**)	Sue built a sandcastle (**accomplishment**)	Tom is in the kitchen
-conseq (atelic)	Sandra hiccupped (**point**)	Max worked in the garden (**activity**)	

Figure 3.2: Moens and Steedman's classification of situations along two dimensions

of characteristic consequences associated with the event or action (see Figure 3.2, adapted from [Moens and Steedman, 1987, Figure 1]).

While all four types of events and actions shown in Figure 3.2 exist, actions that appear in instructional texts tend to be either achievements or accomplishments. Both of these types have consequences, or effects on the world, which is the general point in maintenance instructions. Another feature they have in common, related to the fact that they have consequences, is that they have defined endpoints. That is, achievements and accomplishments, as part of their meaning, include when to stop doing the actions. This inherent termination can be seen in the tripartite representation of actions that [Moens and Steedman, 1987] propose. It allows actions to have a preparatory process, a culmination point, and a consequent state (see Figure 3.3). The culmination point, right before the consequent state begins, is the endpoint of both achievements and accomplishments. The difference between the two types is that an achievement does not have a characteristic preparatory process leading up to the culmination. However, they are interchangeable

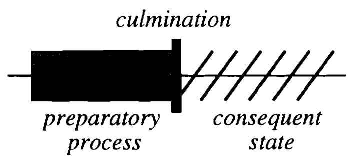


Figure 3.3: Moens and Steedman's tripartite structure of events

by stripping away or adding the preparatory process, depending on the importance to be placed by the hearer on the preparatory process.

An important part of understanding instructions is understanding how the different actions in an instruction step are related temporally. While instructions are usually given in the order in which they are to be done, it is still necessary to express more complex temporal relationships, such as overlap or concurrency. Allen [Allen, 1983; Allen, 1984] has identified a set of thirteen temporal relations between the intervals (spans of time) over which situations hold or take place, shown in Figure 3.4 (adapted from [How, 1993, Figure 2.5]). As shown in Figure 2.2, however, only a few of the interval relationships may be needed for representing the temporal relationships involved in termination information.

In terms of representing action information, work in knowledge representation provides the basis for action representations. First-order logic, description logics, and feature structures are some of the representations that have been used. Steedman [Steedman, 1997] has proposed encoding the semantics of events/actions in a dynamic semantics formalism, an extension of first-order logic. The generation system COMET [McKeown *et al.*, 1990] uses Functional Unification Formalism, an extension of functional unification grammar (related to feature structures) to represent logical-form semantics. Description logic knowledge representations, i.e. combinations of feature structures and logic machinery, include CLASSIC (used by [Di Eugenio, 1993]) and LOOM (used by [Rösner and Stede, 1994], among others). Feature structures are the simplest and most common way of representing actions. Feature structures contain attribute-value pairs (e.g. `<agent,you>`) where the value is a simple token or another feature structure. The action representations used by [Dale, 1992] and [Kalita, 1990] are feature structures. As feature structures appear sufficient for my action representation purposes, I will not go into the details of the others.

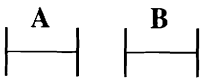
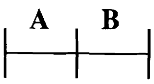
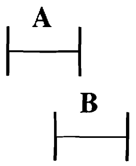
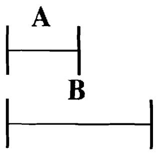
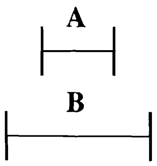
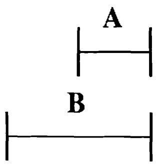
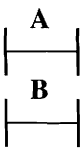
Relation		**Inverse Relation**
A *precedes* **B**		**B** *follows* **A**
A *meets* **B**		**B** *met-by* **A**
A *overlaps* **B**		**B** *overlapped-by* **A**
A *starts* **B**		**B** *started-by* **A**
A *during* **B**		**B** *contains* **A**
A *finishes* **B**		**B** *finished-by* **A**
A *equals* **B**		

Figure 3.4: Allen's thirteen relations between intervals

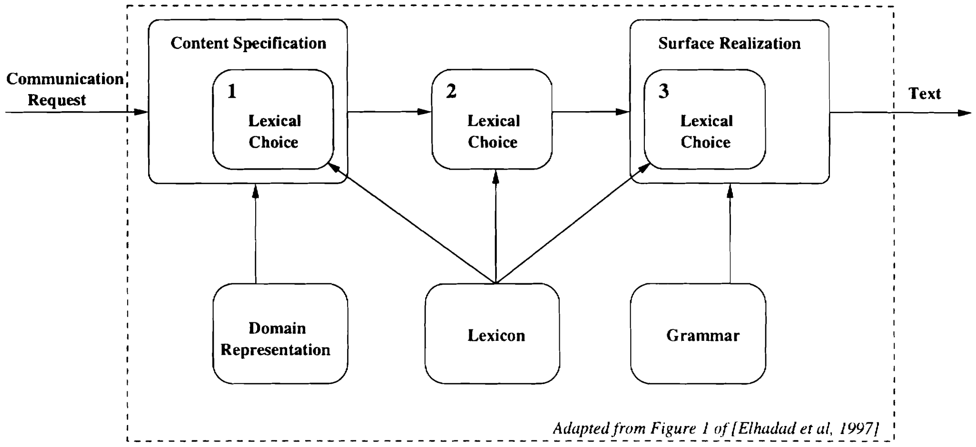


Figure 3.5: Lexical choice in a generation system

3.2 Lexical Choice and Linguistic Constructions

"The problem of determining what words to use for the concepts in the domain representation is termed **lexical choice**. In an effort to make domain representations independent of language, there may be a variety of different words that can be used to express any concept in the domain, and a language generator must choose which one is most appropriate in the current context." [Elhadad *et al.*, 1997, p.195]

The choice of words and linguistic constructions anchors the generation of instructions. Words and constructions need to be chosen based upon their meaning and implications in expressing action information. This *lexical choice* relies on analyses of words and constructions in natural texts. The choice of a particular word or construction to express a piece of information depends on many contextual factors. Contextual factors include previous syntactic and lexical choices, since they can affect the choices that can be made subsequently. The structure of the domain, e.g. its objects and relations, also affects lexical choice as the domain may force or preclude particular choices. What is commonly thought of as "the context," that is, information about the speaker, the hearer, and the previous discourse, also contributes additional contextual factors. All of these contextual factors *constrain* the choice of lexical items and their syntactic constructions, as described by [Elhadad *et al.*, 1997].

The development of a lexical choice algorithm begins with determining the correlations between the contextual factors and the linguistic features of words and constructions, usually through a corpus analysis as demonstrated by [Hartley and Paris, 1996], among others. Once the contextual factors and the ways in which they constrain the range of linguistic features have been determined, several methods can be used to perform lexical choice. Since generation systems depend on lexical choice to determine the most appropriate way to express information, lexical choice algorithms define one of the key differences between generation systems. Lexical choice methods differ in a number of ways, including the constraints which they consider, how those constraints are represented, the location of lexical choice in the system architecture, and what the lexical choice algorithm receives as input. The constraints used by a system determine its ability to choose between similar words and constructions. If the constraints are general, then the lexical choice algorithm will be able to make only coarse-grained decisions. The representation of the constraints, e.g. as rules or heuristics, affects the location of lexical choice, which in turn indicates the focus and flexibility of a system. For instance, if lexical choice occurs early (at location 1 in Figure 3.5), there may be a one-to-one mapping of domain concepts and lexical items, resulting in less flexibility of expression. What input is provided to the lexical choice algorithm, i.e. the information on which it bases its decisions, also determines the quality of the decisions made. Not enough information or the wrong kind of information can result in poor lexical choice. All of these factors determine how well a lexical choice algorithm will be able to choose appropriate words or linguistic constructions.

While the number of different linguistic constructions is considerable, those involving expressions of purpose have been the focus of much attention, especially in terms of their use in instructions. Since the performance of an action can change depending on the purpose for which it is done, conveying an action's purpose is important in instructions. Purpose can modify many aspects of the performance of an action, including its termination and manner. My interest in purpose constructions stems from their use to convey termination information for actions[1] and understanding how to express purpose is necessary in general in order to produce natural and effective instructions. Thus, I briefly review some relevant research which explores how expressions of purpose are related to the

[1]As Chapter 4 shows, nearly a third of the purpose constructions in the corpora provide termination information.

semantics of actions.

[Di Eugenio, 1993; Di Eugenio and Webber, 1996] look at purpose clauses with respect to inferences that must be made to interpret instructions. They consider how actions are related as well as the assumptions made to accommodate such relations. While they deal with interpretation rather than generation, their analysis and conclusions are valuable and can be applied to generation.

[Kosseim and Lapalme, 1995] develop heuristics for determining how to express effects and guidances. Effects are essentially generation relationships between actions and other actions or events. Guidances are conditional generation relationships between actions, i.e. the generated action will only take place if certain conditions hold. This work explores how to realize these "semantic carriers" (*rhetorical relations*) as purpose clauses, means ("by") clauses, or statements of result. (See the next section for more about this work.)

[Vander Linden and Martin, 1995] perform a corpus analysis to determine correlations between contextual factors (e.g., semantics, discourse, and the hearer model) and the ways in which purpose is expressed. The decisions that are made about the purpose expression include: its slot (position with respect to main action), its form (grammatical category), its linker or cue words (fixed lexical items in constructions), and how clauses are combined. (This work is also described in the next section.)

To some extent, I have incorporated the work done on purpose constructions into my implementation, especially in terms of the types of purpose relationships between actions. In the complete dissertation, more of this work will be incorporated with regards to particular choices between purpose constructions encoded in the generation system.

3.3 Natural Language Generation

A generation system should take (or determine) communicative goals and produce text which satisfy them. Generation systems must be given (or plan) plan the content to be conveyed as well as perform lexical choice and surface realization (refer back to Figure 3.5

for an overview of generation system architecture). In order to carry out the transformation of goals into text, systems need a representation of the domain (e.g. concepts, objects, relations, *etc.*), a lexicon supplying words and their meanings, and a grammar providing ways of combining words into sentences (and possibly sentences into discourse). Every system varies in their methods of content and text planning, lexical choice, and surface realization, and each uses different domain representations as well as different forms of lexicons and grammars. In my work, I assume that, by the time a system is generating a single instruction step, no further content or text structure planning is needed beyond choosing to use multi-clausal sentences or multiple sentences. So, leaving aside content and text structure planning, I focus my discussion of generation systems on their domain and lexical representations, their lexical choice method and other aspects of their generation algorithms, and, when possible, the quality of the texts produced. Each of these issues are addressed below, first in general and then briefly in some of the specifics of actual systems.

Domain and lexical representations encode information about the domain, the lexicon, and the connections between the two. Similar to action representation discussed in Section 3.1, domain representation can be done in several formalisms, such as first-order logic, description logics, and feature structures. A key issue in domain representation is whether it is independent of purely linguistic considerations. A domain representation is *language-neutral* if it does not contain elements or structures that are required mainly by any particular Natural Language. A related issue is the mapping of concepts in the domain to words in the lexicon. A one-to-one mapping between domain concepts and lexical items reduces the flexibility of generation. If the connection between concepts and words is many-to-many, there can be many different ways of relating the same concept in different contexts. One final issue about lexical representation is the inclusion of context in the representation of lexical items. That is, not only is the meaning of a word or construction represented, but also the context in which it has that meaning. This issue is important in terms of how lexical choice is done.

Lexical choice and realization algorithms are the tactical ("how to say it" as opposed to "what to say") components of a generation system — they perform *linguistic* *realization*, the transformation of semantics into words and constructions. The variations in lexical choice algorithms were discussed in Section 3.2. While the lexical choice algorithm is a defining difference between generation systems, several other related differences exist. For instance, if a generation system uses a *lexicalized grammar*, one in which every piece of the grammar is associated with at least one word, then lexical choice performs the surface realization as well. Without a lexicalized grammar, a separate surface realization phase is needed to combine the chosen words into legal syntactic structures. The choice of a lexicalized or non-lexicalized grammar obviously affects the lexical choice algorithm, dictating whether lexical choice will choose words alone or words along with the constructions which they anchor. One final issue is whether backtracking, undoing a previous choice or decision, is used when legal sentences cannot be generated at first. Backtracking can be used within the lexical choice algorithm itself, usually when a lexicalized grammar is used, or during the surface realization phase, at which point the lexical choice phase must be redone. Finding a mapping from the semantics to a surface realization is a search problem and differences in search algorithms are also applicable to lexical choice and generation algorithms.

Assuring the sensitivity, efficiency, and effectiveness of generated texts is essential for a successful generation system. Texts need to be *sensitive* to what the hearer knows. Different texts conveying the same information should be generated for hearers with different knowledge, tasks, *etc.* This could include making sure to use only words which the hearer knows or actions which the hearer is able to perform. Texts also need to be *efficient* by avoiding redundancy. In order to produce efficient texts, the generation system needs to be able to check which of the communicative goals have been already achieved by the text at various points in the generation process. Among other benefits, this allows constructions to contribute to more than one communicative goal. Finally, texts need to be *effective*. They need to identify referents (objects, states/conditions, events, and actions) unambiguously and sufficiently to serve the communicative purpose (in the case of instructions, enabling the correct performance of an action). Systems need to verify that the hearer will be able to determine a text's referents in order to generate effective texts.

The generation systems briefly described below address the above issues to a greater or lesser degree.

COMET [McKeown *et al.*, 1990] uses Functional Unification Formalism (FUF), a declarative and uniform representation, for domain and lexicon representation, and unification for lexical choice and generation. Unification incrementally enriches the logical form determined by content planning until all aspects of the utterance are considered, lending COMET the ability to produce efficient texts.

TECHDOC [Rösner and Stede, 1994] uses the description logic LOOM as the representation for text structuring information as well as domain knowledge. It generates descriptions and instructions needed for maintenance activities. Penman, a systemic-functional sentence-level generator, is used for lexical choice and sentence planning. The system is sensitive to the state of the world, i.e. only relevant information is provided, and it utilizes a language-neutral domain representation.

IDAS [Reiter *et al.*, 1995] uses a hybrid action representation in the form of canned text with embedded knowledge-base references and case frames (roughly, predicate-argument structures) with textual case fillers. Such a representation is not as flexible as other approaches which do not use canned text. A description logic representation is used for all information, including the grammar and lexicon. Lexical choice follows [Reiter, 1991] and the generation is sensitive to the user model, which is provided as part of the input.

Scott and de Souza [Scott and de Souza, 1990] rely on Rhetorical Structure Theory (RST) to structure sentences and overall text. (RST is a method for describing relationships, i.e. *rhetorical relations*, between spans of text.) They promote the use of accurate and unambiguous markers (e.g., cue words) of rhetorical relations to make sure the intended message gets across to the hearer despite the lack of a good hearer model. They use heuristics to implement lexical choice with respect to choosing the most appropriate rhetorical relation to lexicalize for the given semantic content.

Kosseim and Lapalme [Kosseim and Lapalme, 1995] address a restricted form of lexical choice, that of choosing which rhetorical relations to use when mapping the semantic representation to a rhetorical structure. Thus, they focus on the choice of linguistic constructions (e.g. those expressing rhetorical relations, such as means or purpose) rather than on the choice of individual words (except those associated with the linguistic constructions). They use heuristics, derived from a corpus analysis, to determine the realization of two *semantic carriers*, effects and guidances, as rhetorical relations.

IMAGENE [Vander Linden and Martin, 1995] uses a system network and sentence-building component on top of Penman, a systemic-functional sentence generator. The system network, which encodes decisions derived from a corpus analysis of instruction manuals, makes choices ranging in scope from discourse to sentence and phrase level. Realization statements, indicating that particular words or constructions are to be used, are associated with features of the networks. The action representation is done in the description logic LOOM and includes some lexical information (and therefore is not language-neutral). Contextual factors considered include interpersonal as well as discourse factors. Lexical choice (in this case, determining the grammatical form of purpose relations) is done by system networks.

Hartley and Paris [Hartley and Paris, 1996] encode correlations of task elements and linguistic features in a strata of networks of realization choices. The task elements include goals, functions, constraints, *etc.*, in the domain of software instruction manuals. The realization choices are based on systemic functional linguistics (SFL) and they use a SFL-based tactical generator.

Dale [Dale, 1992] focuses mainly on generating referring expressions, including determining when particular anaphoric forms (pronouns, reduced noun phrases, *etc.*) are licensed (i.e., appropriate). He takes a simple view of actions, reducing the complexity found in the real world to state-change semantics. However, his approach to generation is part of the inspiration for the SPUD generation system described below. He uses simple feature-structure representations and a series of mapping algorithms to transform semantic content into surface structure.

Nicolov, Mellish, and Ritchie [Nicolov *et al.*, 1996] exploit the declarative relationship between a non-hierarchical semantic representation, in the form of conceptual graphs, and a linguistically-motivated syntactic representation. Conceptual graphs are a language-neutral domain representation. D-Trees, a variation of TAG, are used for the lexicon and grammar. Their approach to generation involves incrementally finding mapping rules (semantics to syntax) to cover as much of the semantics in a conceptual graph as possible while adding as little extra information to the resulting text as possible. Their method allows for the linguistic realization of a conceptual graph to be spread over multiple sentences.

GhostWriter [Merchant *et al.*, 1996] uses a knowledge-based model of plans and actions in language-neutral form as basis for generation. An explicit fine-grained action representation is used, making it mostly language-independent. However, actions can have a linguistically-oriented representation associated with them. In fact, there are concept-lexeme mapping structures in the lexicon. Action schemas are used for building a plan, which then can be used as input to the generator.

SPUD [Stone and Doran, 1997; Stone and Webber, 1998] focuses on generating contextually appropriate descriptions of entities, much like [Dale, 1992]. However, it extends beyond Dale's work since it considers information contributed by the whole sentence to a referring expression. Descriptions are not limited to objects but can be generated for actions, events, and states as well. SPUD uses the idea of distinguishing an entity from its potential distractors to drive the generation process. It can also be given explicit communicative goals to achieve while describing an entity. The generation process is incremental, adding one lexical item at a time and evaluating intermediate results. The lexicon includes information about lexical items' syntax, semantics, as well as pragmatics, all of which is used to perform lexical choice. The syntax for the lexical items is represented by a Lexicalized Tree-Adjoining Grammar, a variant of TAG. Domain information is represented in modal first-order logic which is suitable for non-linguistic tasks, such as planning and reasoning. Thus, SPUD is both a declarative and incremental approach to generation.

Since I use SPUD to implement the generation of expressions of action termination, I will return to it in detail in Chapter 5. First, however, I present an analysis of naturally-occurring expressions of action termination, on which I base my encoding of linguistic constructions in SPUD.

Chapter 4

Corpus Analysis

To see how termination is expressed in naturally-occurring instructions, I look at a "do-it-yourself" book as well as a set of technical orders for maintaining F-16 aircraft. Even restricting the instructions to the *step-by-step* subset discussed in Chapter 2, I find a wide variety of constructions, especially in the "do-it-yourself" corpus. The constructed set of minimal pairs given in Figure 2.1 reflects most of this variety. The main expressions of termination information are predicate-argument structure, statements of purpose, and *until* clauses. In the following sections, I give details about the corpus, how data is coded, how often various constructions are used and in what general contexts, and what future analysis should be done and why.

4.1 About the Corpus

The corpora examined so far are the numbered instructions in the *Reader's Digest New Complete Do-It-Yourself Manual* [Reader's Digest, 1991], in a version of the *Organizational Maintenance Job Guide (Fuel System Distribution, USAF Series F-16C/D Aircraft)* [USAF, 1988] (a set of technical orders for the maintenance of F-16s), and in a set of instructions for a mitre saw assembly line [ITL SIMA, 1997]. The mitre saw assembly line instructions are all numbered with no paragraph-length sections, and are meant as actions to be carried out by (virtual) workers on an assembly line. As mentioned in Section 2.1, such step-by-step instructions are recognized as a sub-genre of instructions manuals by virtue of their distinguishing linguistic characteristics [Hartley and Paris, 1996]. Thus, focusing on the step-by-step portions of the corpora sources is well-motivated and linguistically sound.

The step-by-step subset of the corpora contains over 3000 sentences and over 3500 verb phrases, using over 380 distinct verbs. Many of the instructions contain subordinate clauses, which accounts for the fact that verb phrases outnumber sentences. As noted below, each verb phrase is coded, including its relationship to other verb phrases in its sentence.

4.2 Coding Methodology

I have coded the corpus to indicate the source of termination in each verb phrase (see Figure 4.1). The codes include whether termination comes from a culmination associated with the action. Culmination, which specifies a change of state as well as a termination, can be inherent in a verb (**NC**), provided by a combination of verb and argument (**CC**), or given in a termination condition phrase (**TCC**). (See Section 3.1 for the discussion of actions and culmination.) In addition, verb phrases are coded with **plural** or **mass** when their main arguments are plural or mass objects, respectively. The code **iter** is used when the sentence has an explicit "iteration" phrase (e.g. "five or six times").

If there is more than one verb phrase in a sentence (or if an additional phrase contributes to the culmination or termination of the main action), the relationship between the main verb phrase and the subordinate verb (or other) phrase is coded as shown in Figures 4.2 and 4.3. The additional clauses shown in Figure 4.3 generally indicate temporal relationships between states, events, and actions, and thus potentially provide termination information for actions in the main clause. When the action described in a sentence has a sub-step relationship with an action described in the preceding or following sentence, the sentence is coded with **SS**.

4.3 Analysis Results

As shown in Table 4.1, over 70% of the verb phrases in the corpora have a culmination that is inherent in either the verb or the combination of the verb, its arguments, and

NC (Natural Culmination)
The verb is an accomplishment or achievement verb and thus it has an inherent culmination (and therefore termination).

CC (Composite Culmination)
The verb and its arguments (possibly optional) contribute to a culmination, together specifying an accomplishment or achievement.

TCC (Termination from Culmination Condition)
The base action is an activity but an argument or an additional phrase provides a culmination condition (and thus termination).

TC (Termination Condition)
The base action is an activity but an argument or additional phrase provides a termination condition (but no culmination).

TI (Termination Inferred)
The termination of the base activity has to be inferred from the context of the overall task and other actions involved.

NA (Non-Action) The verb phrase does not involve a specific action on the part of the agent (e.g. *keep*, *prevent*, *maintain*) and has no inherent termination.

Figure 4.1: Verb Phrase Type Codes for Source of Termination Information

PC-to, PC-iot (Purpose Clause using *to*, *in order to*)
The action in the subordinate verb phrase (introduced by *to* or *in order to*) is the purpose of the action in the main verb phrase. This is what linguists call a *purpose clause.*

PC-by ("Purpose Clause" using *by*)
The action in the main verb phrase is achieved by doing the action in the subordinate verb phrase introduced by *by.* In linguistics, this is more properly known as a *means clause* but I will refer to it as "purpose" since a purpose relationship exists between the two actions.

PC-and ("Purpose Clause" using *and*)
A "purposive and" [Doran, 1993] is used to indicate that a purpose relationship (usually enablement) exists between the conjoined actions.

PC-fa ("Purpose Clause" using a free adjunct)
The purpose or goal of the action in the main verb phrase is the action in the free adjunct clause, e.g. "Cut the paper diagonally, *creating two triangles.*"

PC-for ("Purpose Clause" using *for*)
The purpose of the action in the main verb phrase is given in a *for* phrase. Although it does not contain a true verb phrase, the *for* phrase can have a nominalization of a verb. I designate it as a "purpose clause" because of its role of specifying purpose, even though it appears in a noun phrase.

PC-st, PC-so ("Purpose Clause" using *so that*, *so*)
The purpose (and usually manner) of the main action is stated in an additional clause introduced by *so that* or *so.* Typically, the subordinate clause expresses a state or condition to be achieved.

Figure 4.2: Codes for Purpose Relationships between Verb Phrases

term-fa, term-until (Termination using a free adjunct or *until*)
The event or state expressed in a free adjunct clause or an *until* clause specifies the termination condition of the action in the main verb phrase.

term-for (Termination using *for*)
Termination is explicitly stated by giving a duration of time using a *for* phrase.

when, if (*When, If* clauses)
A state of the world expressed in a *when* or *if* clause indicates the initiation (if any) of the action in the main clause. (For an action whose termination is specified by a *when* clause in the next sentence, that action's sentence is coded with **when.**)

before, after (*Before, After* clauses)
A *before* or *after* clause appears in the sentence, indicating the temporal relationship of the main action with the action, state of the world, or event in the subordinate clause.

during, while (*During, While* clauses)
A *during* or *while* clause appears in the sentences, indicating that the main action and the subordinate action, event, or state of the world are to be simultaneous.

Figure 4.3: Codes for Termination Clauses and Other Additional Clauses

additional phrases. Over 25% of the verb phrases do not have an inherent termination (**TCC**, **TC**, and **TI**); nearly half of those have culmination information added to them by their arguments or additional phrases or clauses (those with the code **TCC**). The frequency of the particular types of additional phrases and clauses is shown in Table 4.2. As the table indicates, over 300 "purpose clauses" (in my broad use of the term) appear and nearly two-thirds of them are expressed with a *to* clause (**PC-to**). The occurrences of the sub-step code (**SS**) and the termination codes (nearly all of which are *until* clauses) are less frequent, as are the rest of the phrase and clause codes.

The number of occurrences of the **plural** code, indicating the use of a plural argument, is over 800 but is not listed in the Table 4.2 because it is not an additional phrase or clause. Most of the plurals occur with the "composite culmination" (**CC**) verb phrase type (see Table 4.3). In most cases, the presence of multiple objects, usually of a uniquely identifiable number, creates the iteration of the base action; the entire action terminates when the base action has been done for each object. Over half of the **CC** verb phrases, and therefore nearly a fifth of all the verb phrases, have a plural argument (see Table 4.7). Although I

VP Type	#	Pct
CC	1367	38%
NC	1265	35%
TCC	414	12%
TC	330	9%
TI	143	4%
NA	61	2%

Table 4.1: Distribution of Verb Phrase Types

Type	#
PC-*	315
PC-to	204
PC-by	33
PC-st	33
PC-fa	22
PC-for	12
PC-and	5
PC-so	4
PC-as	1
PC-iot	1
SS	71
term-*	67
term-until	61
term-for	3
term-fa	2
iter	39
when	37
before	17
after	13
if	9
while	9
during	2

Table 4.2: Frequency of Additional Phrase/Clause Relation Types

VP Type	Pct of plural
CC	83%
TCC	11%
TC	3%
TI	2%

Table 4.3: Distribution of Plurals across Verb Phrase Types

VP Type	Pct of PCs
TC	32%
NC	31%
CC	19%
TCC	12%
TI	3%

Table 4.4: Distribution of Purpose Clauses across Verb Phrase Types

have focused my work so far on the additional phrases and clauses that provide termination information, the effect of plural arguments on the termination of actions cannot be ignored and should be addressed in the completed dissertation. (See Chapter 6 for proposed further work on the corpus analysis).

The distribution of purpose clauses across the verb phrase types (Table 4.4) is interesting. As expected, purpose clauses seem to provide termination information for verb phrases that do not otherwise have it. Nearly a third of the purpose clauses co-occur with verb phrases that acquire just termination from additional phrases (**TC**); over ten percent co-occur with verb phrases that also acquire culmination information (**TCC**), which provides termination information. For the purposes of my work, the distinction between termination and culmination is not vital. However, coding verb phrases as **TCC** and **TC** keeps track of how termination information is being provided, i.e. as part of a culmination or as a separate termination condition.[1] Forty percent of the purpose clauses are used with verb phrases that already have culmination information (**NC** and **CC**). In these cases, the purpose clauses are more frequently providing manner information rather than additional

[1] I should note at this point that termination conditions can provide termination for activities as well as actions which are normally accomplishments or achievements but are to be terminated before they culminate.

PC type	VP type in PC					
	NC	CC	TCC	TC	TI	NA
PC-to	42.6%	21.5%	3.4%	2.4%	8.8%	19.1%
PC-by	54.5%	33.3%	3.0%	0.0%	3.0%	3.0%
PC-fa	45.4%	27.2%	18.1%	0.0%	4.5%	0.0%

Table 4.5: Distribution of Verb Phrase Types in Purpose Clauses

VP Type	Pct of term
TC	62%
TCC	26%
CC	6%
NA	6%

Table 4.6: Distribution of Termination Phrases across Verb Phrase Types

or modified culmination information. For instance, here is an example from [Reader's Digest, 1991]:

(16) Glue panels together with white or yellow glue. Clamp around perimeter, and weight the center *to ensure proper bonding.*

A more detailed analysis would show the distribution of the different functions of purpose clauses, e.g. how often purpose clauses are used to provide termination/culmination versus manner information.

One measure of whether purpose clauses are providing termination or manner information is an analysis of the type of verb phrases in purpose clauses. As Table 4.5 shows, most of the verb phrases used in purpose clauses have inherent culmination. Since this culmination can provide termination information for the action in the main clause, this could indicate that the purpose clauses are providing termination information. Purpose clauses with "non-action" (**NA**) verb phrases (e.g. *keep*, *prevent*, *etc.*) can only convey manner information (see Example 16 above) since **NA** verb phrases cannot pass along termination information. As mentioned before, however, further analysis is needed to sort out how purpose clauses are used in the corpus.

Turning now to the termination phrases and clauses (the most frequent being *until*

Main VP	No code	plural	PC	PC-to	term	term-until	SS
TC	40.0%	8.8%	32.3%	21.5%	13.0%	12.6%	7.6%
TCC	63.7%	22.4%	9.6%	6.6%	4.1%	3.7%	2.2%
CC	37.5%	54.3%	8.0%	5.0%	0.3%	0.3%	3.5%
NC	82.2%	0.3%	5.2%	3.4%	0.0%	0.0%	2.0%

Table 4.7: Co-occurrence of Verb Phrase Types with Various Codes

clauses), nearly ninety percent of them are used with verb phrases that thereby acquire termination or culmination (**TC** and **TCC**), as shown in Table 4.6.[2] However, my analysis has not taken into consideration all of the phrases that can provide termination information, such as prepositional phrases (see Examples 9, 12b, and 13b in Figure 2.1). Further analysis could refine how termination information is provided when an explicit termination phrase or clause, such as *until*, is not present.

A rough analysis of the syntactic and semantic contexts in which each of the main verb phrase types appears is shown in Table 4.7. For instance, verb phrases with verbs that have inherent culmination (**NC**) appear, by and large, on their own; only five percent appear with purpose clauses, which are mostly providing manner information. Over ninety percent of the verb phrases with composite culmination (**CC**) appear alone or with a plural argument (discussed above). The most diverse distribution of verb phrase type versus other codes is the verb phrases that only acquire termination (**TC**). Only forty percent of these appear on their own, while 45% appear with a purpose clause or a termination phrase. (Note that in the table, the **PC-to** column indicates the percentage of *all* of the **TC** verb phrases which appear with a *to* purpose clause.) The fact that the **TCC** verb phrases, which acquire culmination, appear with no code over 60% of the time is a bit surprising, but this is because the analysis does not encode prepositional phrase arguments. Prepositional phrases, especially those describing a path such as "to the store", can add a culmination to an activity verb, e.g. "Run" versus "Run to the store". In the full dissertation, the corpus analysis will take into account termination information from prepositional phrases.

[2]An example of a verb phrase which is coded as already having culmination but which also co-occurs with an *until* phrase is:

(17) Refuel aircraft until FWD FUEL LOW and AFT FUEL LOW caution lights go out. [USAF, 1988]

While further corpus analysis is needed for a fuller picture of how termination is expressed, the current analysis has been sufficient for choosing the specifications of the action representation as well as the constructions to examine. The next chapter describes the work that has been done following the corpus analysis.

Chapter 5

Generating Instructions from Action Information

To build an action description, information about the individual parts of the action must be available. Combining the descriptions of those aspects of an action which need to be expressed can fill out the entire action description. If, after choosing the main parts of the action description (e.g., the verb *turn* and the object *the knob*), the action description is not sufficient for performing the action (e.g., no termination information is specified or inferable), then other pieces of information will need to be expressed in the action description. In this chapter, I describe how action information is represented, how lexical information is represented in order to describe action information, and how instructions are generated from the action and lexical information, including how effective action descriptions are ensured.

5.1 Parameterized Action Representation (PAR)

As described in Section 2.3.1, the action representation that I use is a *Parameterized Action Representation* (PAR). It can be viewed as a feature structure as in Figure 5.1. It can also be reformulated into first-order logic predicates, mapping attributes (e.g. `agent`) to predicates and values (e.g. `agent1`) to arguments. Given a PAR instance, say `act1`, an attribute-value pair could then be predicated of `act1`, e.g. `agent(act1, agent1)`. This

```
PAR
[ applicability conditions:  CONDITION boolean-expression
  start:                     TIME/STATE
  result:                    TIME/STATE
  participants:        [ agent:   AGENT
                         objects: OBJECT list ]
  core semantics:      [ preconditions:  CONDITION boolean-expression
                         postconditions: CONDITION boolean-expression
                         motion:         MOTION
                         force:          FORCE ]
  path:             [ direction: DIRECTION
                      start:     LOCATION
                      end:       LOCATION
                      distance:  LENGTH ]
  purpose:          [ achieve:  CONDITION boolean-expression
                      generate: PAR
                      enable:   PAR ]
  termination:     CONDITION boolean-expression
  duration:        LENGTH
  manner:          MANNER
  subactions:      PAR constraint-graph
  parent action:      PAR
  previous action:    PAR
  concurrent action:  PAR
  next action:        PAR ]

MOTION
[ object:        OBJECT
  caused:        BOOLEAN
  translational: BOOLEAN
  rotational:    BOOLEAN ]

FORCE
[ object:           OBJECT
  point of contact: OBJECT LOCATION ]

LENGTH
[ units:  UNIT
  number: QUANTITY ]
```

Figure 5.1: Parameterized Action Representation

states that agent1 is the agent of action act1. Such a reformulation is used to provide action and world information to SPUD, as shown in the next section. What is important is not how the action representation looks, but rather what information it holds.

The components of PAR marked with an asterisk (*) below are not addressed in the implemented examples that appear in this proposal. These are necessary for a full account of instructions that appear in the corpus and therefore will be used in the implementation of my complete dissertation.

Applicability conditions* is a boolean expression of conditions (conditions conjoined with logical *and*s and *or*s) which must hold (be true) in order for the action to be appropriate to perform. These conditions generally have to do with certain properties of the objects, the abilities of the agent, and other unchangeable or uncontrollable aspects of the environment. Unlike the preconditions (see below), it would be impossible or impractical to try to satisfy the applicability conditions as subgoals before performing the action.

Start is the time or state in which the action begins.

Result is the time or state after the action is performed.

Agent is the animate entity who performs the action. The representation of the agent can include its physical attributes and its capabilities.

Objects* is the list of entities/objects involved in the action. The representation of objects include physical properties such as geometry and current state as well as actions defined for the objects. It is possible that the list could associate roles, such as *instrument*, along with the entities.

Core semantics represents the primary components of meaning of the action.

- **Preconditions*** is a boolean expression of conditions that must be satisfied before attempting the action in order for the action to be successful. Although disjunctions are possible, it is generally just a condition or conjunction of conditions. The use of preconditions is the traditional method of *subgoaling* that is found in planning.

- **Postconditions** is a boolean expression of conditions which holds after the action is done (i.e., in the result state). These generally predicate changes of state in object properties and/or relations between objects.

- **Motion** represents any motion component of the action. It is a substructure which indicates the object undergoing the motion, whether the motion is translational and/or rotational, and whether it is *caused* motion.

- **Force*** represents any explicit force component of the action. It is a substructure containing the object to which the force is applied and the point of contact.

Path represents any path information for the action. It has multiple components:

Direction* gives the direction of any motion or force. Directions can be absolute or relative to an object or agent.

Start* indicates the starting location of the motion. The location will generally be represented by a relation (e.g. *on*, *at*) with an object.

End indicates the end location of the motion.

Distance* indicates the length along the path. A length consists of units (e.g. *miles*, *degrees*) and a quantity (e.g. *90*).

Each of these path components can appear alone or with any of the others. For instance, the instruction, "Move the lever *downward to the locked position*," has both the **direction** and **end** components, respectively.

Purpose indicates the purpose of the action. The purpose can include a boolean expression of conditions to **achieve** (make true), an action to **generate**, and/or an action to **enable**. Each of these has a corresponding slot under **purpose** in the PAR.

Manner* indicates any constraints on the manner in which the action is to be done.

Termination* indicates any termination conditions which would not be otherwise covered (e.g., by **purpose**). This is needed for actions in which there is no relation between the action and the conditions except that the conditions provide termination. For example, in the instruction, "Do your homework *until your mother comes home*," performing the action (*doing your homework*) does not (and cannot normally be seen as being done to) bring about the termination condition (*your mother comes home*). In the maintenance activity domain, actions with these arbitrary termination conditions are very rare since actions are usually done for a purpose not for an arbitrary condition.

Duration* indicates any explicit duration for the action. It is similar to the **distance** component of the **path** in that has units and a quantity. Although the units used for duration are usually those for time (e.g. *seconds*, *minutes*) and iteration, durations involving spatial units are also possible, e.g. "Watch the speedometer *for 10 miles*."

Subactions* represents the breakdown of the action into sub-steps. It is a collection of actions connected in a graph structure which indicates the temporal relationships (if any) between the actions (e.g. whether two actions are to be done sequentially, in parallel, *etc.*).

Parent action* is the parent action of which the action is a sub-step.

Previous action* is an action done immediately before the action.

Concurrent action* is an action which is done in parallel with the action (as indicated by the parent action's subactions graph).

Next action* is an action which is done after the action.

The remaining sections discuss the SPUD generation system, how constructions are encoded in it, and examples of the generation of instructions.

5.2 The SPUD Generation System

Developed by Matthew Stone and Christine Doran here at the University of Pennsylvania, SPUD (which stands for Sentence Planning Using Descriptions) generates descriptions of actions, events, states, and objects by choosing lexical items which serve its communicative goals best [Stone and Doran, 1997]. The generation method, described in more detail in Section 5.2.5, is an incremental approach which produces efficient and effective texts [Stone and Webber, 1998]. Using a reasoning component and various sources of information available to it, SPUD can determine what the hearer will be able to conclude from a text (even a partial one) and thus direct the generation process appropriately.

SPUD needs three main types of information to carry out the generation process, as shown in Figure 5.2. First, world knowledge (including action instances) provides the necessary information for reasoning about the world. Second, a collection of Lexicalized Tree-Adjoining Grammar (LTAG) trees provides the detailed syntactic information for lexical items. Third, the lexicon provides semantic and pragmatic information about lexical items, as well as pointers to the LTAG trees which specify their syntactic information.

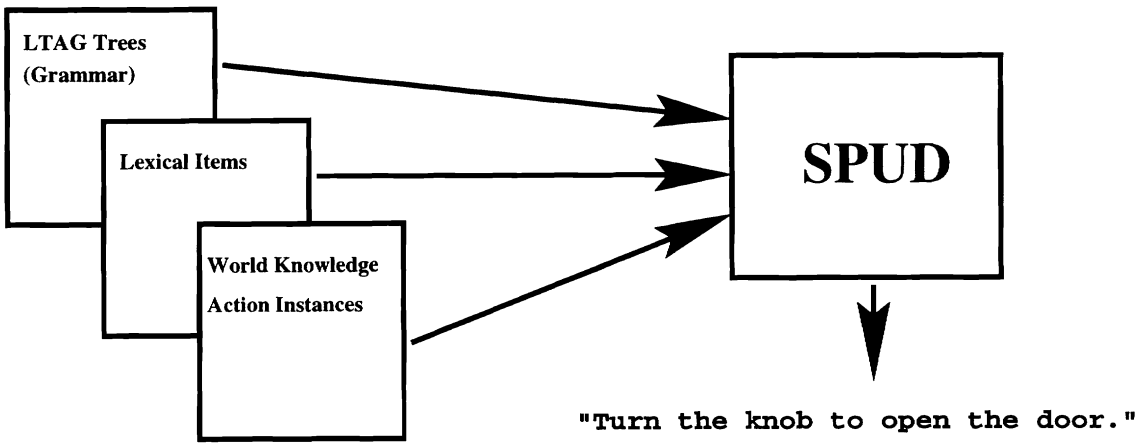


Figure 5.2: Overview of SPUD

These three sources of information are discussed below in more detail, as well as some auxiliary information.

5.2.1 Preliminaries

In addition to the three main types of information provided to SPUD, some auxiliary information, which generally does not change from domain to domain, must also be supplied. One type is morphology information (see Appendix A.6). It describes how to inflect verbs for person and tense as well as what the form of pronouns are for different cases, such as nominative and accusative. This information is used at the final stage of generation by SPUD to transform lexical items into their surface realizations. Another type of information is modal operator information (see Appendix A.5). It defines the modal operators which are used to describe the world information, including action information. A modal operator's definition includes its logical properties and its relationship to other modal operators. The two modal operators are used below: `C`, which is for overall general world knowledge, or what both the system and the hearer knows, and `I`, which is for private knowledge, or what only the system knows.

5.2.2 World Knowledge

SPUD needs information about the world in order to generate appropriate texts. World information includes descriptions of objects in terms of their properties and relationships

```
C door(door1).
C openable(door1).
C configuration(r, door1, closed).
C uniquid(door1).
```

Figure 5.3: Object information for the door

with other objects in the world. Figure 5.3 shows information about a door, where C specifies that this information is common knowledge. It indicates the internal identifier (door1), its main semantic category (door), the fact that it is an openable object[1], its configuration (closed) for the time period r, and the fact that it can be uniquely identified. Other objects are represented similarly.

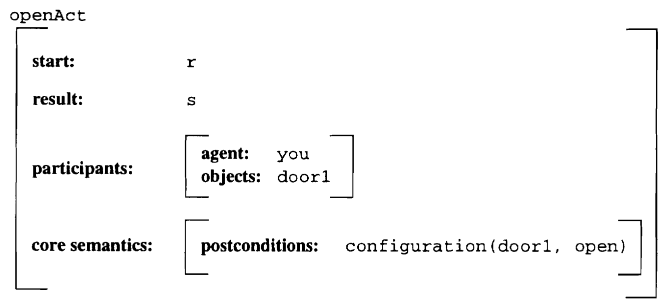
World information also includes the distractors for each entity or rules for determining an entity's distractors. SPUD uses this distractor information when determining the referring expression for an object as well as other entities. SPUD will keep adding information to the referring expression until the entity can be distinguished from its distractors by the hearer.

Any rules that are needed when reasoning about the world are also included in the world information. For instance, rules about how *places* are formed from a relation name and an object are specified. The rules for determining termination, described in Section 5.3.4, are also included. In all cases, the rules are done in modal first-order logic with quantifiers and implications.

Finally, *action instances*, instances of the action representation, are given in the world information. These represent the actions for which instructions will be generated. Figure 5.4 shows both the feature structure representation of an action instance and its logical form equivalent used in SPUD. The modal status of information is not currently included explicitly in PAR; rather, information about specific actions are considered private knowledge, otherwise there would be no point of generating instructions since the hearer would already know all about the actions. Therefore, the facts about actions use the private knowledge modal operator, I. However, there is one exception: facts of the

[1]A property such as openable could be inherited from its semantic category if an object hierarchy were used. Even in that case, however, object information would have to include exceptions to inheritance (e.g., certain doors might not be openable).

(a)



(b)

```
I start(openAct, r).
I agent(openAct, you).
C result(openAct, s).
I configuration(s, door1, open).
```

Figure 5.4: "Open the door" action instance in (a) feature-structure and (b) logical form.

```
word = { name = { open }
         basic = { true }
         decl = { alpha(S,R,E,A,O) }
         site = { s(S,R,E) }
         match = {()}
         semantics = { start(E,R), agent(E,A),
                       ?Q(result(E,Q), configuration(Q,O,open)) }
         presupposition = { true }
         pragmatics = { openable(O) }
         trees = { sVnp(S,R,E,A,O), iVnp(S,R,E,A,O) } }.
```

Figure 5.5: Lexical entry for *open*

form result(A, R) are considered common knowledge (C). It is reasonable to assume that the hearer knows that there is some time period (e.g., s in Figure 5.4) following an action or event (e.g., openAct), but not the particular conditions that hold in it (e.g., I configuration(s, door1, open)).

The same world and action information features are used in the specification of lexical items, as shown below. For a complete listing of the world information file used in generating the example instructions, see Appendix A.1.

5.2.3 Lexical Information

SPUD needs to know about the lexical items that it can use to form descriptions. Information about lexical items include syntax, semantics, and pragmatics. An example lexical entry, giving information for the transitive verb *open*, is shown in Figure 5.5. The fields in the lexical entry are as follows:

- `name` gives the word represented by the lexical entry.
- `basic` indicates whether the lexical item is considered a basic word.

Syntactic information:

- `decl` indicates the type of trees the lexical item uses (`alpha` is for initial trees, explained in the next section) and gives the declaration of how the trees will be passed information (i.e., the order of its arguments).
- `site` indicates the type of node which the lexical item can expand, which is important for determining which lexical items are applicable at any given time in the generation process.
- `match` gives any syntactic features on the node to be expanded which must be matched in order to use the lexical item.
- `trees` gives the names of those trees which the lexical item can anchor. The full structure of these trees, i.e. the syntax, is given in a separate file (see the next section), as many lexical items can refer to the same tree.

Semantic and pragmatic information:

- `semantics` contains the meaning of the lexical item, using the same action representation features as in the world knowledge. As shown in the lexical entry in Figure 5.5, existential quantifiers (e.g., `?Q` which stands for $\exists Q$) can be used in the semantics to predicate the existence of some entity which is not otherwise declared in the lexical entry.
- `presupposition` can be used to license the particular lexical item when the presupposition information is common knowledge. This is a useful feature for producing efficient texts, but I do not currently take advantage of it and thus this field will always contain `true` and lexical items will always be licensed in this sense.
- `pragmatics` provides constraints on the situations in which the lexical item can used. For instance, *open* can only be used when its object is an `openable` object.

```
word = { name = { "open" }
         basic = { true }
         decl = { beta(X) }
         site = { n(X) }
         match = { () }
         semantics = { label(X,open) }
         presupposition = { true }
         pragmatics = { true }
         trees = { bNN(X,open) } }.
```

Figure 5.6: Lexical entry for the adjective *"open"*

```
word = { name = { position }
         basic = { true }
         decl = { alpha(X) }
         site = { np(X) }
         match = { (number singular; person third; gender neuter) }
         semantics = { position(X) }
         presupposition = { true }
         pragmatics = { true }
         trees = { aTheNNs(X), aANNs(X), aNN(X) } }.
```

Figure 5.7: Lexical entry for the noun *position*

Appendix A.2 gives all of the lexical entries used in the example instructions. However, Figures 5.6 and 5.7 show two more lexical entries for other parts of speech and more lexical entries are shown in Section 5.4. The next section briefly describes the specification of the LTAG trees.

5.2.4 Trees (Grammar)

In order to generate sentences, SPUD needs syntactic trees which can be combined to form sentences. These are specified in a Lexicalized Tree-Adjoining Grammar (LTAG) [Schabes, 1990]. LTAG is a variant of TAG in which each tree is *anchored* by (i.e., associated with) at least one lexical item. An LTAG tree gives the syntactic structure associated with the anchor lexical item. For example, consider the tree shown in Figure 5.8.[2] The tree

[2]This graphical tree format is used for readability purposes. The actual input format to SPUD is shown in Appendix A.3. Although not shown in the graphical format, syntactic features as well as pragmatic information are associated with each tree.

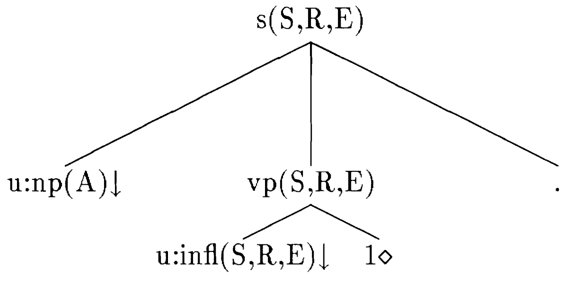


Figure 5.8: Tree for intransitive verbs, *sV(S,R,E,A)*

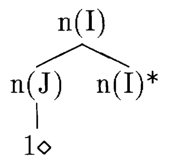


Figure 5.9: Noun modifier auxiliary tree, *bNN(I,J)*

represents an intransitive verb anchoring a declarative sentence. Each node is labeled with its category and its semantics, e.g. np(A) is a noun phrase describing the entity A.[3] The leaves with downward arrows (↓) indicate substitution sites, where trees of the right category can be inserted into the tree. The numbered diamond (1◇) indicates the position of the lexical item which anchors the tree.

The type of tree shown in 5.8 is called an initial or *alpha* tree; it provides the complete syntax for the category indicated by the top node. Initial trees fill substitution sites. The other type of trees, called auxiliary or *beta* trees, are spliced into initial trees through the TAG operation of adjunction. For instance, the tree shown in Figure 5.9 is an auxiliary tree which adds a noun-modifier to a noun node. The foot node, indicated by the asterisk (*), is the same category as the top node and gives the location for the adjunction operation. In this case, adjunction will apply to a noun node and create a noun subtree which consists of the noun and a noun-modifier with syntactic category N (noun). This same method is used to adjoin subordinate clauses to a main clause, as will be shown in Section 5.4.

This proposal does not rely heavily on the syntactic details of TAG. The most important

[3]The u: that is appended before the leaf node categories indicates that the information in these nodes can be given (or already known) as opposed to new.

point of the manipulation of the trees is how the semantics is handled, which descriptions of TAG generally do not address. As described in [Stone and Doran, 1997], when a substitution or adjunction operation is applied to a tree, the semantics of the substituted or adjoined tree is simply *conjoined* to the semantics of the original tree. Therefore, the semantics of a complicated syntactic construction is easy to compute. The operation of SPUD relies on this ease of computation, as will be shown in the next section.

5.2.5 How SPUD Works

When told to describe a particular action instance, SPUD uses the information about the action, the world, and the lexical items to choose a lexical item which furthers the description of the action (and any other communicative goals) the most. It is simply a greedy algorithm:

- Start with a tree with one node, usually an S or NP node, to describe an action or object.
- While the current tree is incomplete or there are unsatisfied goals:
  - Consider the trees resulting from adding (i.e., substituting or adjoining) one new lexical item to the current tree.
  - Compute the rank of the resulting trees based on
    - the number of goals satisfied,
    - the number of distractors for the unsatisfied goals,
    - the number of flaws (e.g., unfilled substitution sites),
    - the specificity of licensing (semantic) information (i.e., give a lower rank to trees which are subsumed semantically by other trees), and
    - whether the added lexical item is basic or not.
  - If there are no lexical items which can be added to the tree or there is no improvement in satisfying goals, leave the loop.[4]

[4]A possible useful variant of this algorithm is one which relaxes the restriction of requiring additions to provide immediate improvement. Additions which do not improve the tree immediately can pave the way for future additions which will satisfy goals.

  - Otherwise, make the highest ranking tree the current tree and go to the beginning of the loop.

- Return the current tree (which could be empty) and its derivation status:
  - If it satisfies all goals, then its *derivation completed successfully.*
  - If it satisfies some goals but none of the possible additions furthered the unsatisfied goals, then there is *no more improvement.*
  - If there are unsatisfied goals but no lexical items which could be added at all, then there are *no actions possible.*

When deciding which lexical item to add to the current tree, SPUD essentially checks which communicative goals have not been achieved. It considers all of the information provided by the tree that it has built so far. Of value to the current work is that this gives SPUD the capability of easily dealing with the fact that termination information can be provided by many different parts of a sentence. If SPUD is given the communicative goal of conveying termination information and the current tree does not yet provide it, SPUD will try to add a lexical item (possibly involving another clause) which will give termination information. In Section 5.4, I outline how SPUD does this for particular examples. First, however, I show how the constructions under consideration are encoded in SPUD.

5.3 Encoding Constructions in SPUD

For my proposal, I encoded three constructions which provide termination information: *to* prepositional phrases, *to* purpose clauses, and *until* clauses. Currently, they are characterized, distinguished, and implemented as described below.

5.3.1 Path "to"

A *path* prepositional phrase is used when the action has a path component, particularly an **end** path component. Consider instructions involving the action of turning a knob, which does not have inherent termination. By adding the *to* prepositional phrase, as in

(18) Turn the knob *to the "open" position.*

```
word = { name = { to }
         basic = { true }
         decl = { beta(P,R) }
         site = { pp(P) }
         match = { () }
         semantics = { end(P,Q), onatref(Q,R) }
         presupposition = { true }
         pragmatics = { true }
         trees = { bPPpathNP(P,R) } }.
```

Figure 5.10: Lexical entry for path preposition *to*

the turning action now has a termination (namely, that of the knob being at the "open" position).

Figure 5.10 shows the lexical entry for *to* which anchors a prepositional phrase. In the semantics, `Q` is a *place*, and `onatref` is a relation between a place and its associated entity. For example, in Example 18 above, `Q` would be the place formed by `plc(at,open_position)` and `R` would be `open_position`, which is the entity representing the "open" position of the knob. The entity (`R`), not the place (`Q`), is given as an argument to the prepositional phrase tree since the entity, not the place, is referred to in such a prepositional phrase.[5]

5.3.2 Purpose "to"

A *to* purpose clause[6] is used when a generation relationship (coextensive or delayed, as defined in Section 2.3.2) exists between the action and the purpose action described in the infinitival *to* clause. Therefore, the example instruction

(19) Turn the knob *to open the door.*

implies that turning the knob will generate, on its own, opening the door.[7] The termination of the action is defined by the generation of the purpose action.

Figure 5.11 shows the lexical entry for *to* anchoring the purpose clause. The purpose clause is adjoined to a verb phrase (VP) node and the purpose action gets described in the

[5]`onatref` is used to extract the entity which the place is defined as being *on* or *at*. The entity needs to be extracted in this manner so that it can be passed to the prepositional phrase LTAG tree (`bPPpathNP(P,R)`) as the entity to describe.

[6]This should probably be specialized to non-fronted, i.e. not sentence initial, *to* purpose clauses.

[7]The *to* purpose clause can also be used in an enablement sense, which could be implemented in a similar way.

```
word = { name = { to }
         basic = { true }
         decl = { beta(S,R,E,P) }
         site = { vp(S,R,E) }
         match = { () }
         semantics = { purpose(E, generate(P)) }
         presupposition = { true }
         pragmatics = { true }
         trees = { bAuxVP(S,R,E,P) } }.
```

Figure 5.11: Lexical entry for purpose clause *to*

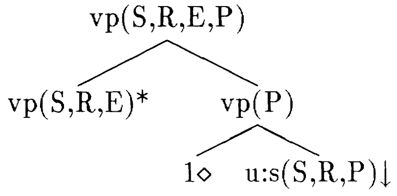


Figure 5.12: Auxiliary tree for subordinate verb phrases, *bAuxVP(S,R,E,P)*

subordinate verb phrase headed by *to*, as shown by the auxiliary tree in Figure 5.12 which *to* anchors.

5.3.3 "Until"

An *until* clause is used when the purpose of the action is to achieve a state or the occurrence of an event which is not an action done by the agent.[8] Thus, for the instruction

(20) Turn the knob *until the door opens.*

turning the knob is done for the purpose of having the door open. The event of the door opening defines the termination of the turning action.

Figure 5.13 shows the lexical entry for *until*, which is the same as the entry for the purpose *to* except for having `achieve` instead of `generate` in the `purpose` semantics.[9]

[8]An *until* clause can also be used even when no purpose relationship exists between the action and the state or event. See the discussion of the **termination** attribute in Section 5.1.

[9]*to* and *until* can use the same LTAG tree (see Figure 5.12), even though they have different deep syntax, since the entity represented by `P` contains different information in each case. `P` is an action done by the agent in the `generate` semantics as opposed to a state or event in the `achieve` semantics. The `generate`

```
word = { name = { until }
         basic = { true }
         decl = { beta(S,R,E,P) }
         site = { vp(S,R,P) }
         match = { () }
         semantics = { purpose(E, achieve(P)) }
         presupposition = { true }
         pragmatics = { true }
         trees = { bAuxVP(S,R,E,P) } }.
```

Figure 5.13: Lexical entry for *until*

$$
\begin{array}{rcll}
\forall A, S & \text{result}(A, S) & \rightarrow \text{termination}(A) & (5.1) \\
\forall A, P, L, D & \text{path}(A, P) \wedge (\text{end}(P, L) \vee \text{distance}(P, D)) & \rightarrow \text{termination}(A) & (5.2) \\
\forall A, G & \text{purpose}(A, G) & \rightarrow \text{termination}(A) & (5.3) \\
\forall A, T & \text{termination}(A, T) & \rightarrow \text{termination}(A) & (5.4) \\
\forall A, D & \text{duration}(A, D) & \rightarrow \text{termination}(A) & (5.5)
\end{array}
$$

Figure 5.14: Rules for sources of action termination

5.3.4 Defining termination

When generating instructions, SPUD needs to determine whether an action description provides termination information. Since termination is provided by a number of sources, rules are needed to allow SPUD to reason about the information given by an action description and whether it implies the intended termination. Figure 5.14 gives some preliminary rules, included in SPUD's world knowledge, which identify the sources of termination information.

The first rule says that if an action has a result, then it has termination. This rule covers accomplishment and achievement actions, such as *open*, which proceed to their inherent culmination. (All actions with inherent culmination have a result state associated with them.) The second rule states that if an action has a path that has an endpoint or which has a finite distance, then it has termination. The third rule says that if an action

action is expressed in imperative sentence form, which is indistinguishable on the surface from an infinitival verb phrase. In the complete dissertation, additional trees will be created to reflect the different syntactic structures of the constructions.

```
I start(a13, r).  I agent(a13, you).
I caused_motion(a13, knob1).  I motion(a13, rotational).
I path(a13, p).  I end(p, plc(at,open_position)).
```

Figure 5.15: Action instance for *Turn the knob to the "open" position*

has a purpose, of any sort, then it has termination.[10] Finally, the last two rules, included for completeness although they are not needed by the current examples, state that if an action has arbitrary termination conditions or an explicit duration, then it (obviously) has termination. Using these rules, SPUD can ensure that the generated instructions provide termination information.

5.4 Generation of Example Instructions

The action instances for the example instructions (Examples 18, 19, and 20, above) all have the same basic action information about turning the knob, namely:

```
start(Action, r).  agent(Action, you).
caused_motion(Action, knob1).  motion(Action, rotational).
```

where `Action` is a placeholder for the actual action instance identifiers. Each action instance has other information which distinguishes it from the others, e.g. the information about path or purpose. Figures 5.15, 5.16, and 5.17 show the representations of the action instances, where the first two lines are like those shown above and the last line gives the features of the action which cause different instructions to be generated. (See Section 5.2.2 for details about the modal operators.)

As described in Section 5.2.2, other information about the world, such as information about the objects, events, or other actions referred to by action instances, must also be provided to SPUD to allow it to generate action descriptions. Figure 5.18 shows the action instance for the generated action (*open the door*) of Example 19 and the event instance

[10]This is obviously a first approximation. An addition to the rule could include that if the purpose is to generate another action, then the termination of the generating action is dependent on the termination status of the generated action. As discussed in Chapter 4, purposes can also convey manner information which does not necessarily provide termination information. Distinguishing the representation of manner purposes from termination purposes will have to wait for the full dissertation.

```
I start(a14, r).  I agent(a14, you).
I caused_motion(a14, knob1).  I motion(a14, rotational).
I purpose(a14, generate(open_act)).
```

Figure 5.16: Action instance for *Turn the knob to open the door*

```
I start(a15, r).  I agent(a15, you).
I caused_motion(a15, knob1).  I motion(a15, rotational).
I purpose(a15, achieve(open_event)).
```

Figure 5.17: Action instance for *Turn the knob until the door opens*

(*the door opens*) for Example 20. Figure 5.19 shows the object information for the knob, the door, and the "open" position, which is treated as an object.[11]

Finally, the following additional pieces of world information:

```
C command.
C present(r).
```

are given to SPUD in order to have it generate the present-tense imperative sentences needed for instructions.

To ensure that the generated instructions provide termination information, SPUD is given the communicative goal of `termination(A)` in addition to the goal of describing the action instance represented by `A`. SPUD can check whether this communicative goal is achieved by using the rules described in Section 5.3.4. The full specification of a generation task, what I am taking to be called a *generation instance*, is shown in Figure 5.20. It includes the symbols for the modal operators for private (`I`) and shared (`C`) knowledge used in the action instance, which action instance to describe (in this case, `a13`), what structure to realize it as (in this case, a sentence), and any communicative goals (in this case, expressing termination). Given such a generation instance, which tells SPUD to describe an action instance and communicate termination information, SPUD proceeds to construct the action description.

[11]Distractor information for these objects, as described in Section 5.2.2, is also included in the world information.

```
I agent(open_act, you).  C result(open_act, s).
I configuration(s, door1, open).

C result(open_event, s).  I configuration(s, door1, open).
```

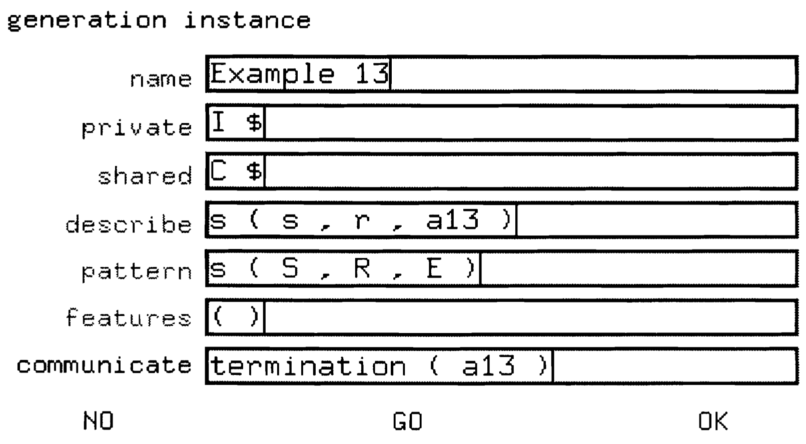
Figure 5.18: Other action/event information

```
C knob(knob1).
C rotatable(knob1).
C uniquid(knob1).

C door(door1).
C openable(door1).
C configuration(r, door1, closed).
C uniquid(door1).

C position(open_position).
C label(open_position, open).
C uniquid(open_position).
```

Figure 5.19: Object information

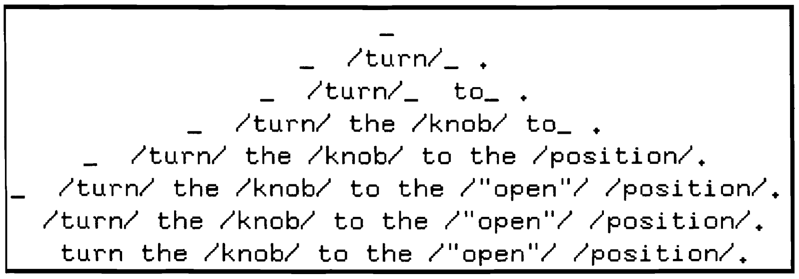
For the example instructions, SPUD constructs the sentences in the following manner. (See Figures 5.21, 5.22, and 5.23 for the generation output for the examples.) The first lexical item it adds to the sentence in each case is the verb *turn*, as it matches the basic action information in the action instances as well as being applicable to the only site (an S node) in the current tree. After the verb is added, multiple sites, corresponding to arguments of the verb, are available. SPUD looks for a lexical item which can be substituted at or adjoined to one of these sites and which furthers its communicative goals the most. In Example 18, this is the preposition *to* which adjoins to the VP node and brings in the path information. The purpose clause *to* in Example 19 and the *until* clause in Example 20 are adjoined to the VP node as well, each bringing purpose information to its respective action description. Each of these additions contribute towards satisfying the communicative goal of expressing termination.[12] Using rules such as those shown in Figure 5.14, SPUD can verify that the goal of conveying termination information is satisfied at this point. Depending on the example, the remaining uncompleted portions of the tree are

[12]Without the termination communicative goal, SPUD generates only "Turn the knob" in all three cases.



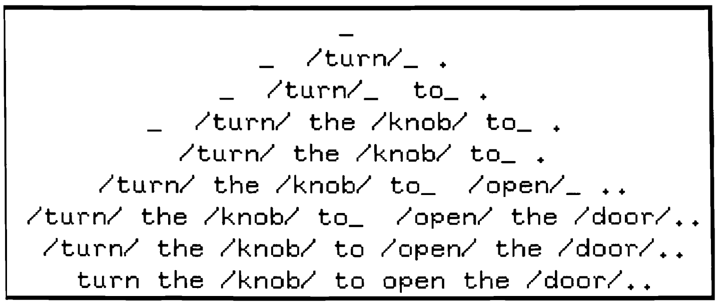
```
generation instance
       name  Example 13
    private  I $
     shared  C $
   describe  s ( s , r , a13 )
    pattern  s ( S , R , E )
   features  ( )
communicate  termination ( a13 )
     NO              GO              OK
```

Figure 5.20: Example of a generation instance



```
                          _
                    _  /turn/_  .
                 _  /turn/_  to_  .
             _  /turn/ the /knob/ to_  .
       _  /turn/ the /knob/ to the /position/.
_  /turn/ the /knob/ to the /"open"/ /position/.
  /turn/ the /knob/ to the /"open"/ /position/.
   turn the /knob/ to the /"open"/ /position/.
```

Figure 5.21: Generation of Example 13: *Turn the knob to the "open" position*



```
                        _
                  _  /turn/_  .
               _  /turn/_  to_  .
           _  /turn/ the /knob/ to_  .
             /turn/ the /knob/ to_  .
       /turn/ the /knob/ to_  /open/_  ..
  /turn/ the /knob/ to_  /open/ the /door/..
    /turn/ the /knob/ to /open/ the /door/..
      turn the /knob/ to open the /door/..
```

Figure 5.22: Generation of Example 14: *Turn the knob to open the door*

```
                        _
                   _  /turn/_ .
             _  /turn/_  /until/_ .
       _  /turn/ the /knob/ /until/_ .
         /turn/ the /knob/ /until/_ .
    /turn/ the /knob/ /until/_ _  /open/..
/turn/ the /knob/ /until/ the /door/_  /open/..
 /turn/ the /knob/ /until/ the /door/ /open/..
  turn the /knob/ /until/ the /door/ opens..
```

Figure 5.23: Generation of Example 15: *Turn the knob until the door opens*

filled in, starting with "head" (verb or noun) information, until the tree is completed. All three examples are successfully generated by SPUD, when given the information described in the sections above (listed in its entirety in Appendix A).

I have shown in this chapter how actions, information about the world, and lexical constructions are encoded and given to SPUD. Along with communicative goals, SPUD takes this information to generate effective instructions, ensuring that they include termination information. In the next chapter, I describe the proposed work to expand my preliminary work.

Chapter 6

Proposed Work

In this dissertation proposal, I have shown how action termination is expressed in effective instructions. All effective instructions, those that provide enough information to be carried out correctly, rely on some source of action termination. I have carried out a preliminary corpus analysis to guide my implementation of termination expressions using the SPUD generation system. SPUD is a declarative, incremental generation system which is capable of producing texts which are sensitive to the hearer's knowledge, efficient in the expression of information, and effective in that they satisfy the intended communicative goals.

Given the way in which SPUD constructs sentences, it is easy to see that changing information about an action instance or about the world can affect SPUD's output. For instance, the description of an action may be different depending on whether it *generates* or *enables* another action (e.g., "Turn the knob to open the door" versus "Turn the knob so that you can open the door"). Different action descriptions may also be generated if world information, e.g. a *connected* relation between two objects, guides SPUD to choose one lexical item over another (e.g., "disconnect" when objects are connected, "remove" otherwise). This flexibility and the consideration of information conveyed by all parts of a sentence makes SPUD an ideal system to use in the generation of efficient, effective instructions.

The work I propose to complete this thesis falls into two broad categories: corpus analysis and SPUD. My corpus analysis needs to be expanded to include more types of termination sources, such as prepositional phrases, as well as more context information, such as relations between actions. This expansion will add to my characterizations of linguistic constructions which can then be encoded in SPUD. The coding methodology needs to be refined and formalized to reflect this expansion and to allow a clear, consistent coding of the corpus. Using this refined methodology, I will then recode the existing corpus material. Revised results from the new coding will then update those shown in Chapter 4. I will do limited validation of my coding of the corpus by providing volunteers with the coding methodology and a representative sample of the corpus to code. I will then compare their coding to my coding of the same sample; the complete dissertation will present agreement results.

Once the expanded corpus analysis is completed, the results need to be encoded in SPUD. Whatever revisions need to be made to the currently implemented constructions presented in Chapter 5 will be made. I will then encode new constructions, based on the corpus analysis results and on the results of others (e.g. [Vander Linden and Martin, 1995]). For instance, the following types of clauses and possible hypotheses of their encoding will be addressed, from among the examples in Figure 2.1:

"so that" clauses might be used when there is a purpose or manner relationship, but not a generation relationship, between the action in the main clause and the state or event in the *so that* clause. [Vander Linden and Martin, 1995] hypothesize, based on their corpus analysis, that it can be used to express the possible "volitional" action of an inanimate substance (e.g., "so that the water runs out"). With the expansion of the corpus analysis, this hypothesis could be confirmed. If so, rules could be added to the world information for SPUD to use in determining whether the purpose involves such an action.

free adjuncts clauses might be used in the case when there is a generation relationship between the actions in the main clause and subordinate clause *and* the two actions happen concurrently. By examining the corpus, a characterization of how free adjuncts are used will be obtained and encoded in SPUD. ([Webber and Di Eugenio, 1990] perform a corpus analysis of free adjuncts that may prove helpful in my characterization.)

"by" (means) clauses might be used when the purpose (expressed in the main clause) of the action (in the "by" clause) is local and/or conditional, as defined by [Vander Linden and Martin, 1995]. In addition, [Balkanski, 1992] develops a set of *interpretation* rules to draw the appropriate inferences associated with means clauses. This work could be useful in determining when to generate means clauses. The corpus analysis will again provide a characterization of the contexts in which means clauses are used and this characterization will be encoded in SPUD.

The same will be done for the other constructions given in the set of minimal pairs (Figure 2.1). I do not expect that any substantial changes to the PAR specification will be needed in order to represent the actions expressed using these constructions. By using the existing PAR structure and adding rules to SPUD's world knowledge, these unimplemented constructions will be possible. The syntax for these constructions and others will be borrowed heavily from the work of another group at Penn who has been working on the linguistically-realistic syntax of motion verbs.

The generation of the full range of termination expressions will include multi-sentence instructions. For instance, the expression of a result, where termination is assumed as well as a causal connection, is often included in a separate sentence:

(21) Turn the knob. The door will open.

Currently, SPUD considers one sentence at a time. It will have to be expanded by adding LTAG trees which contain multiple sentences, which is being explored by [Webber and Joshi, 1998]. SPUD will then be given an action (or set of actions) and told to generate instructions in a multi-sentential (discourse) unit, corresponding to an instruction step.

Finally, some manner of validation of the instructions generated by SPUD should be done. It can be done in a similar fashion to [Vander Linden and Martin, 1995]. They reserved a portion of their corpus for testing purposes. They then compared the generated text for that portion with the actual corpus text. They then analyzed and explained the differences between the generated text and the actual text. In a similar way, my corpus analysis results and the implementation in SPUD will be supported.

In summary, my proposed work is as follows:

1. Corpus analysis:

   (a) Expand the set of constructions (e.g. path prepositional phrases, plural arguments, *etc.*) and contexts (e.g. relations between actions). Include interactions between actions; determine an appropriate causal model. Incorporate results from others where appropriate.

   (b) Formalize the coding methodology.

   (c) Recode the corpus (except for the "test set") using formal methodology and produce revised statistics and characterizations for constructions of interest. Determine the implications associated with each construction as well as any expectations that are raised with respect to the termination of actions.

   (d) Determine a representative sample of corpus and give to volunteer coders along with the formal coding methodology. Compare the coding results to determine coding agreement.

2. SPUD work:

   (a) Revise or add LTAG trees needed for constructions. Add multi-sentential LTAG trees to SPUD's repertoire.

   (b) Encode the revised characterizations of constructions as well as the characterizations of new constructions.

   (c) Modify and expand the rules for action termination to reflect the new and revised characterizations, including interactions between actions. Distinguish manner purposes from termination purposes.

   (d) Encode the action information for actions in uncoded portion of corpus (the "test set") and give to SPUD for generation. Compare the generated instructions with the actual corpus text.

Bibliography

[Allen, 1983] James Allen. Maintaining knowledge about temporal intervals. *Communications of the ACM*, 26:832–843, 1983.

[Allen, 1984] James Allen. Towards a general theory of action and time. *Artificial Intelligence*, 23:123–154, 1984.

[Badler et al, 1997] Badler et al. Natural Language Text Generation from Task Networks for Use in Automatic Generation of Technical Orders from DEPTH Simulations. Final Report to Air Force HRGA, March 1997.

[Badler et al, 1998] Badler et al. Automating maintenance instructions. Final Report to Air Force HRGA, March 1998.

[Balkanski, 1992] Cecile T. Balkanski. Actions, beliefs and intentions in rationale clauses and means clauses. In *Proceedings of AAAI'92*, 1992.

[Dale, 1992] Robert Dale. *Generating Referring Expressions: Constructing Descriptions in a Domain of Objects and Processes.* ACL-MIT Press Series in Natural Language Processing. MIT Press, 1992.

[Di Eugenio and Webber, 1996] Barbara Di Eugenio and Bonnie Lynn Webber. Pragmatic overloading in Natural Language instructions. *International Journal of Expert Systems, Special Issue on Knowledge Representation and Reasoning for Natural Language Processing*, 9(1), March 1996.

[Di Eugenio, 1993] Barbara Di Eugenio. *Understanding Natural Language Instructions: A Computational Approach to Purpose Clauses.* PhD thesis, University of Pennsylvania, 1993.

[Doran, 1993] Christine Doran. Purposive AND clauses in spoken discourse. Unpublished manuscript, University of Pennsylvania, 1993.

[Elhadad *et al.*, 1997] Michael Elhadad, Kathleen McKeown, and Jacques Robin. Floating constraints in lexical choice. *Computational Linguistics*, 23(2):195–239, 1997.

[Hartley and Paris, 1996] Anthony Hartley and Cécile Paris. Two sources of control over the generation of software instructions. In *Proceedings of the 34th Annual Meeting.* Association for Computational Linguistics, June 1996.

[How, 1993] K. Y. How. *A Processing Framework for Temporal Analysis and Its Application to Instructional Texts.* PhD thesis, University of Edinburgh, 1993.

[ITL SIMA, 1997] ITL SIMA. Visual Interface to Manufacturing: Black&Decker/ITL Visualization SIMA Project. Web Site, 1997. http://speckle.ncsl.nist.gov/~sima/vim/.

[Kalita, 1990] Jugal Kalita. *Animation of Task Performance in a Physical Domain.* PhD thesis, University of Pennsylvania, 1990.

[Kosseim and Lapalme, 1995] Leila Kosseim and Guy Lapalme. Choosing rhetorical relations in instructional texts: The case of effects and guidances. In *Proceedings of the Fifth European Workshop on Natural Language Generation*, May 1995.

[McKeown *et al.*, 1990] Kathleen McKeown, Michael Elhadad, Yumiko Fukumoto, Jong Lim, Christine Lombardi, Jacques Robin, and Frank Smadja. Natural language generation in COMET. In Robert Dale, Chris Mellish, and Michael Zock, editors, *Current Research in Natural Language Generation*, Cognitive Science Series, chapter 5. Academic Press, Harcourt Brace Jovanovich, 1990.

[Merchant *et al.*, 1996] B. P. Merchant, F. Cerbah, and C. S. Mellish. The GhostWriter Project: a demonstration of the use of AI techniques in the production of technical publications. In A. Macintosh and C. Cooper, editors, *Applications and Innovations in Expert Systems IV.* SGES Publications, 1996.

[Moens and Steedman, 1987] Marc Moens and Mark Steedman. Temporal ontology in natural language. In *Proceedings of the 25th Annual Meeting of the ACL*, pages 1–7, 1987.

[Mourelatos, 1981] A. Mourelatos. Events, processes, and states. In P J Tedeschi and A Zaenen, editors, *Tense and Aspect*, volume 14 of *Syntax and Semantics.* Academic Press, 1981.

[Nicolov *et al.*, 1996] Nicolas Nicolov, Chris Mellish, and Graeme Ritchie. Approximate generation from non-hierarchical representations. In *Proceedings of the Eighth International Workshop on Natural Language Generation*, 1996.

[Passonneau, 1987] Rebecca J. Passonneau. Situations and intervals. In *Proceedings of the 25th Annual Meeting of the ACL*, pages 16–24, 1987.

[Reader's Digest, 1991] Reader's Digest. *New Complete Do-It-Yourself Manual.* Reader's Digest Association, 1991.

[Reiter *et al.*, 1995] Ehud Reiter, Chris Mellish, and John Levine. Automatic generation of technical documentation. *Applied Artificial Intelligence*, 9, 1995.

[Reiter, 1991] Ehud Reiter. A new model of lexical choice for nouns. *Computational Intelligence*, 7(4):240–251, 1991.

[Rösner and Stede, 1994] Dietmar Rösner and Manfred Stede. Generating Multilingual Documents from a Knowledge Base: The TECHDOC Project. In *Proceedings of COLING'94*, 1994.

[Schabes, 1990] Yves Schabes. *Mathematical and Computational Aspects of Lexicalized Grammars.* PhD thesis, Computer Science Department, University of Pennsylvania, 1990.

[Scott and de Souza, 1990] Donia R. Scott and Clarisse Sieckenius de Souza. Getting the message across in RST-based text generation. In Robert Dale, Chris Mellish, and Michael Zock, editors, *Current Research in Natural Language Generation*, Cognitive Science Series, chapter 3. Academic Press, Harcourt Brace Jovanovich, 1990.

[Steedman, 1997] Mark Steedman. Temporality. In J van Benthem and A ter Meulen, editors, *Handbook of Logic and Language.* Elsevier North Holland, 1997.

[Stone and Doran, 1997] Matthew Stone and Christine Doran. Sentence planning as description using Tree-Adjoining Grammar. In *Proceedings of ACL/EACL'97*, pages 198–205, 1997.

[Stone and Webber, 1998] Matthew Stone and Bonnie Webber. Textual economy through close coupling of syntax and semantics. International Workshop of Natural Language Generation, August 1998.

[USAF, 1988] USAF. Organizational Maintenance Job Guide (Fuel System Distribution, USAF Series F-16C/D Aircraft). Technical Order Manual, 1988.

[Vander Linden and Martin, 1995] Keith Vander Linden and James H Martin. Expressing rhetorical relations in instructional text: A case study of the purpose relation. *Computational Linguistics*, 21(1):29–57, 1995.

[Vander Linden, 1994] Keith Vander Linden. Generating precondition expressions in instructional text. In *Proceedings of the 32nd Annual Meeting of the ACL*, 1994.

[Vendler, 1967] Zeno Vendler. Verbs and times. In *Linguistics in Philosophy*, pages 91–121. Cornell University Press, 1967.

[Webber and Di Eugenio, 1990] Bonnie Lynn Webber and Barbara Di Eugenio. Free adjuncts in Natural Language instructions. In *COLING90, Proceedings of the Thirteenth International Conference on Computational Linguistics*, pages 395–400, 1990.

[Webber and Joshi, 1998] Bonnie Lynn Webber and Aravind K. Joshi. Anchoring a lexicalized tree-adjoining grammar for discourse. In *Proceedings of COLING-ACL'98 Workshop on Discourse Relations and Discourse Markers*, 1998.

Appendix A

Complete SPUD Input Files

A.1 World/domain information

```
% Objects in the domain

G true.

C hearer(you).

C knob(knob1).
C rotatable(knob1).
C uniquid(knob1).

C door(door1).
C openable(door1).
C state(r, door1, closed).
C uniquid(door1).

C position(open_position).
C label(open_position, open).
C uniquid(open_position).

C position(closed_position).
C label(closed_position, closed).
C uniquid(closed_position).

% Setting up distractors for objects

C domain(you,you).

C domain(knob1,knob1).
C domain(knob1,door1).
C domain(door1,door1).
C domain(door1,knob1).

C domain(open_position, open_position).
C domain(open_position, closed_position).
```

```
% Places and their distractors

C domain (on,on).
C domain (on,at).
C domain (at,on).
C domain (at,at).

*X C onref(plc(on,X),X).
*X C atref(plc(at,X),X).
*X (onref(Y,X) -> C onatref(Y,X)).
*X (atref(Y,X) -> C onatref(Y,X)).

*X *Z *Y *W (domain(X,Z), domain(Y,W) ->
C (domain(plc(X,Y), plc(Z,W)))).

*X *Y C (domain(plc(X,Y), infinity)).

% Path distractors

*P *X *Y (end(P,X), domain(X,Y) -> C (domain(end(P,X), end(dummy,Y)))).

% Termination rules

*E *S C (result(E,S) -> termination(E)).

*E *P C (path(E, P), bounded(P) -> termination(E)).
*P *X C (end(P, X) -> bounded(P)).

*E *P C (purpose(E, generate(P)) -> termination(E)).

*E *P C (purpose(E, achieve(P)) -> termination(E)).

% Actions, Events, and Times

C domain(a13,a13).
C domain(a14,a14).
C domain(a15,a15).
C domain(open_act,open_act).
C domain(open_event,open_event).
C domain(r,r).
C domain(s,s).

% ----------------------------------------------------------------------------

C command.
C present(r).

% Example 13

I start(a13, r).
I agent(a13, you).
```

```
I caused_motion(a13, knob1).
I motion(a13, rotational).
I path(a13, p).
I end(p, plc(at,open_position)).

% Example 14

I start(a14, r).
I agent(a14, you).
I caused_motion(a14, knob1).
I motion(a14, rotational).
I purpose(a14, generate(open_act)).

I start(open_act, r).
I agent(open_act, you).
C result(open_act, s).
I configuration(s, door1, open).

% Example 15

I start(a15, r).
I agent(a15, you).
I caused_motion(a15, knob1).
I motion(a15, rotational).
I purpose(a15, achieve(open_event)).

I start(open_event, r).
C result(open_event, s).
I configuration(s, door1, open).
```

A.2 Lexical entries

```
word = {
   name = { open }
   basic = { true }
   decl = { alpha(S,R,E,A,O) }
   site = { s(S,R,E) }
   match = {()}
   semantics = { agent(E,A), start(E,R),
                 ?Q(result(E,Q), configuration(Q,O,open)) }
   presupposition = { true }
   pragmatics = { openable(O) }
   trees = { sVnp(S,R,E,A,O), iVnp(S,R,E,A,O) }
}.

word = {
   name = { open }
   basic = { true }
   decl = { alpha(S,R,E,O) }
   site = { s(S,R,E) }
   match = {()}
   semantics = { start(E,R), ?Q(result(E,Q), configuration(Q,O,open)) }
   presupposition = { true }
   pragmatics = { openable(O) }
   trees = { sV(S,R,E,O) }
}.

word = {
   name = { turn }
   basic = { true }
   decl = { alpha(S,R,E,A,O) }
   site = { s(S,R,E) }
   match = {()}
   semantics = { agent(E,A), start(E,R),
                 caused_motion(E,O), motion(E,rotational) }
   presupposition = { true }
   pragmatics = { rotatable(O) }
   trees = { sVnp(S,R,E,A,O), iVnp(S,R,E,A,O) }
}.

word = {
   name = { turn }
   basic = { true }
   decl = { alpha(S,R,E,A,O,P) }
   site = { s(S,R,E) }
   match = {()}
   semantics = { agent(E,A), start(E,R),
                 caused_motion(E,O), motion(E,rotational), path(E,P) }
   presupposition = { true }
   pragmatics = { rotatable(O) }
   trees = { sVMnp(S,R,E,A,O,P), iVMnp(S,R,E,A,O,P) }
}.
```

```
word = {
   name = {to}
   basic = {true}
   decl = {beta(P,R)}
   site = { pp(P)}
   match = {()}
   semantics = { end(P,Q), onatref(Q,R) }
   presupposition = {true}
   pragmatics = {true}
   trees = { bPPpathNP(P,R)}
}.

word = {
   name = {present}
   basic = {true}
   decl =  {alpha(S,R,E)}
   site = { infl(S,R,E) }
   match = { (tense present) }
   semantics = { present(R) }
   presupposition = { true }
   pragmatics = {true}
   trees = { atense(S,R,E) }
}.

word = {
   name = { knob }
   basic = {true }
   decl = {alpha ( X ) }
   site = {np ( X ) }
   match = {( number singular; person third; gender neuter ) }
   semantics = { knob ( X ) }
   presupposition = {true }
   pragmatics = {true }
   trees = { aTheNNs(X), aANNs(X) }
}.

word = {
   name = { door }
   basic = {true }
   decl = {alpha ( X ) }
   site = {np ( X ) }
   match = {( number singular; person third; gender neuter ) }
   semantics = { door ( X ) }
   presupposition = {true }
   pragmatics = {true }
   trees = { aTheNNs(X), aANNs(X) }
}.

word = {
   name = { position }
   basic = {true }
   decl = {alpha ( X ) }
   site = {np ( X ) }
   match = {( number singular; person third; gender neuter ) }
   semantics = { position ( X ) }
   presupposition = {true }
   pragmatics = {true }
   trees = { aTheNNs(X), aANNs(X) }
}.

word = {
   name = { "open" }
   basic = {true}
   decl = {beta(X)}
   site = {n(X)}
   match = {()}
   semantics = { label(X,open) }
   presupposition = {true}
   pragmatics = {true}
   trees = { bNN(X,open) }
}.

word = {
   name = { "current" }
   basic = {true}
   decl = {beta(X)}
   site = {n(X)}
   match = {()}
   semantics = { label(X,current) }
   presupposition = {true}
   pragmatics = {true}
   trees = { bNN(X,current) }
}.

word = {
   name = {you}
   basic = {true }
   decl = {alpha ( X ) }
   site = {np ( X ) }
   match = {(number singular; person second)}
   semantics = { hearer ( X ) }
   presupposition = {true }
   pragmatics = {true }
   trees = { propN(X) }
}.

word = {
  name = { to }
  basic = { true }
  decl = { beta(S,R,E,P) }
  site = { vp(S,R,E) }
  match = { () }
  semantics = { purpose(E, generate(P)) }
  presupposition = { true }
  pragmatics = { true }
  trees = { bAuxVP(S,R,E,P) }
}.

word = {
  name = { until }
  basic = { true }
  decl = { beta(S,R,E,P) }
  site = { vp(S,R,E) }
  match = { () }
  semantics = { purpose(E, achieve(P)) }
  presupposition = { true }
  pragmatics = { true }
  trees = { bAuxVP(S,R,E,P) }
}.
```

A.3 Tree specifications

```
entry = {
name = {sV(S,R,E,A)}
pragmatics = {true}
tree = {
node = {
type = { s(S,R,E) }
top = { (cat s) }
bottom = { (cat s) }
kids = {
subst = {
        type = { u:np(A)}
        top = { (cat np; number X; person Y; case nom) } }
node = {
        type = { vp(S,R,E) }
        top = { (cat vp; tense T; form main; number X; person Y; mode indicative) }
        bottom = { (cat vp ) }
        kids = {
                subst = { type = { u:infl(S,R,E) }
                          top = { (tense T; number X; person Y; mode indicative) } }
                anchor = { index = {1}}
        }}
words = { words = {\.}}
}}}}.

entry = {
name = {iV(S,R,E,A)}
pragmatics = {command, hearer(A)}
tree = {
node = {
type = { s(S,R,E) }
top = { (cat s) }
bottom = { (cat s) }
kids = {
node = {
        type = { u:np(A)}
        top = { (cat np; person second; case nom) }
        bottom = { (cat np; person second; case nom) }
        kids = {} }
node = {
        type = { vp(S,R,E) }
        top = { (cat vp; tense T; form main; person second; mode indicative) }
        bottom = { (cat vp ) }
        kids = {
                subst = { type = { u:infl(S,R,E) }
                          top = { (tense T; person second; mode indicative) } }
                anchor = { index = {1}}
        }}
words = { words = {\.}}
}}}}.
```

```
entry = {
name = {sVnp(S,R,E,A,O)}
pragmatics = {true}
tree = {
node = {
type = { s(S,R,E) }
top = { (cat s) }
bottom = { (cat s) }
kids = {
subst = {
        type = { u:np(A)}
        top = { (cat np; number X; person Y; case nom) } }
node = {
        type = { vp(S,R,E) }
        top = { (cat vp; tense T; form main; number X; person Y) }
        bottom = { (cat vp ) }
        kids = {
                subst = { type = { u:infl(S,R,E) }
                          top = { (tense T; number X; person Y) } }
                anchor = { index = {1}}
                subst = {
                        type = {u:np(O)}
                        top = { (cat np; case obj) } }
        }}
words = { words = {\.}}
}}}}.

entry = {
name = {iVnp(S,R,E,A,O)}
pragmatics = {command, hearer(A)}
tree = {
node = {
type = { s(S,R,E) }
top = { (cat s) }
bottom = { (cat s) }
kids = {
node = {
        type = { u:np(A)}
        top = { (cat np; person second; case nom) }
        bottom = { (cat np; person second; case nom) }
        kids = {} }
node = {
        type = { vp(S,R,E) }
        top = { (cat vp; tense T; form main; person second) }
        bottom = { (cat vp ) }
        kids = {
                subst = { type = { u:infl(S,R,E) }
                          top = { (tense T; person second) } }
                anchor = { index = {1}}
                subst = {
                        type = {u:np(O)}
                        top = { (cat np; case obj) } }
        }}
words = { words = {\.}}
}}}}.

entry = {
name = {sVMnp(S,R,E,A,O,P)}
pragmatics = { true }
tree = {
node = {
type = { s(S,R,E) }
top = { (cat s) }
bottom = { (cat s) }
kids = {
node = {
        type = { u:np(A)}
        top = { (cat np; number X; person Y; case nom) }
        bottom = { (cat np; number X; person Y; case nom) }
        kids = {} }
node = {
        type = { vp(S,R,E) }
        top = { (cat vp; tense T; form main; person Y) }
        bottom = { (cat vp ) }
        kids = {
                subst = { type = { u:infl(S,R,E) }
                          top = { (tense T; person Y) } }
                anchor = { index = {1}}
                subst = {
                        type = {u:np(O)}
                        top = { (cat np; case obj) } }
                node = {
                        type = {u:pp(P)}
                        top = { (cat pp) }
                        bottom = { (cat pp) }
                        kids = {}
                }
        }}
words = { words = {\.}}
}}}}.

entry = {
name = {iVMnp(S,R,E,A,O,P)}
pragmatics = {command, hearer(A)}
tree = {
node = {
type = { s(S,R,E) }
top = { (cat s) }
bottom = { (cat s) }
kids = {
node = {
        type = { u:np(A)}
        top = { (cat np; person second; case nom) }
        bottom = { (cat np; person second; case nom) }
        kids = {} }
node = {
        type = { vp(S,R,E) }
        top = { (cat vp; tense T; form main; person second) }
        bottom = { (cat vp ) }
        kids = {
                subst = { type = { u:infl(S,R,E) }
                          top = { (tense T; person second) } }
                anchor = { index = {1}}
                subst = {
                        type = {u:np(O)}
                        top = { (cat np; case obj) } }
                node = {
                        type = {u:pp(P)}
                        top = { (cat pp) }
                        bottom = { (cat pp) }
                        kids = {}
                }
        }}
words = { words = {\.}}
}}}}.

entry = {
name = { propN(E) }
pragmatics = { true }
tree = {
node = {
        type = { u:np(E) }
        top = { (cat np) }
        bottom = { (cat np) }
        kids = { anchor = { index = {1}}}
}}}.

entry = {
name = { p:aTheNNs(I) }
pragmatics = { uniquid(I) }
tree = {
node = {
        type = { u:np(I) }
        top = { (cat np; number X; gender Y) }
        bottom = {(cat np; number X; gender Y) }
        kids = {
                words = { words = {the}}
                node = {
                        type = { p:n(I) }
                        top = {(cat n; number X; gender Y) }
                        bottom = {(cat n; number X; gender Y) }
                        kids = { anchor = {index = {1}}}
                }
        }
}}}.
```

```
entry = {
name = { aANNs(I) }
pragmatics = { true }
tree = {
node = {
        type = { u:np(I) }
        top = { (cat np; number X; gender Y) }
        bottom = {(cat np; number X; gender Y) }
        kids = {
                words = { words = {a}}
                node = {
                        type = { n(I) }
                        top = {(cat n; number X; gender Y) }
                        bottom = {(cat n; number X; gender Y) }
                        kids = { anchor = {index = {1}}}
                }
        }
}}}.

entry = {
name = { bNN(I,J) }
pragmatics = {true}
tree = {
node = {
type = { n(I) }
top = { (cat n; number X; gender Y) }
bottom = { (cat n; number X; gender Y) }
kids = {
        node = {
                type = { n(J) }
                top = { (cat n; number singular) }
                bottom = { (cat n; number singular) }
                kids = {
                        anchor = { index = {1}}
                }
        }
        foot = { type = {n(I)} top = {(cat n; number X; gender Y)} }
}}}}.

entry = {
name = { bPPpathNP(P,R) }
pragmatics = {true}
tree = {
        node = {
        type = { pp(P) }
        top = { (cat pp) }
        bottom = { (cat pp) }
        kids = {
                foot = { type = { pp(P) }
                         top = { (cat pp) }}
                anchor = { index = {1}}
                subst = { type = { u:np(R) }
```

```
                          top = { (cat np; case obj) }}
          }
}}}.

entry = {
name = { atense(S,R,E) }
pragmatics = { true }
tree = {
node = {
          type = { infl(S,R,E) }
          top = { (tense X; form Y; number Z; person P) }
          bottom = { (tense X; form Y; number Z; person P) }
          kids = {
                    anchor = { index = {1}}
                    }}
}}.

entry = {
name = { bAuxVP(S,R,E,P) }
pragmatics = { true }
tree = {
node = {
          type = { vp(S,R,E,P) }
          top = { (cat vp; tense X; form Y; number Z; person F; mode I) }
          bottom = { (cat vp; tense X; form Y; number Z; person F; mode I) }
          kids = {
                    foot = { type = { vp(S,R,E) }
                              top = { (cat vp; tense X; form Y; number Z;
                                        person F; mode I) } }
                    node = { type = { vp(P) }
                              top = { (cat vp) }
                              bottom = { (cat vp) }
                              kids = { anchor = { index = {1} }
                                        subst = { type = { u:s(S,R,P) }
                                                  top = { (cat s) } }
                              }}
          }}
}}.
```

A.4 Generation instances

```
gen = {
        name = { Example 13 }
        private = {I $}
        shared = {C $}
        describe = { s(s,r,a13) }
        pattern = { s(S,R,E) }
        features = { () }
        communicate = { termination(a13) }
}.

gen = {
        name = { Example 14 }
        private = {I $}
        shared = {C $}
        describe = { s(s,r,a14) }
        pattern = { s(S,R,E) }
        features = { () }
        communicate = { termination(a14) }
}.

gen = {
        name = { Example 15 }
        private = {I $}
        shared = {C $}
        describe = { s(s,r,a15) }
        pattern = { s(S,R,E) }
        features = { () }
        communicate = { termination(a15) }
}.
```

A.5 Modal operators

```
dim local.

% G has information used to test specificities
G S4.

% C is overall general common knowledge
C S4 G.

% For fun purposes, we include C1, C2, C3, ...
% for what the shared info might be after some
% dialogue.  These lines indicate the chaining
% or precedence of modal operators, all of which
% are of logical type ''S4''.

C1 S4 C.
C2 S4 C1.
C3 S4 C2.
C4 S4 C3.
C5 S4 C4.
C6 S4 C5.
C7 S4 C6.

% I represents private information

I S4 C7.
```

A.6 Morphology

```
you = begin () ~> you ; end.

pro = begin
        (number singular; person third) ~> it ;
        (number plural; person third; case obj) ~> them ;
        (person second) ~> you ;
end.

the = begin () ~> the ; end.

to = begin () ~> to ; end.

\. = begin () ~> \. ; end.

open = begin
        (mode interrogative) ~> open ;
        (tense present; form main; number singular; person third) ~> opens ;
        (tense present; form main) ~> open ;
        (tense past; form main) ~> opened ;
end.

turn = begin
        (mode interrogative) ~> turn ;
        (tense present; form main; number singular; person third) ~> turns ;
        (tense present; form main) ~> turn ;
        (tense past; form main) ~> turned ;
end.
```