A FOUNDATION FOR GENERAL-PURPOSE NATURAL LANGUAGE GENERATION: SENTENCE REALIZATION USING PROBABILISTIC MODELS OF LANGUAGE

by

Irene Langkilde-Geary

A Dissertation Presented to the
FACULTY OF THE GRADUATE SCHOOL
UNIVERSITY OF SOUTHERN CALIFORNIA
In Partial Fulfillment of the
Requirements for the Degree
DOCTOR OF PHILOSOPHY
(COMPUTER SCIENCE)

December 2002

UNIVERSITY OF SOUTHERN CALIFORNIA
THE GRADUATE SCHOOL
UNIVERSITY PARK
LOS ANGELES, CALIFORNIA 90089-1695

This dissertation, written by

Irene Langkilde-Geary

under the direction of her dissertation committee, and approved by all its members, has been presented to and accepted by the Director of Graduate and Professional Programs, in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

Director

Date December 18, 2002

Dissertation Committee

Chair

Kathleen McKeown

Paul Rosenbloom

Jerry Hobbs

Acknowledgments

I owe special thanks to the Powell Family, whose foundation supported the Dean's Merit Fellowship I received during my first three years of doctoral studies. It gave me the freedom to pursue ideas that I passionately believed would work, and ultimately shaped the course of my doctoral research.

Thanks also to my advisor, Kevin Knight, whose guidance was essential to successfully completing my degree.

Thanks to the rest of my committee—Eduard Hovy, Kathy KcKeown, Paul Rosenbloom, and John Hawkins for their time, interest, patience, and contributions toward the betterment of my work.

Special appreciation goes to my fellow grad students, friends, and members of the Natural Language Group in the Intelligent Systems Division at ISI, who made the years memorable and rewarding and were always available for answering questions. This includes Sheila Tejada, Jafar Adibi, Gal Kaminka, Jose Luis Ambite, Behnam Salemi, Rogelio Adobbati, Philipp Koehn, Kenji Yamada, Alex Fraser, Hal Daume, Michael Fleischman, Deepak Ravichandran, Jay Modi, Ion Muslea, Greg Barish, Stephan Munteanu, Radu Soricut, Richard Whitney, Ulf Hermjakob, Uli Germann, Chin-Yew Lin, David Traum, Daniel Marcu, Jihie Kim, Amy Hughes, Kary Lau, and Lauri Grier. I wish also to thank

Yolanda Gil, whose simple presence, belief in me, and warm occasional interaction gave me invaluable support and encouragement. I leave for special mention my office mate Yaser Al-Onaizan, whose friendship will leave a lasting impact on my life.

I thoroughly enjoyed the three summer internships I was able to do during this period. They diversified my experiences and helped extend and refine my understanding of various areas in natural language processing. They have had a telltale impact on my research, and will continue to do so probably throughout my career. I especially appreciated the opportunity to extend my acquaintance to additional colleagues in the field. I am most indebted to Marilyn Walker, Brad Miller, Alex Rudnicky, and Dallan Quass for the opportunity to work with them.

Finally, I owe the deepest thanks to my family: to my parents, on whose shoulders I have stood, and without whom I wouldn't be here; and to my husband Stephen Geary, whose support during the last two and a half years has eased many challenges, and who in a variety of ways is enabling me to fulfill my dreams.

Contents

List Of Tables

List Of Figures

Abstract

Natural language generation (NLG) is the task of formulating a fluent sequence of words in natural language to communicate information or ideas in applications like machine translation, human-computer dialogue, automatic summarization, and question-answering. Realization, a fundamental subtask of NLG, produces an individual sentence from a sentence plan specified in terms of linguistic relations between words and/or concepts. It involves determining the order of words, inserting function words like determiners and prepositions, performing morphological inflections, and ensuring grammaticality and agreement.

An ultimate goal for natural language generation is to develop a large-scale, robust, general-purpose system. Two primary challenges are scaling up to broad coverage of syntax and producing high quality output. The irregularity of natural language makes it difficult to know how to combine linguistic primitives into fluent sentences. Also, the knowledge resources for making such a determination are time-consuming and labor-intensive to assemble, leading to a knowledge acquisition bottleneck. Evaluating whether a realizer performed appropriately is an additional challenge. There can often be more than one acceptable output, and no tools exist that can automatically assess grammaticality or fluency.

This thesis takes an approach of using probabilistic models learned from text corpora to rank candidate sentences and output the most likely. It contributes 1) a symbolic mapping rule formalism and ruleset for mapping inputs to candidate outputs that achieves broad coverage through greater regularity, 2) a packed forest representation and efficient ranking algorithm that can manage the combinatorial growth in output candidates, and 3) an empirical evaluation of coverage, correctness, and the ability to handle underspecification. This evaluation is the first large-scale empirical evaluation of coverage and quality ever performed for sentence realization.

The empirical evaluation is performed by automatically converting a set of 2400 hand-parsed sentences from the Penn Treebank corpus into system inputs, and then regenerating them using the system. The top-ranked output of the generator is compared to the original sentence. The results show better than 80% coverage of newspaper text and 94% precision (57% are exact matches) for almost fully-specified inputs, and the same coverage with 55% precision for minimally specified inputs.

Chapter 1

Introduction

Almost as soon as computers were invented, interest developed in applying them to process and generate human languages. In the late 1940's Warren Weaver and Alan Turing suggested using computers to translate between languages. The first public demonstration of a Russian-English translation system took place in 1954. By the late 1960's, popular culture dreamed of the possibility of human-computer conversations such as those in the TV show *Star Trek* and the movie *2001: A Space Odyssey.* In 1966, the famous computer psychologist program ELIZA was written which simulated the ability to converse with people. As of today, over 300 systems have been developed to generate English for task-specific purposes. In addition to machine translation and human-computer dialogue, such applications include automatic summarization, report creation, proof/decision explanation, customized instructions, item and event descriptions, question answering, tutorials, technical documentation, stories, and more.

Natural language processing is intriguing as a grand challenge for artificial intelligence. It also provides a wealth of interesting problems in traditional core areas of computer science including data structures, algorithms, complexity management and analysis,

database design and optimization, and general problem solving theory. Additionally, it has practical appeal in its potential to ease human-computer interaction, automate information management, and facilitate communication and dissemination of knowledge. The advent of the Internet has increased the demand for information processing, and many people believe that information technology will be one of the key factors driving progress during the next few decades. Natural language generation in particular can be seen as a key challenge, since learning how to synthetically reproduce a phenomenon (whether it be bird flight, growth of a baby from a single cell, or fluent English) may be one of the most effective ways of gaining a good understanding of it.

Natural language generation (NLG) is the task of formulating a fluent sequence of words in natural language to communicate information or ideas. A common subtask of NLG is sentence realization. Realization is the process of generating an individual sentence from a sentence plan that is specified in terms of linguistic relations between words and/or concepts. In the interest of achieving reusability and cost-savings in the development of applications, two prominent questions to ask are:

- *Is it possible for a single realization system to serve the varying needs of applications that perform generation—from translation, to dialogue, to summarization, etc.?*
- *If so, how would this system work?*

This thesis addresses four of the main challenges to building such a general-purpose system:

1. scaling up to broad coverage of syntax
2. producing high quality output

3. defining the input

4. evaluating the degree of success

The approach taken by this thesis builds on the work of Knight and Hatzivassiloglou (1995b) in using probabilistic language models to aid in the realization process. It develops representations and algorithms needed to scale the Nitrogen sentence generator up from a toy system to real-world, complex domains. It also performs the first large-scale empirical evaluation of a realization system, measuring syntactic coverage and quality on previously unseen inputs. The main goals of this work are to A) advance the state of the art in generation by building a realization system that meets the challenges of scale, quality, and input flexibility to a greater degree than previous systems, and B) increase understanding of the nature of language by identifying strengths and weaknesses of the symbolic and probabilistic language models used by the system to produce English. An ultimate goal is to lay a foundation on which solutions to the larger problem of general-purpose generation can more easily be built.

The rest of this chapter is organized as follows: Section 1.1 describes in greater detail the problem of natural language generation, with a focus on the subproblem of sentence realization. It offers concrete examples of the realization process from two very different applications: machine translation and human-computer dialogue. Section 1.2 outlines the main reasons why general-purpose realization is hard, which are 1) the difference in subtasks needs between different applications, 2) the complex interdependence between these subtasks, 3) the irregularity of natural language and 4) the knowledge acquisition bottleneck. Section 1.3 sketches two classes of previous approaches to reusable realization

systems—symbolic and statistical. Finally, Section 1.4 gives an overview of the work done in this thesis.

1.1 What is Natural Language Generation?

1.1.1 Automatic Text Generation

The task of formulating a fluent sequence of words in natural language is called Natural Language Generation (NLG). A variety of language-related applications, such as machine translation, dialogue, and others mentioned in the introduction need to perform this task. Generation is the inverse of the understanding task—a sequence of words is the output rather than the input in generation. Both tasks, however, involve mapping between a sequence of words and an abstract representation of their relationship to some world or domain context. This abstract representation might include the meaning, some communicative goals, and non-linguistic content.

Both tasks share the difficulty that this abstract representation is ill-defined and often task-specific. Broadly construed, an abstract representation might range from a simple category label to a fine-grained representation of syntactic, semantic, rhetorical, referential, and domain-specific structure. For example, in a broad sense natural language understanding includes shallow tasks such as topic identification and information extraction in addition to detailed sentence analysis and interpretation tasks. Analogously, NLG encompasses the production of natural language in canned form and from templates, as well as from deep specifications of goals, plans, and content.

Reiter (1995) referred to generation in this broad sense as "automatic text generation" since the NLG research community usually uses the term "natural language generation" to refer exclusively to fine-grained approaches. However, (Reiter, 1995) and others have noted that in real-world systems, a hybrid of coarse- and fine-grained techniques is often used. Elhadad and Robin (1998) argued that a re-usable generation system should be able to accommodate generation across a continuous granularity gradient. Since this thesis is concerned with the problem of general-purpose generation, its use of the term NLG includes the sense of "automatic text generation".

Yorick Wilks is credited with pointing out that while the problem of natural language understanding is somewhat like counting from one to infinity, the generation problem is analogous to going from infinity to one. In the understanding problem, a sequence of natural language words is the obvious starting point and an analytical representation of their import can be successively deepened. The deeper the analysis, the greater variety of ways there are of representing its interpretation. Also, the deeper the understanding, the more effort it requires, and the more hidden structure that must be hypothesized. In contrast, in generation the appropriate starting point is not known. The problem can be made arbitrarily easy or difficult, depending on how close to "one" it starts.

1.1.2 Examples

Illustrations of generation in the context of machine translation and human-computer dialogue are shown in Figures 1.1 and 1.2. In Figure 1.1, the translation task is modeled as a wheel with spokes. Human languages such as English, Japanese, Arabic, etc., form the rim of the wheel. The hub of the wheel is an intermediary representation, loosely

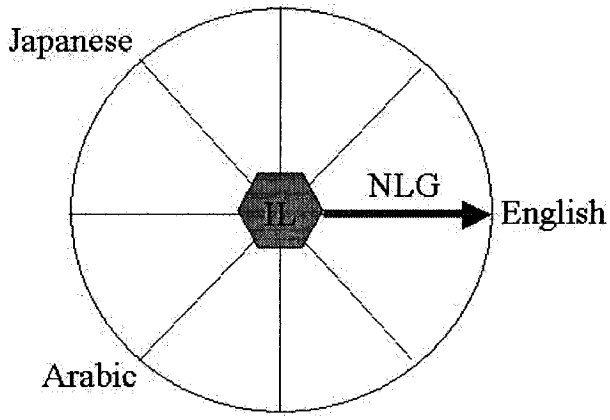


Figure 1.1: Interlingua-Based Translation

referred to as an interlingua (IL), that expresses a meaning in language-neutral terms. The translation process is analogous to traveling from the rim of the wheel to the center, and back out to the rim again, along spokes corresponding to the source and target languages respectively. An arrow from the hub to the rim corresponds to the generation task.

Figure 1.2 illustrates generation as the process of converting database output to sentences in the context of a human-computer dialogue application. The application in this example is a system that can provide people with information about restaurant and music venues around town. The figure shows a person asking a sample question about the type of food a particular restaurant serves. The task of generation begins after the system has processed the person's question and queried an internal database for the answer to the question. The figure shows a sample result of such a query, which needs to be expressed in natural language to be communicated back to the system user. The database query

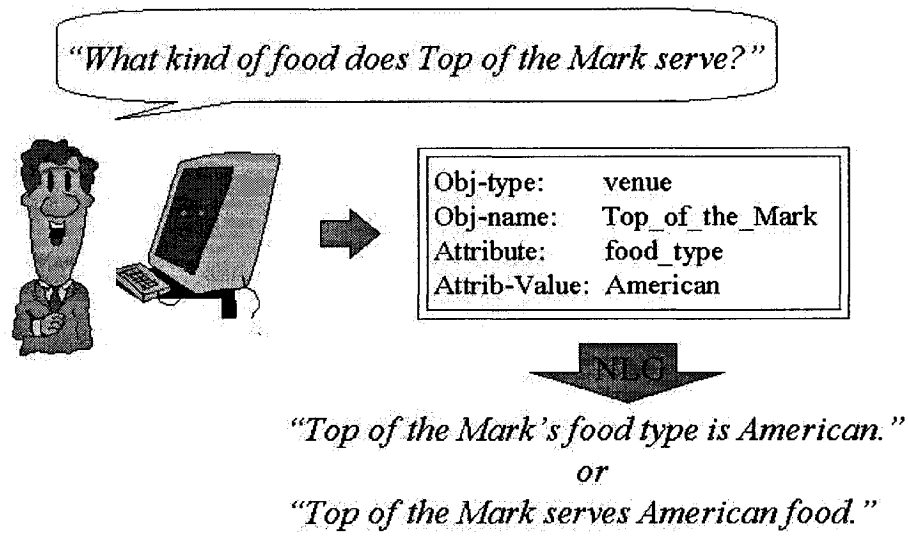


Figure 1.2: Human-Computer Dialogue

result is shown in the square box with a double border. The arrow downward from the box to two possible output sentences represents the generation task.

1.1.3 Templates versus General NLG

A simple template could suffice for generation in the isolated example above of a human-computer dialogue exchange. For example, given the query result in the figure, the following two templates could produce the output that is shown by substituting the term in angle brackets with its corresponding value in Figure 1.2.

```
<Obj-name> 's <Attribute> is <Attrib-Value>.
<Obj-name> (VERBIFY <Attribute>) <Attrib-Value> food.
```

However, templates only work in very controlled or limited situations. They cannot provide the expressiveness, flexibility or scalability that many real domains need. The

following list demonstrates the actual variety of output that may be desired. The templates above would not generate these responses, nor is it reasonable to create a different template for every possible variation. This list thus helps show how quickly the capabilities of templates may be outgrown. It also illustrates some of the subtasks that a more general generator would take on.

1. The Four Brothers' food type is American. (variation on possessification and addition of definiteness with the word "The")

2. Top of the Mark serves seafood. (example of why second template above would not always be applicable—don't want to say "seafood food")

3. The type of food that Top of the Mark serves is unknown. (missing Attrib-Value)

4. The database does not have information about the type of food served at Top of the Mark. (variation on previous output)

5. The database does not have information about the type of food served there. (use of pronoun "there" is a variation that may be more appropriate for certain contexts)

6. They serve American. (use of pronoun "they" may feel more natural)

7. American. (elliptical response more natural for certain contexts)

8. Top of the Mark serves American, Italian, and seafood. (multiple values)

9. Top of the Mark and Sullivan's serve American. (verb "serve" must agree with plural subject)

10. Top of the Mark's location is downtown. (different Attribute, if user asks about location instead)

11. Top of the Mark is located downtown. (verb expressing Attribute needs to be passive, rather than active voice).

12. Top of the Mark is located near the university. (use of prepositional phrase to express location)

1.1.4 Component Subtasks of NLG

Generation is often viewed in three stages: content determination, sentence planning and finally sentence realization (Reiter, 1994). Content planning is deciding "what to say". For instance, in the context of the dialogue application in the above example, the response to a user query about restaurants that serve American food might be a list of several restaurants. The content planning task is to decide whether to list them all, highlight a few of them, ask the user to be more specific in their query, provide the user with some options for narrowing the query, and/or reiterate to the user what the query was in order to provide feedback about the system's comprehension.

Once the content has been determined, sentence planning is the task of deciding "how to say it". This involves taking into account models of what the user already knows and the history of what has been generated previously (such as previous conversation exchanges or simply preceding statements in the current conversation turn). This information is used to decide whether to use pronouns ("they", "it", etc.) to refer to entities, or to elide some pieces of content (eg: just say "American" in response to the query in

```
(s1 / |dish up|
     :agent (t1 / Top_of_the_Mark)
     :patient (f1 / |food|
                  :mod (g1 / ''American'')))))
```

Figure 1.3: Sample Input to Realizer

Figure 1.2). It is also used to help decide how to refer to domain entities. For instance, a planner must decide whether to refer to the restaurant "Top of the Mark" with its name, or as "Top of the Mark restaurant", or "the Top of the Mark", or colloquially just "the Top", or with a descriptive qualifier like "the Top of the Mark that is located downtown". Similarly, it must decide whether to express locations as "downtown", "near downtown", or "in the downtown area".

Sentence planning can also determine what kinds of linguistic structures to use. For instance, it can express the foodtype concept using a nominal phrase ("X's food type"), or a clause ("X serves Y"). The planner needs to manage focus (ie., whether to say "American food is served by Top of the Mark" or "Top of the Mark serves American food"). It may also need to perform aggregation by grouping items together (ex: "Top of the Mark serves American, Italian and seafood", instead of "Top of the Mark serves American and Top of the Mark serves Italian and Top of the Mark serves seafood"); or by generalizing ("Top of the Mark has locations in the northern region", instead of "Top of the Mark has locations in the northeast, the north central, and the northwest regions"). Finally, a planner must organize content into paragraph and sentence-sized chunks when there is a lot of content to be expressed.

After sentences have been planned, they need to be realized. Realization assumes sentence-sized segments of input that are specified in terms of linguistic structure as

in Figure 1.3. The specification in the figure could represent "Top of the Mark serves American food", as well as some variations. Subtasks that need to be performed during realization may include:

- mapping semantic relations onto syntactic roles (such as :agent to :subject, and :patient to :object)
- making lexical choice decisions ("dish up" versus "serve", "food" versus "nutrients")
- inserting closed-class function words (such as articles: "a", "an", "the"; prepositions: "of", "for", "near", "by"; conjunctions: "and", "but"; auxiliary verbs: "is" in "is located"; etc.)
- providing defaults for unspecified syntactic features (eg: singular versus plural for nouns, present versus past tense for verbs, whether a word is a verb, noun, adjective, etc.)
- applying agreement and grammaticality constraints (not allowing "Top of the Mark serve American food" or "Top of the Mark serves a American food" or "Top of the Mark dish American food")
- determining linear precedence between syntactic constituents (eg: subject before verb, verb before object, adjective modifier before head noun)
- performing morphological inflections ("food" + plural ="foods", "serve" + past-tense = "served")

To further illustrate the realization subtasks, Figure 1.4 shows a sentence specification derived in the context of machine translation, together with its ideal output and some

not-so-ideal alternatives. The task of a realizer is to produce the ideal rather than the alternatives. The input sentence specification represents that there is some event o that is possible, or has potential, where o represents another event e, an eating action, that is obligatory. The eating action e has an agent, "you", that is performing the action, and a patient, "chicken meat", that is the object of the action.

```
(p / |possible,potential|
  :domain (o / |obligatory|
             :domain (e / |eat|
                        :agent you
                        :patient |chicken meat|)))
```

Not-so-ideal:

- You may be obliged to eat that there was the poulet.
- A consumption of poulet by you may be the requirements.
- It might be the requirement that the chicken are eaten by you.
- That the eating of chicken by you is obligatory is possible.

Ideal:

- You may have to eat chicken.

Figure 1.4: Another Sample Input to Realizer

1.2 Why general-purpose NLG is hard

1.2.1 Differences in subtasks across applications

The foregoing description of the three stages of generation—content determination, sentence planning, and realization—should be helpful for understanding what generation is. The focus in this thesis is on the last stage, realization, as it is described above. In

practice, the boundaries between the stages are not completely clear. Recent analysis has shown that implemented systems differ in how they organize subtasks into stages, and in what order they perform them (Cahill et al., 1999). To be general-purpose, a generator would need to have the flexibility to accommodate this variety. Doing so has implications for the design of the input to a sentence realizer and its system architecture. Smedt, Horacek, and Zock (1996) listed the questions this presents:

> 1 Processing aspects: How is the generation process decomposed into subtasks, modules, stages or levels? What is the control structure that coordinates the various modules and directs the information flow between them? How can we manage to be efficient while remaining flexible to handle complex demands?
>
> 2 Representation aspects: What information is relevant at the various levels to achieve linguistic expressiveness? What are the intermediate representations that serve as input and output at each level? What are the most adequate formalisms to represent and manage the various kinds of knowledge involved in generation?

Many different answers to these questions have been tried. Though certainly progress has been made, there does not yet exist a system that is truly versatile across the breadth of generation applications, attesting to the difficulty of addressing this problem. Although this thesis alone does not solve the entire problem, it makes progress on developing elements of an input representation and architecture that enable broader system applicability.

1.2.2 Expressibility

Another reason that NLG is hard is the problem of expressibility, first described by Meteer Meteer (1990). Meteer defined the problem as the gap between a text plan representation, which is usually represented in the terms of the application program; and a sentence plan,

"battalion assigned to defense/offense"
Realization: "defending battalion" / *"offending battalion"

"person who runs/skips"
Realization:"runner" / *"skipper"

"make a quick/important decision"
Realization: "decide quickly" / *"decide importantly"

Figure 1.5: Generation gap between text plan and realization

which represents in linguistic terms the resources used by the realization component to carry out the text plan. The problem of expressibility arises because it is difficult to assure that there exists a grammatical/fluent realization for any given text plan. Some of Meteer's examples are shown in Figure 1.5. (Awkward or ungrammatical phrases are marked with an asterisk, according to linguistic convention.)

Meteer posited that an intermediate level of representation, which she called a Text Structure, could resolve the expressibility problem. The Text Structure would represent: 1) what realizations, or linguistic resources/components, were available, 2) the constraints on the composition of the resources, 3) what had been committed to thus far in the utterance, which constrains the choice of resources. Her representation consisted of three kinds of constituency, semantic features corresponding to sets of interdependent grammatical features, and the semantic relationship between constituents.

Meteer's work did not try to define an exhaustive or finalized set of semantic categories and relations. Doing so would have been "tantamount to saying the whole field of linguistics is 'complete' ", (Meteer, 1990). Although the Text Structure solution is appealing, it remains to be shown whether it could actually accomplish its purpose in a large-scale system. In practice, the burden of compiling a complete set of semantic

categories and relations together with appropriately annotated linguistic resources (such as dictionary entries) acts as a formidable barrier to a large-scale implementation.

Furthermore, the problem of expressibility seems to be more idiosyncratic than Meteer originally envisioned. Some additional examples are shown in Figure 1.6. Natural language exhibits a high degree of irregularity. The acceptability of every possible combination of linguistic resources (relations, words, features) in an input is difficult, if not impossible, to predict. The traditional approach in generation, and in linguistics in general, is to annotate linguistic resources with features to regulate allowable combinations. However, creating and using such features is very labor-intensive and error-prone. Not only is such knowledge difficult to compile, its effectiveness weakens as the breadth of coverage increases. The class features and corresponding rules are simultaneously too general to eliminate undesirable combinations, and yet too restrictive to allow some combinations that are valid. This leads to a seemingly irreconcilable conflict between broad coverage and high quality output. High quality output is much easier to achieve with smaller-scale applications or in specialized domains when the size of the vocabulary or range of syntactic expressions can be limited.

The problem of expressibility extends beyond the combination of individual words. It also includes meeting application constraints on length, form, and style. For example, one might need to generate summaries of a pre-specified length, format html text to fit on a single screen, translate from one language to another the poetic effects of rhyme, rhythm, and alliteration, or tailor utterances to convey personality for animated characters. All of these tasks involve interdependencies between subtasks across the breadth of representational levels—from conceptual, rhetorical and semantic, to genre/layout and

"won/lost three straight/consecutive X."
"won/lost three straight."
*"won/lost three consecutive."
(Robin, 1994)

"finger" vs "digit" vs "phalange"
"in the end zone" vs *"on the end zone"
"tell her hi" vs *"say her hi"
"the vase broke" vs *"the food ate"
(Knight et al., 1995a)

strong/ *powerful tea
*strong/powerful car
*do/make a mess
do/*make a job

Figure 1.6: Expressibility problems

syntactic. These interdependencies add to the difficulty of designing an architecture for a general-purpose system and defining the input.

In summary, the degree of irregularity inherent in natural language and the interdependence of generation subtasks lead to the problem of expressibility. This problem needs to be addressed to achieve high quality output and general system usefulness.

1.2.3 Knowledge Acquisition

As alluded to in the previous section, a third major difficulty for general-purpose generation is the problem of knowledge acquisition. General-purpose NLG needs to operate on a scale of 200,000 entities (concepts, relations, and words), and requires large and sophisticated lexicons, grammars, ontologies, collocation lists, and morphological tables. Traditional approaches to generation acquire and apply this knowledge by hand, making large-scale, robust systems prohibitively labor-intensive and costly. Statistical approaches offer the tremendous advantage of automatic learning. However, they depend on suitably

annotated corpora which might not be available, or can themselves be expensive to obtain (though usually less so).

1.2.4 Evaluation

A final challenge of generation does not pertain to the process itself, but to evaluating its success. Generation has usually been notoriously difficult to evaluate for several reasons. First, there is the problem of what to evaluate—grammaticality? fluency? coherence in context? or some other application-specific criteria? Second, there is the inability to perform automatic evaluations. There is no tool yet that can adequately substitute for manual human evaluation of grammaticality, fluency or coherence. The same is often true of application-specific success criteria as well. This makes evaluations expensive to perform, and limits the practical size of the evaluation. Third, there are usually multiple acceptable outputs rather than a single right answer, making correctness difficult to quantify.

Fourth, the acceptability of an output can vary dramatically from one application to the next. For example, machine translation applications can tolerate some degree of disfluency in the output, while for most other applications a lack of fluency would be unacceptably distracting. Still others, such as those containing agents endowed with personality, may actually rely on errors and stylistic idiosyncrasies to convey human-like qualities.

Although grammaticality is often seen as a sufficient criterion for success, so that any of several grammatical paraphrases is considered acceptable, in reality different paraphrases make different contributions to the focus and coherence of a series of sentences.

A particular paraphrase may be grammatical, but not coherent in combination with surrounding text. Take the following pair of sentences, for instance, which are less than ideal in combination with each other: "First, the dog ate the bone. Some water was drunk by the dog then." This means that the evaluation of a sentence depends not just on the sentence itself but also on its context, which naturally varies considerably from one application to another.

Three final reasons that evaluation is challenging for generation relate to the nature of the input. Different systems use different input representations, so that comparative evaluations are either not possible or not very informative. The input representation itself is highly specialized and difficult for non-experts to comprehend. This aspect makes even human evaluations of input/output pairs difficult. Lastly, developing a test set can be problematic. Where would the inputs come from? How would they be constructed—automatically or by hand? Constructing a large set by hand can be prohibitively expensive. An exhaustive test set is impossible, since an infinite variety of inputs and outputs exist. Instead, how would a representative test set be defined? How could bias towards a specific system be avoided?

In summary, the nature of the input, the dependency on application context, and the lack of satisfactory automatic methods contribute to the difficulty of evaluating generation output.

1.3 Previous Approaches

The timeline in Figure 1.7 highlights the development of the most prominent reusable sentence realizers to date. Symbolic methods have been the standard approach to sentence realization, but in recent years the availability of large text corpora have motivated the use of statistical approaches. The systems on the timeline can be divided into two categories according their approach. The two in square boxes, Nitrogen and Fergus, used statistical approaches. This section describes the two types of approach, and relates them to the work in this thesis.

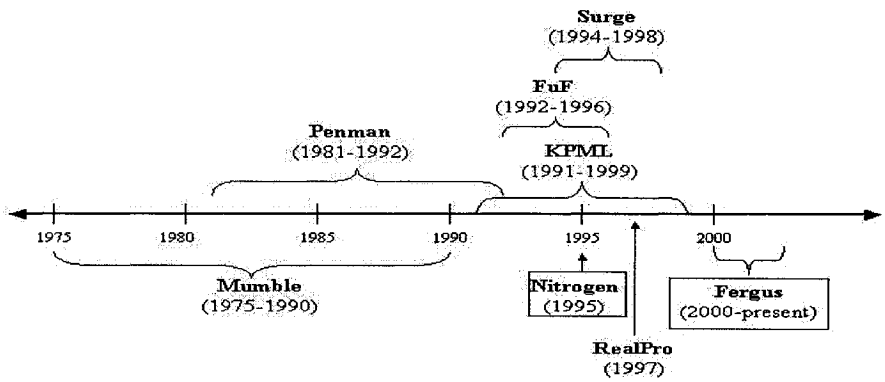


Figure 1.7: Timeline of Prominent Realization Systems

1.3.1 Purely Symbolic Approaches

MUMBLE (Meteer et al., 1987) was perhaps the first attempt to create a truly general, reusable and broad coverage grammar. In development for nearly 15 years, MUMBLE had historic roots in systemic-functional linguistic theory (Halliday, 1985), but later versions

evolved to perform realization using a surface sentence representation that is functionally equivalent to Tree-Adjoining Grammar (TAG) (Joshi, 1987). Systemic-functional theory represents and interprets language not as a rule system for generating *structures*, but as a resource for expressing and making *meanings*. It is concerned with the representation of experience (ideational aspects), interaction between speaker and listener (interpersonal aspects) and the presentation of information as text in context (textual aspects). Choice is the primary principle of organization, not structure. Systemic-Functional Frammar (SFG) does not include any representation of head-child structural dependency relations. This absense, and the problems that result from it, led to the shift in MUMBLE to TAG. TAG is a linguistic theory that represents sentence structure as elementary trees in which all relational dependencies are localized.

Penman (Matthiessen and Bateman, 1991), whose development largely overlapped with MUMBLE's was based on systemic-functional theory. It contributed the largest systemic grammar, and possibly largest machine grammar of the time, called Nigel. Generation in Penman was guided by a mechanism for making inquiries about the appropriateness of semantic alternatives. An important innovation in Penman was the use of an *Upper Model* (a domain independent word meaning ontology) to declaratively represent linguistic knowledge. It facilitated mapping between domain-specific entities and general linguistic resources. KPML (Komet-Penman Multilingual System) (Bateman, 1996) extended Penman to maximize sharing and reuse of linguistics resources for generating text in multiple languages.

FUF (Elhadad, 1993) stands for Functional Unification Formalism, and is an interpreter specifically designed to develop text generation applications. Its companion, Surge

(Elhadad and Robin, 1998), is written in FUF's language, and is probably the broadest-coverage symbolic grammar for generation of English currently in existence. FUF/Surge uses a feature-based methodology to control the grammaticality of its output. Surge's features and relations are organized overall around Systemic Functional Grammar, but additionally borrow from Head-driven Phrase Structure Grammar (HPSG) (Pollard and Sag, 1994) and other descriptive works (Quirk and Greenbaum, 1973) in order to provide extensive coverage of syntax. Surge encodes its grammar declaratively, in contrast to MUMBLE, which encodes it procedurally; and also to the Nigel grammar used in both Penman and KPML, which encodes functional aspects declaratively but its structural aspects procedurally. A declarative grammar makes Surge more portable, and increases the flexibility of the order in which it makes decisions. MUMBLE and KPML instead have a fixed order of decision-making.

RealPro (Lavoie and Rambow, 1997) is a fast commercial system based on Meaning Text Theory (MTT) (Melcuk and Pertsov, 1987). MTT is a structural dependency representation that assumes several strata of representation and focuses on the declarative mapping between them. RealPro uses the deep-syntactic level (predicate-argument and predicate-modifier structures) as its input. A key component of MTT is the Explanatory Combinatorial Dictionary, which constitutes the tool for organizing lexical information. It is a knowledge base that allows linguistic dependencies to be encapsulated outside of the input, thereby greatly simplifying the input representation. RealPro includes a medium-sized grammar. One of its greatest advantages is speed.

A commonality among all symbolic systems is the reliance on relations, "accessories", and special lexical features to tightly constrain the generation possibilities. For example, roles such as ACTOR, AGENT, PATIENT, POSSESSOR, POSSESSED, etc., in KPML and FUF/SURGE are specializations of the deep subject relation and verb complement relation, and are used to constrain the possible resulting phrases. In MUMBLE, constraints are specified categorially in the input via argument labels such as "general-clause", "general-np", etc., as well as through attachment-function specifications that limit the combinatorial possibilities. RealPro embeds constraints in the entries of its detailed lexical dictionary. Though the techniques vary, all purely symbolic systems require extensive knowledge bases to map inputs to outputs, causing the knowledge acquisition bottleneck described in Section 1.2.3. Attempts to constrain the generation possibilities also suffer from the expressibility problems discussed in Section 1.5, leading to a lack of robustness.

An additional difficulty posed by symbolic systems is that formulating their inputs can be a daunting task, requiring great expertise. Most of these generators require the specification of syntactic information in the input, which means a client must concern itself with lingustic details that are external to its application. Those that simplify the input, such as KPML and RealPro, actually just shift the burden from the input to the knowledge bases. Though this is an improvement that makes the system easier to use, the point is that it is not truly a decrease in the total information required. In practice, an application developer must often extend the knowledge resources due to lack of coverage and unique aspects of the application domain. Examples of the inputs to various systems,

together with a sample dictionary entry when available, are shown in Appendix A for a rough eyeball comparison.

In conclusion, one application developer made the following observations about the limitations of current realizers:

> After experimenting with the KPML, Surge, and RealPro realization packages, we changed our mind and reverted to shallow techniques.... The problem with the packages we examined is that none of them had both adequate documentation and broad enough grammatical coverage. For example, there is essentially no documentation on the Nigel grammar used in KPML other than a large set of examples. Surge does have some documentation, but experimentation revealed that it has many undocumented aspects as well. For example, producing the passive form of Sam sees Mary (Mary is seen by Sam) requires not just changing the focus, but also specifying the feature (agentless no); this feature is not described in the current surge documentation.
>
> The third system we looked at, RealPro, was a commercial system and did have reasonable documentation. However, RealPro's grammatical coverage did not include many constructs that we needed, and hence we could not use it in [our system] Stop. Again this is perfectly understandable; producing a well-documented commercial quality realizer is expensive, and RealPro's grammatical coverage is dictated by what is needed in the commercial projects it is used in.
>
> Using shallow techniques for syntactic processing in Stop was a disappointment. I hope that in the future some nlg group does develop a realization component which is well documented, well engineered as a software artifact, and has a wide-coverage grammar; this would allow future Stop-like projects to use deep techniques for realization. (Reiter, 1999)

1.3.2 Statistical Approaches

1.3.2.1 Nitrogen

Knight and Hatzivassiloglou (Knight et al., 1995a) introduced a statistical approach to generation that alleviated several of the problems faced by purely symbolic systems, in

particular the knowledge acquisition bottleneck. Their experiments showed that corpus-based knowledge in the form of ngram (n adjacent words) frequencies greatly reduced the need for deep, hand-crafted knowledge. Statistical knowledge could be applied to a set of semantically related sentences to help sort good ones from bad ones. In its approach, a corpus-based statistical ranker took a set of sentences packed efficiently into a *word lattice* (a state transition diagram with links labeled by English words), and extracted the best path from the lattice as output, preferring fluent sentences over contorted ones. A symbolic component produced a lattice in the first place by encoding various alternative possibilities when the information needed to make a linguistic decision was not available.

The organization of their system, Nitrogen, is shown in Figure 1.8. It is robust against underspecified and even ambiguous input meaning structures. Traditionally, underspecification is handled with rigid defaults (e.g., assume present tense, use the alphabetically-first synonyms, use nominal arguments, etc.). However, the word lattice structure permits all the different possibilities to be encoded as different phrasings, and the statistical extractor selects a good sentence from among them according to corpus-based probabilities. The same technique can be applied to knowledge gaps, which is essential, since in reality incomplete symbolic knowledge is inevitable.

Nitrogen was a significant innovation in its approach to generation. In addition to a large probabilistic language model learned from 6 million words of newspaper text, Nitrogen applied a large WordNet-based dictionary representing on the order of 100,000 words and concepts (Knight and Luk, 1994; Miller, 1990). In spite of this, however, Nitrogen was a toy system that faced several obstacles to becoming general-purpose. First, the rules in the symbolic component mapped semantic relations from the input

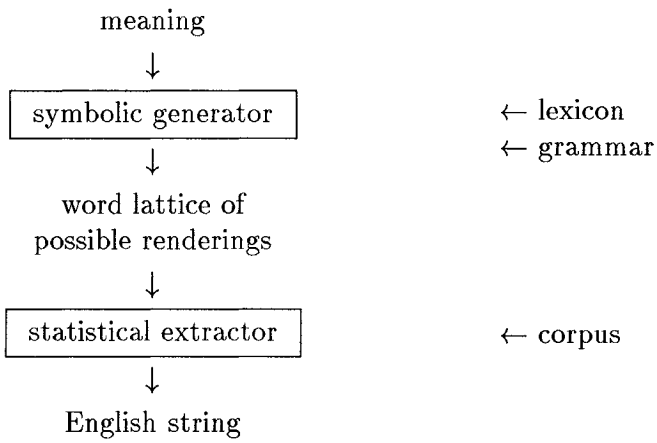


Figure 1.8: Combining Symbolic and Statistical Knowledge in a Natural Language Generator

directly to lattices. This meant that syntactic constraints were not localized and ended up being reduplicated many times. Scaling up to wide syntactic coverage was impractical under these circumstances. Second, Nitrogen offered no means of client control over the output. If an application knew it needed a definite reference to an object, or a passive voice sentence to provide coherent focus, there was no way to specify this or limit generation output accordingly. Third, the amount of overgeneration quickly led to lattices so large that they could not be searched optimally. A heuristic search provided a good approximation only for sentences of less than about 10 words in length. Finally, although the ngram model used by Nitrogen worked surprisingly well, it had some obvious weaknesses—including an inability to capture long distance dependencies or appropriately model syntactic structure. The quality of the output suffered as a result.

1.3.2.2 FERGUS

FERGUS (Bangalore and Rambow, 2000a; Bangalore, Rambow, and Whittaker, 2000b; Bangalore and Rambow, 2000c) addressed the syntactic modeling weakness of Nitrogen's ngram model in its approach to statistical generation. Leveraging the Tree-Adjoining Grammar (TAG) formalism (Joshi, 1987), it probabilistically assigned SuperTags to lexemes in a input. Then it used a TAG grammar to determine the order of constituents, except for adjuncts (words and phrases like "quickly", "yesterday", and "with a pen" that are grammatically optional). A lattice was used to encode the different possible positions and orders of adjuncts. Then FERGUS applied an ngram model, like Nitrogen, to select the most likely output. An empirical evaluation showed that FERGUS's approach significantly improved over a baseline model parameterized according to whether a child node in a tree occurred to the left or right of its parent.

These results were encouraging for statistical approaches to generation. However, the FERGUS system was not actually a practical system because of the type of input it used. It assumed dependency trees as input that already contained all the words of a desired sentence, including function words, and whose nodes were labeled with fully inflected lexemes. Client applications in generation do not typically have access to either of these pieces of information. Furthermore, the dependency relations were unlabeled, making the input inherently ambiguous. Finally, similar to Nitrogen, there was no means for any client control over the output. See Figure A.7 for a sample FERGUS input.

1.4 This Thesis

An ultimate goal for natural language generation is to develop a large-scale, robust, general-purpose system. General-purpose systems offer the advantages of re-usability and significant cost savings in application development. From a scientific point of view, the attempt to craft a general-purpose system advances broad objectives of greater knowledge and understanding by necessitating the discovery and use of general language models and engineering principles.

The contributions of this thesis are as follows.

1. I design a symbolic mapping rule formalism, set of mapping rules, and processing engine that allow greater flexibility in the input and efficiently map inputs to a packed set of candidate outputs.

2. I adopt a forest structure to represent the candidate outputs, and develop a bottom-up dynamic programming algorithm for statistical ranking that makes the search for an optimal solution tractable.

3. I perform the first large-scale empirical evaluation of a sentence realization system, measuring coverage, accuracy, and the ability to handle underspecification.

I also perform two other exploratory experiments. One addresses the problem of ambiguity in machine translation, and exploits HALogen's forest representation to develop an efficient approach for maintaining the ambiguity through the translation process. The second experiment explores the impact on output quality of using a statistical model of syntax.

1.5 Organization of Dissertation

The rest of this thesis is organized as follows. Chapter 2 describes the input to HALogen. Chapter 3 gives an overview of the system architecture and describes the symbolic generation module. Chapter 4 describes the forest representation and statistical ranker. It compares the forest representation to a lattice and shows experimental results on the time and space efficiency of each. Chapter 5 discusses the practical value of ngrams for generation, and Chapter 6 contains the results of the system evaluation on coverage and correctness. Chapter 7 describes the ambiguity maintenance experiments, and Chapter 8 an experiment with probabilistic syntax models. Chapter 9 concludes, reviewing the major contributions, discussing the impact of the research, and outlining limitations and areas for future work.

Chapter 2

HALogen's Input

HALogen is a successor to Nitrogen (Knight and Hatzivassiloglou, 1995b; Langkilde and Knight, 1998a). It is a hybrid symbolic and statistical system, with a two-stage architecture, like Nitrogen. In the first stage, symbolic knowledge is used to transform an input into a forest of possible expressions. In the second stage a statistical ranker computes the most likely expression using a corpus-based probabilistic model. This chapter describes the input used by HALogen.

2.1 Basic Structure

HALogen has inherited much of the structural form of its input from the Penman generation system. The structural form is a labeled feature-value structure that can also be thought of as a labeled directed graph. Graphs are geometrically convenient for representing meanings because they easily express unordered relationships between entities. FUF/Surge, RealPro, and (of course) KPML also use graph-type structures for their input. The syntax of HALogen's input is as follows:

```
INPUT -> FVSET
FVSET -> ( LABEL FVPAIR+)
FVPAIR -> feature VALUE
VALUE -> atomicval | LABEL | FVSET
```

An input consists of a label plus one or more feature-value pairs, surrounded by parentheses. Labels are arbitrary symbols used to identify a set of feature-value pairs. Features are represented as symbols preceded by a colon. They can express relationships between entities, or properties of a set of relationships or of an atomic value. The value of a feature can be an atomic entity, or a label, or recursively another labeled set of feature-value pairs.

The most basic input is a leaf structure of the form

```
(label / word-or-concept)   Examples: (m1 / "dog")  (m1 / |dog<canid|)
```

The slash is shorthand for the ":instance" feature (the most fundamental relation), and in logic notation this input might be written as Instance(m1, DOG). It also represents the semantic or syntactic head of a set of relationships. The value of the instance feature can be a word or a concept. A word can be enclosed in string quotes, and must usually be in root form. A concept should be a valid Sensus (Knight and Luk, 1994) symbol, which is a mnemonic name for a WordNet synset enclosed in vertical bars. A concept represents a unique meaning and can map to one or more words. The Sensus Ontosaurus Browser is accessible via the web at http://mozart.isi.edu:8003/sensus2, and can be used to look up concept names for words and synset classes. The basic input above can represent "the dog," "the dogs," "a dog," or "dog," etc.

2.2 Relations

There are two types of features: relations and properties. Relation features describe the relationship between the instance value and another content-bearing value. A content-bearing value can be a simple word or concept as in the instance example above, or a compound value composed of nested feature-value structures. Here are two examples that both express the idea, "The dog eats a meaty bone", the first using syntactic relations, and the second using more semantic relations. The value labeled 'b1' in each example is a compound value.

```
(e1 / eat                              (e1 / eat
    :subject (d1 / dog)                    :agent (d1 / dog)
    :object (b1 / bone                     :patient (b1 / bone
                :premod (m1 / meaty)))                 :premod (m1 / meaty)))
```

As shown in the dog-eat-bone examples just above, multiple relations can appear at a nesting level in the input. Most relations can only occur once at any given nesting level. The main exceptions are modifier and adverbial relations (adjuncts), which can occur any number of times. Relations are order-independent, which means that the order in which relations occur in the input does not affect the order in which their values occur in the output. A conditional exception is when the same relation occurs more than once in a nesting level. If generation is performed with the *permute-nodes* flag set to 'nil', then the values with the same relation will occur adjacent to each other in the output in same order that they appeared in the input. If the *permute-nodes* flag is set to 'T', they will occur adjacent to each other in an order determined by the statistical model.

Most of the relations that HALogen currently recognizes are listed in Table 2.1, grouped roughly according to their degree of abstraction—shallow syntactic, deep syntactic, and semantic. (The mapping rules in the symbolic generator, described in the next section, define which relations are recognized and their meaning. The mapping rules can easily be extended to also recognize non-linguistic and domain-specific relations.)

Shallow Syntactic
:SUBJECT, :OBJECT, :DATIVE, :COMPLEMENT, :PREDICATE,
:ANCHOR, :PREMOD, :POSTMOD, :WITHINMOD, :PREDET,
:TOPIC, :CONJ, :INTROCONJ, :BCPP, :COORDPUNC,
:LEFTPUNC, :RIGHTPUNC, :DETERMINER
Deep Syntactic
:LOGICAL-SUBJECT, :LOGICAL-OBJECT, :LOGICAL-DATIVE,
:LOGICAL-SUBJECT-OF, :LOGICAL-OBJECT-OF,
:LOGICAL-DATIVE-OF, :ADJUNCT, :CLOSELY-RELATED :QUESTION
:PUNC, :SANDWICHPUNC, :QUOTED ,
Semantic
:AGENT, :PATIENT, :RECIPIENT, :AGENT-OF, :PATIENT-OF,
:RECIPIENT-OF, :DOMAIN, :RANGE, :DOMAIN-OF, :SOURCE,
:DESTINATION, :SPATIAL-LOCATING, :TEMPORAL-LOCATING,
:ACCOMPANIER, :SANS, :ROLE-OF-AGENT, :ROLE-OF-PATIENT,
:MANNER, :MEANS, :CONDITION,:THEME,
:GENERICALLY-POSSESSED-BY, :NAME, :QUANT,
:RESTATEMENT, :GENERICALLY-POSSESSES
Miscellaneous
:INSTANCE, :OP, :PRO, :TEMPLATE, :FILLER

Table 2.1: Relation features

There is not actually any formal definition of a level of abstraction in HALogen to separate the different levels, in contrast to most other systems that recognize different strata of information. Instead, the grouping of relations into different levels of abstraction is for conceptual purposes. Relations at different levels of abstraction can be mixed in the same input and at the same level of nesting, according to the convenience of a client application.

A hierarchy of relations does exist in HALogen, however, due to the fact that deeper levels of abstraction are defined in terms of shallower levels of abstraction. Each mapping

from a deeper relation to a shallower relation captures an equivalence that exists at the shallower level. Rather than there being a fixed number of abstraction levels, in HALogen there is more a continuum of abstraction levels. The approach of using a continuum rather than a rigid stratification of abstraction levels not only increases the flexibility of the input from a client's perspective, it also increases the conciseness and modularity of the symbolic generator's mapping rules.

2.2.1 The Notion of Paraphrases

One significance of using a continuum of abstraction levels is that it provides a way to clarify some aspects of the general notion of a paraphrase, and simplify their definition. Paraphrasing has long been considered an important ability of a general-purpose generator, and has even been used as a qualitative indicator of a system's prowess—the assumption being that the more paraphrases a system could produce, the more powerful the system. However, different systems have applied the term "paraphrase" in different and somewhat inconsistent ways, usually with respect to an application-specific task. In general, the term has remained vague and not very well-defined .

HALogen's approach of treating abstraction as a continuum and defining deeper relations in terms of shallower ones alleviates some of these problems. A paraphrase is straightforwardly defined as one or more alternations sharing some equivalence that is encapsulated in a single representation using one or more relations at a deeper level abstraction. The deeper input should produce all and only those alternations that share the equivalency. A property feature can be used to distinguish between the alternations, and thus control or limit the generation of paraphrases when desired.

Using property features based on objective, well-founded principles enhances the general purposeness of a realization system and helps to overcome the problems of subjectivity that often plague deeper levels of abstraction. For example, some of the property features used to define deeper levels of abstraction in HALogen include voice, subject-position, and the syntactic category of a dominant constituent (ie. whether the phrasal head is a noun versus a verb). Examples of these are illustrated shortly. (Additional properties that it would be desirable to implement more fully in future work include features based on change of perspective, change of emphasis, change of relation, change of density, and clause movement, as described in (Dras, 1999).)

A result of this definition style is that equivalencies at higher levels of abstraction naturally produce a greater number of variations or paraphrases than those at lower levels of abstraction. The realizer has both the ability to generate a large number of paraphrases given an input at a deep level of abstraction, and to limit the variation in a principled way by specifying relevant property features or using a shallower level of abstraction. Additionally, the problem of whether different sentence variations really mean the same thing is avoided by definition, since there can simultaneously be a way to represent an equivalence and to distinguish between the possible paraphrases.

The following descriptions of HALogen's relations provide examples that illustrate HALogen's paraphrasing ability more concretely.

2.2.2 Semantic Relations

The semantic relations that HALogen recognizes are those that were defined and used in the GAZELLE machine translation project. Since that project is outside the scope of this

thesis, I will not describe the semantic relations in this thesis except for two aspects that are worth mentioning. Otherwise, readers interested in example usages of HALogen's semantic relations may refer to Appendix C.1.

One interesting aspect at this level of abstraction is HALogen's ability to paraphrase the ideas of possibility, ability, obligatoriness, etc., as modal verbs using the :domain relation. (Modal verbs in English are the words "may", "might", "can", "could", "will", "would", "should", and "must".) However, by having access to other syntactic structures as well to express these ideas, HALogen can generate fluent sentences when a :domain relation is nested inside another :domain, and any combination of :polarity is applied to inner and outer domain instances, even though modal verbs themselves cannot be nested. For example, the following sentence is not grammatical: "You may must eat chicken." However, it can be paraphrased as, "You may be required to eat chicken." Appendix C.2 provides a variety of examples of the :domain relation and its expression using modal verbs.

Another interesting aspect is HALogen's definition of the :agent and :patient relations to capture the equivalence between alternations like the following: "Napoleon invaded France," and "Napolean's invasion of France." The :agent and :patient relations are used to represent the similarity between expressions whose semantic head is realized as a noun versus as a verb. More formally, :agent can map to either :logical-subject, or :generalized-possession-inverse (which can produce a possessive phrase using an "'s" construction). The :patient relation maps to either :logical-object, or to :adjunct with a prepositional anchor like "of".

2.2.3 Deep Syntactic Relations

The deep syntactic relations capture equivalencies that exist at the shallow syntactic level. For example, the :logical-subject, :logical-object, and :logical-dative relations capture the similarity that exists between sentences that differ only in active versus passive voice. More concretely, the two sentences, "The dog ate the bone" and "The bone was eaten by the dog" would both be represented at the deep syntactic level as:

```
(e1 / eat
    :logical-subject (d1 / dog)
    :logical-object (b1 / bone))
```

The :voice feature could be added to distinguish the two variations in output. With "active" voice, :logical-subject would map to :subject and :logical-object would map to :object. In contrast, with "passive" voice, :logical-object would map to :subject and :logical-subject would map to :adjunct with the addition of a prepositional anchor "by". (See Figure 3.2 for an illustration.

The :adjunct relation at the deep syntactic level maps to either :premod, :postmod, or :withinmod at the syntactic level, abstracting away from ordering information to capture the similarity that all three syntactic relations are adjuncts. The :closely-related relation can be used to represent the uncertainty of whether a particular constituent is a required argument of a verb, or an optional adjunct. The :question relation consolidates in one relation the combination of three syntactic features that can sometimes be independent.

For example, the following two inputs are equivalent, and may be used to represent, "What did the dog eat?"[1]

```
(e1 / eat                          (e1 / eat
    :question (b1 / what)              :topic (b1 / what)
    :subject (d1 / dog))               :subject (d1 / dog)
                                       :subject-position post-aux
                                       :punc question_mark)
```

The :punc relation generalizes the :leftpunc, :rightpunc, and :sandwichpunc relations. The :sandwichpunc relation is itself a generalization of :leftpunc and :rightpunc. For example, ":punc $question_mark$" maps to ":rightpunc $question_mark$". The three feature-value pairs ":punc $double_quotes$", ":quoted +", and ":sandwichpunc $double_quotes$" are all synonyms for the combination ":leftpunc $double_quotes$:rightpunc $double_quotes$".

2.2.4 Shallow Syntactic Relations

Most of the shallow syntactic relations shown in Table 2.1 should be self-explanatory. However, a few deserve some comment. For the most part, the Penn Treebank annotation conventions (Marcus, Santorini, and Marcinkiewicz, 1993) motivated these relations, including :subject, :object, :dative, :complement, :predicate (marked with -PRD in the Treebank, :predet (corresponds to PDT part-of-speech label in the Treebank), :topic (marked -TPC in the Treebank), :conj (corresponds to CC and CONJ labels in Treebank), and :determiner (corresponds to DT part-of-speech label).

[1]This representation of questions is nonstandard, but for generation purposes it is very efficient because it utilizes an equivalency with the :topic relation and handles the surface production of questions without requiring any new mapping rules. More conventional representations of questions can be mapped to this representation using HALogen's mapping rule formalism. Furthermore, this representation can be straightforwardly derived from Penn Treebank-style annotation of questions.

However, the :predet relation in HALogen has been broadened to include any head noun modifier that precedes a determiner. The :topic relation additionally includes question words/phrases, since these exhibit the same kind of shifted pattern as those marked with -TPC in Treebank. For example, the phrase "Hello!" in the sentence, " 'Hello!' she said" would be marked -TPC in Treebank because it has been shifted from its default position after the verb "said."

The :anchor relation represents both prepositions (labeled IN and sometimes TO in Treebank) and function words like "that", "who", "which", etc., that can be viewed as explicitly expressing the relation that holds between two content-bearing constituents of a sentence.

The remaining syntactic relations that deserve comment all relate to coordinated phrases. Coordinated phrases can be represented in either of two ways in HALogen, depending on convenience, as shown here:

```
(c1 / and                    (c1 / (a1 / apple)
    :op (a1 / apple)             / (b1 / banana)
    :op (b1 / banana))           :conj (d1/ and))
```

The first is mapped to the second. The representation of coordinated phrases in HALogen combines elements of dependency notation and phrase-structure notation. At the lowest level of abstraction, coordination is signaled by the presence of more than one instance relation. Besides :conj, the relations that are optionally involved in coordinated phrases include :coordpunc, :bcpp, and :introconj. If not specified, :coordpunc defaults to "comma" most of the time, but to "semicolon" when coordinated phrases already contain commas. However, it can be specified to be words like "or" (eg: "apples or oranges or

bananas"), or specified to be other types of punctuation. The relation :bcpp is a boolean property that controls whether the value specified by :coordpunc occurs immediately before the conjunction (for example: "a, b, and c" if ":bcpp +" is specified, versus "a, b and c". It defaults to negative unless more than two entities are being coordinated. Finally, :introconj is used to represented the initial phrases that occur in paired conjunctions, such as "not only ... but", and "either ... or".

Relations in HALogen can have aliases, to accommodate the varying nomenclature of different applications. Many of the relations in HALogen do already have aliases–e.g., you may refer to :AGENT as :SAYER or :SENSER, and to :DATIVE as :INDIRECT-OBJECT. An alphabetical list of relations with example usages is shown in Appendix C.1 along with some already-defined aliases.

2.3 Partial Ordering on Adjuncts, and Scope

HALogen departs to a small extent from Penman-style input in allowing Instances to be compound nodes, not just atomic values. This serves two purposes: it provides a flexible means of controlling adjunct generation, and it allows the representation of scope. These inputs illustrate the first case:

```
(c1 / flight
    :postmod (l1 / "Los Angeles"
                 :anchor "to")
    :postmod (m1 / "Monday"
                 :anchor "on"))
```

```
(c1 / (f1 / flight)
        :postmod (l1 / "Los Angeles"
                       :anchor "to"))
  :postmod (m1 / "Monday"
                 :anchor "on"))
```

Both inputs are interpreted by HALogen to have equivalent meanings. However, the second one constrains the set of possible outputs, which an application might wish to

do for rhetorical purposes (eg. to generate a response that parallels a user utterance). The first input can generate both "a flight to Los Angeles on Monday" and "a flight on Monday to Los Angeles" (as long as the flag *permute-nodes* is set to true), while the second input is constrained so that the latter will NOT be generated. The nesting of the Instance relation specifies a partial order on the set of relations so that those in the outer nest are ordered more distantly from the head than those in the inner nest. (Setting the *permute-nodes* flag to false could accomplish the same thing. HALogen offers both means of controlling adjunct ordering for the sake of convenience. However, the nesting method is more flexible, because it can be used to specify a partial order, while the *permute-nodes* flag in effect specifies a full ordering.)

The semantic notion of scope can be added to a nested feature-value set via the :unit feature, as shown in the next example. The example can generate "the popular University of Southern California". The :unit feature together with the nested structure indicates that the adjunct "popular" modifies the whole phrase "University of Southern California", not just "University".

```
(c1 / (u1 / "University"
          :postmod (c1 / "California"
                       :adjunct (s1 / "Southern")
                       :anchor "of")
          :unit +)
    :adjunct (p1 / popular))
```

2.4 Input disjunctions

One last structural detail: A meta-level *OR* can be used to express an exclusive-or relationship between a group of inputs or values. Semantically, it represents ambiguity,

or a choice between alternate expressions. It can also be viewed as a type of underspecification. In HALogen, the choice between alternatives will eventually be made by the statistical ranker.

Here is an example that represents two semantic interpretations of the clause, "I see a man with a telescope," as well as a choice between the words "see", and "watch", and an ambiguity about whether John said it or Jane sang it.

```
(*OR* (a1 / say
          :agent (j1 / "John")
          :saying (*OR* (s1 / (*OR* see watch)
                               :agent I
                               :patient (m1 / man
                                           :accompanier (t1 / telescope)))
                        (s2 / see
                               :agent I
                               :patient (m2 / man)
                               :instrument (t1 / telescope))))
      (a2 / sing
          :agent (j2 / "Jane")
          :saying (*OR* s1 s2)))
```

2.5 Property Features

The other main type of features are properties. Table 2.2 lists those that HALogen uses with their possible values. In contrast to relations, they have only atomic values, and describe linguistic properties of an instance. They do not generally need to be specified in the input, unless one wants to override the defaults that HALogen provides.

Here, for example, is a noun concept specified to be plural:

```
(m2 / |dog<canid|
   :number plural)
```

VERB	
:TAXIS	perfect, none
:ASPECT	continuous, simple
:VOICE	active, passive
:MOOD	infinitive, to-infinitive, imperative, present-participle, past-participle, indicative
:TENSE	present, past
:MODAL	should, would, could, may, might, must, can, will
:PERSON	s (3s), p (1s 1p 2s 2p 3p)
:SUBJECT-POSITION	default, post-aux, post-vp
NOUN	
:CAT1	common, proper, pronoun
:NUMBER	singular, plural
ADJECTIVE	
:CAT1	comparative, superlative, cardinal, possessive, none
ADVERB	
:CAT1	comparative, superlative, negative ("not"), wh, none
GENERAL	
:LEX	(root form of a word as a string)
:SEM	(a SENSUS Ontosaurus concept)
:CAT	vv, nn, jj, rb, etc.
:RHS	(inflected form of a word as a string)
:POLARITY	+, -
:GAP	+, -

Table 2.2: Property features

The :number property in this case narrows the meaning to "the dogs," or "dogs." Without the :number specification, the representation would be ambiguous between one or multiple dogs, and the choice would be left up to the statistical ranker.

To ease the information burden on a client application, realizers often allow inputs to be underspecified. However, most realizers handle the underspecification by supplying rigid hand-coded defaults and are limited in the variety of features that can be underspecified—usually to just syntactic properties. In contrast, a corpus-based statistical model provides preferences for all kinds of linguistic decisions, at a finer-grain, without manual effort.

The choice between singular and plural for a noun, for example, is often semantic rather than syntactic. That HALogen will make such semantic decisions as well as syntactic ones, via statistical ranking, distinguishes it from other realizers. Of course, the goodness of such decisions depends on the nature of the statistical model. That HALogen will make semantic decisions does not imply that it should, or that it would do it well. However, since the distinction between syntax and semantics is often blurred, and since corpus statistics can often capture syntax preferences well, the ability to handle the underspecification of any property feature is an advantage. Chapter 5 discusses this further using a variety of examples.

The ":number plural" feature for nouns triggers a morphological inflection on the base form of a word ("dog" $--> $ "dogs"), but some features do not—instead serving only a descriptive purpose. The :cat feature is like this, as well as the :cat1 feature for nouns, and the "possessive", "cardinal", "negative", and "wh" values of the :cat1 feature for adjectives and adverbs (words like "his", "five", "not", and "when", respectively).

Descriptive properties are extensively used in traditional symbolic realizers to constrain the variety of phrases that can be generated to those that are grammatical. HALogen makes very little use of descriptive features for this purpose, however, instead allowing overgeneration and leaving it to the statistical language model to apply softer, more graded preferences.

Another role of property features is to generate auxiliary function words. Many of the verb properties serve this purpose—including :modal, :taxis, :aspect, and :voice. In combination with the verbal :mood property, these four features can be used to generate the entire range of auxiliary verbs used in English.

The following example illustrates a possible use of verbal properties by explicitly specifying values for each possible property. The resulting output is also shown.

```
(e1 / "eat"
    :mood indicative
    :modal "might"
    :taxis none
    :aspect continuous
    :voice active
    :person 3s
    :subject (j1 / "Jane")
    :subject-position default
    :object (i1 / "ice cream"))

OUTPUT: "Jane might be eating ice cream."
```

The :taxis feature generates perfect tense when specified ("might have been eating"). By default only ":taxis none" is generated. The :aspect feature generates continuous tense when specified, as shown above. (If not specified, the above example would have generated "Jane might eat ice cream.") By default, only ":aspect simple" is generated. The :voice feature can be either "passive" or "active". If it had been "passive" in the

	Singular	Plural
First	(I) eat	(we) eat
Second	(you) eat	(you) eat
Third	(he, she, it) eats	(they) eat

Table 2.3: Verb conjugation table

example above, it would have generated "Ice cream might be eaten by Jane." The modal feature should be self-explanatory. By default it is assumed to have value "none".

The :person feature has six primary values, corresponding to each possible combination of person (first, second, and third) and verbal number (singular or plural), as illustrated by the following chart.

Since all verbs except "be" have a distinct value for only third-person singular, the :person feature value can be abbreviated as just "s" (for "3s") or "p" (for all others). By default HALogen generates all unique inflections when the :person feature is not specified in the input.

Finally, the :subject-position feature has two non-default values, "post-aux" and "post-vp". The post-aux value is used to produce questions and some inverted sentences, such as "Might Jane be eating ice cream?" and "Marching down the street was the band" (using also the :topic relation with the main verb). The post-vp value is usually used in combination with the verb "say" and its synonyms, together with the :topic relation, which shifts verbal constituents to the front of the sentence. For example: " 'Hello!', said John."

Appendix B contains additional examples as well as a table that lists the output of every possible combination of the verbal properties and values. The table uses the verb

"eat" as an example, but also lists the output of the verbal properties when applied to the main verb "be", which is a uniquely irregular case.

2.6 Completeness

A major difference between HALogen and other sentence realizers is that HALogen's input is intended to be expressive enough to provide complete coverage of English, without being as concerned about disallowing ungrammatical sentences as other realizers are. The ranking capability of the statistical model enables this shift in priorities. Broader coverage is achieved by designing an input (and set of mapping rules) that are simpler and more regular than that used by other generators.

Figure 2.1 illustrates metaphorically the effect of this. The gray curvy area represents valid English, and the black ellipsis represents the set of phrases that can be produced with HALogen's input representation and mapping rules. The set of outputs that can be generated by HALogen is significantly greater than the set of valid English sentences, meaning that HALogen overgenerates and can sometimes produce ungrammatical output. However, the regularity of the input and mapping process greatly simplifies the generation task, analogous to the difference between a polynomial and exponential equation. The usefulness of this approach depends on the effectiveness of statistical ranking in preferring valid English over nonsensical output.

Empirically verifying completeness is not straightforward because of the irregularity and complexity of natural language. However, the evaluation described in Chapter 6

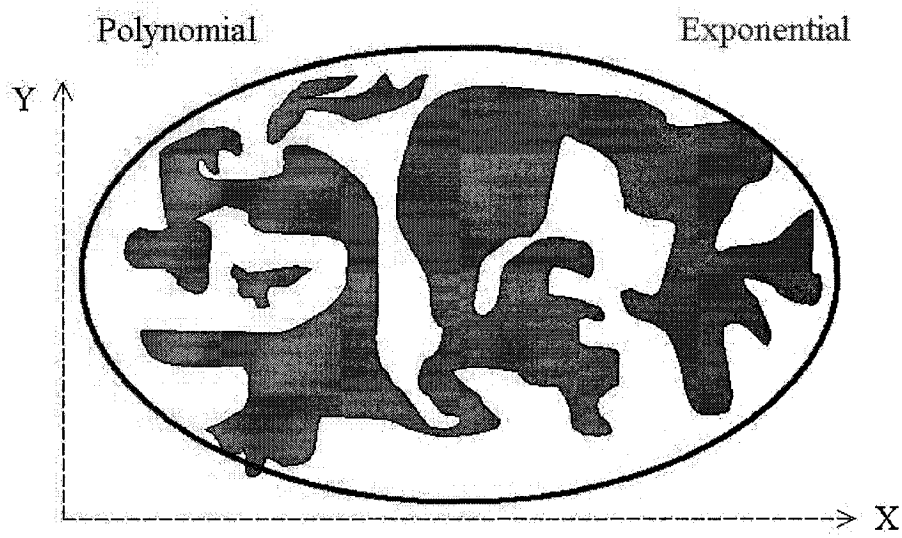


Figure 2.1: Metaphorical Illustration of the Tradeoff between Regularity and Goodness of Fit

attempts to estimate completeness/coverage using 2400 newspaper sentences from the Penn Treebank corpus as a blind test set.

Chapter 3

HALogen Symbolic Generator

3.1 Introduction

This chapter describes HALogen's symbolic generator, which maps inputs to a set of possible expressions in a compact form as illustrated in the first half of Figure 3.1. The main tasks that the symbolic generator performs are

- mapping higher-level relations and concepts to lower-level ones (and ultimately to the lowest level of abstraction),
- filling in details not specified in the input,
- determining constituent order, and
- performing morphological inflections.

The symbolic generator uses simple lexical, morphological and grammatical knowledge bases to perform these tasks. Many of the linguistic decisions that need to be made in realizing the input are actually delayed until the statistical ranking stage. The symbolic

generator instead itemizes the alternatives, packing them concisely into an intermediate data structure. The chapter is organized as follows: first I describe the knowledge sources used by HALogen, which include a Wordnet-based dictionary, a closed-class lexicon, an application-specific lexicon, morphological inflection tables, and input mapping rules (somewhat analogous to grammar rules). The most significant of these is the set of mapping rules, which form the core of the symbolic generator. Second, I briefly introduce the *forest* data structure used to store the intermediate results from the symbolic generator. (The next chapter compares a forest structure to two alternatives: a simple list, and a lattice (which was used by Nitrogen). Next, I describe the engine of the symbolic generator and how the generation process proceeds. Finally, I discuss how HALogen's approach dramatically reduces the need for hand-crafted knowledge. Portions of this chapter were previously published in (Langkilde and Knight, 1998a).

3.2 Lexical Knowledge

3.2.1 Sensus Concept Ontology

The Sensus concept ontology is a WordNet-based (Miller, 1990) hierarchy of word meanings segregated at the top-most level into events (verbal concepts), objects (nominal concepts), qualities (adjectives), and adverbs. Each concept represents a set of synonyms, referred to as a synset. The ontology consists of a list of 110,000 tuples of the form:

```
(<word> <part-of-speech> <rank> <concept>)
Examples:
 ("eat"     VERB 1  |eat,take in|)
 ("eat"     VERB 2  |eat>eat lunch|)
 ("take in" VERB 14 |eat,take in|)
```

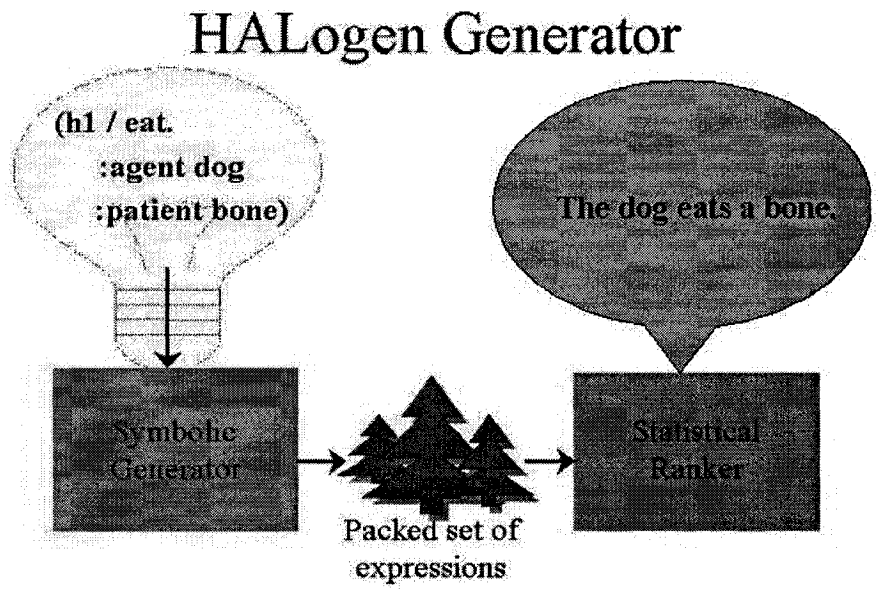


Figure 3.1: HALogen

The <rank> field orders the concepts by sense frequency for the given word, with a lower number signifying a more frequent sense.

Like other types of knowledge used in HALogen, the lexicon is very simple. It contains no information about features like transitivity, sub-categorization, gradability (for adjectives), or countability (for nouns), etc. Such features are needed in other generators to produce correct grammatical constructions. In HALogen, the statistical post-processor instead more softly (and robustly) ranks different grammatical realizations according to their likelihood.

At the lexical level, several important issues in word choice arise. WordNet maps a concept to one or more synonyms. However, some words may be less appropriate than

others, or may actually be misleading in certain contexts. An example is the concept `|sell<cozen|` to which the lexicon maps the words "betray" and "sell." However, it is not very common to use the word "sell" in the sense of "A traitor sells out his friends." In the sentence "I cannot `|sell<cozen|` their trust" the word "sell" is misleading, or at least sounds very strange; "betray" is more appropriate.

This word choice problem occurs frequently, and HALogen deals with it by taking advantage of the word-sense rankings that the lexicon offers. According to the lexicon, the concept `|sell<cozen|` expresses the second most frequent sense of the word "betray," but only the sixth most frequent sense of the word "sell." To minimize the lexical choice problem, I have adopted a heuristic of associating with each word a preference score according to the formula, $weight = 1/e^{(rank-1)}$, and allowing the statistical ranker to choose the best alternative.

This seems to work well in practice. In this way, HALogen avoids technically accurate but misleading English. Of course, there are other possible alternatives, such as using a method like Bayes' Rule and probabilities computed from a corpus such as SEMCOR.

Another issue in word choice relates to the broader issue of preserving ambiguities. In source language analysis, it is often difficult to determine which concept is intended by a certain word. HALogen allows several concepts to be listed together in a disjunction. For example,

```
(m6 / (*OR* |sell<cozen| |cheat on| |bewray| |betray,fail| |rat on|))
```

The lexical lookup will attempt to preserve the ambiguity of this `*OR*`. If it happens that several or all of the concepts in a disjunction can be expressed using the same word,

then the lookup will return only that word or words in preference to the other possibilities. For the example above, the lookup returns only the word "betray." This also reduces the complexity of the set of candidate sentences. (Chapter 7 contains a fuller discussion of an approach HALogen offers for addressing the general problem of preserving ambiguities in translation.)

3.2.2 Closed-Class Lexicon

HALogen's closed-class lexicon consists of entries in the following form:

```
(:cat <cat>  :orth <orthography> :sense <sense>)
Examples:
(:cat ADJ  :orth "her" :sense 3s_fem_possessive)
(:cat CC  :orth "and" :sense cc_0)
(:cat DT  :orth "a" :sense indef_det)
(:cat IN  :orth "with" :sense with)
(:cat MD  :orth "can" :sense modal_verb)
(:cat NOUN  :orth "he" :sense 3s_pronoun)
(:cat PDT  :orth "all" :sense pdt_0)
(:cat RB  :orth "when" :sense wrb_2)
(:cat RP  :orth "up" :sense rp_27)
(:cat WDT  :orth "which" :sense wdt_clocla)
(:cat UH  :orth "ah" :sense uh_0)
(:cat |-COM-|  :orth "," :sense comma)
```

In some cases, the sense has been named so that it can be used to specify the closed-class item in a more language-neutral way in an input. Table 3.1 lists the number of entries in each category.

3.2.3 Application-Specific Lexicon

HALogen also allows for a user-defined lexicon that can be used to customize HALogen for a specific application. This lexicon is consulted before the other lexicons, to allow an

CAT	NAME	COUNT
ADJ	possessive adjective	8
CC	conjunction	22
DT	determiner	24
IN	preposition	228
MD	modal verb	8
NOUN	pronoun	41
PDT	predeterminer	7
RB	adverb pronoun	12
RP	particle	32
WDT	wh-determiner	5
UH	interjection	40
–	punctuation	72

Table 3.1: Summary of entries in HALogen's closed-class lexicon

application to easily override the knowledge bases HALogen provides, without having to actually change them.

The user lexicon consists of entries in the following form:

```
(<concept> <template-expansion>)
Example:
 (|morning<antemeridian| (*OR* (:cat NN :lex "a.m") (:cat NN :lex "morning")))
```

3.3 Morphological Knowledge

The lexicon contains words in their root form, so morphological inflections such as plural nouns and past tense verbs must be generated. The system also contains tables for performing derivational morphology, such as adjective→noun and noun→verb (ex: "translation"→"translate"). This gives the generator greater paraphrasing power and more flexibility in expressing an input. It is also useful for solving problems of syntactic divergence in machine translation.

Both kinds of morphology are handled the same way. Pattern rules and exception tables are merged into a single, concise knowledge base. Here, for example, is a portion of the table for pluralizing nouns:

```
("-child" "children")
("-person" "people" "persons")
("-a" "as" "ae")       ; formulas/formulae
("-x" "xes" "xen")     ; boxes/oxen
("-man" "mans" "men") ; humans/footmen
("-Co" "os" "oes")
```

The last line means: if a noun ends in a consonant followed by "-o," then compute two plural forms, one ending in "-os" and one ending in "-oes," and store both possibilities for the statistical ranker to choose between later. Deciding between these usually requires a large word list. However, the statistical extractor already has a strong preference for "photos" and "potatoes" over "photoes" and "potatos," so there is really no need to create such a list. Here again corpus-based statistical knowledge greatly simplifies the task of symbolic generation.

Derivational morphology raises the issue of meaning shift between different part-of-speech forms (such as "depart"→ "departure"/"department"). Errors of this kind in the final output have not been too frequent in practice. However, if they occur they can be corrected by adding exceptions to the morphology tables.

The following table lists the sets of morphology rules that HALogen provides, together with the number of entries in each, to emphasize the reduction in hand-coded knowledge that use of a corpus-based statistical knowledge source enables.

MORPHOLOGY TYPE	PATTERN COUNT
comparative	15
superlative	15
root comparative	10
root superlative	10
present tense	15
present participle	12
plural	54
past irregular	286
adjective to adverb	6
adjective to verb	5
adjective to noun	24
noun to adjective	31
verb to noun	44
noun to verb	95
ordinal	12
root	432

Table 3.2: Summary of HALogen's morphology tables

3.4 Mapping Rules

The core of the symbolic generator is the set of rules that map inputs into a packed intermediate data structure for subsequent statistical ranking. They are organized around patterns that occur at the top-level of an input—primarily the relations between constituents. Currently, there are about 255 mapping rules grouped into four sets: general, clausal, nominal, and other. There are four main kinds of mapping rules: recasting, ordering, filling, and morphing.

3.4.1 Recasting Rules

Recasting rules map one relation to another. They are used to map deeper relations to shallower ones, such as semantic relations into syntactic ones. They can be used to map

```
RULE:
((x1 :logical-subject)
 (x0 :rest)
 (x4 :voice passive)
  ->
 (1.0 -> (x0 :postmod (x1 :anchor "by"))))

ENGLISH INTERPRETATION:
IF top level contains logical-subject feature,
   and also contains voice feature with value passive,
THEN map logical-subject to postmod and add feature anchor with value by.

ILLUSTRATION:
( / "serve"                ==>     ( / "serve"
  :voice passive                     :voice passive
  :logical-object <cuisine>          :logical-object <cuisine>
  :logical-subject  <venue>)         :postmod ( / <venue>
                                                  :anchor by ))
```

Figure 3.2: Recasting rule, English gloss, and illustration

:agent into :logical-subject or :adjunct, for example. They make it possible to localize constraints. As a result, the rule set as a whole is more modular and concise. Recasting rules facilitate a continuum of abstraction levels from which an application can choose to express an input. Recasting rules are also a tool that an application can use to customize HALogen, if desired. By adding additional rules, the generator can be extended to map non-linguistic or domain-specific relations into those already recognized by HALogen.

The recasting mechanism also makes it possible to handle non-compositional aspects of language. One area in which this mechanism is used is in the `:domain` rule. Take for example the sentence, "It is necessary that the dog eat." It is sometimes most convenient to represent this as:

```
(m8 / |obligatory<necessary|
   :domain (m9 / |eat,take in|
              :agent (m10 / |dog,canid|)))
```

```
RULE:
((x2 :logical-object)
 (x1 :passive-subject-role logical-object)
 (x0 :rest)
 (x4 :voice passive)
  ->
 (1.0 -> (x0 :subject x2)))

ENGLISH INTERPRETATION:
IF top level contains logical-object feature,
   and also contains voice feature with value passive,
   and also contain passive-subject-role feature with value logical-object
THEN map logical-object to subject.

ILLUSTRATION:
( / "serve"                          ==>  ( / "serve"
  :voice passive                             :voice passive
  :logical-object <cuisine>                  :subject <cuisine>)
  :passive-subject-role logical-object       :postmod ( / <venue>
  :postmod ( / <venue>                                  :anchor by ))
             :anchor by ))
```

Figure 3.3: Another recasting rule, English gloss, and illustration

and at other times as:

```
(m11 / |have the quality of being|
   :domain (m12 / |eat,take in|
              :agent (d / |dog,canid|))
   :range (m13 / |obligatory<necessary|))
```

but we can define them to be semantically equivalent. Both are accepted, and the first is automatically transformed into the second.

Other ways to say this sentence include "The dog is required to eat," or "The dog must eat." However, the grammar formalism cannot directly express this, because it would require inserting the result for `|obligatory<necessary|` *within* the result for m9 or m12—and the formalism can only concatenate results. The recasting mechanism solves this problem, by recasting the above input as:

```
(m14 / |eat,take in|
   :modal (m15 / |obligatory<necessary|)
   :agent (m16 / |dog,canid|))
```

which makes it possible to form these sentences through simple concatenation of the constituents. The syntax for recasting the first input to the second is:

```
((x2 :domain)
 (not :range)
 (x0 (:instance /))
 (x1 :rest)
  ->
(1.0 -> (/ |hqb| :domain x2 :range (/ x0 :splice x1))))
```

and for recasting the second into the third:

```
((x2 :domain)
 (x3 :range)
```

```
RULE:
((not :voice)
 (x1 :rest)
 (x2 (:logical-subject :logical-object :logical-dative))
 ->
 (1.0 -> (x1 :voice active))
 (1.0 -> (x1 :voice passive)))

ENGLISH INTERPRETATION:
IF top level contains logical-subject, logical-object,
   or logical-dative feature, but does not contain voice feature,
THEN make copies of input and to one add voice feature with value active,
     and to another add voice feature with value passive.

ILLUSTRATION:
                                      ( / "serve"
                                        :voice active
                                ==>     :logical-subject <venue>
( / "serve"                             :logical-object <cuisine> )
   :logical-subject <venue>
   :logical-object <cuisine> )        ( / "serve"
                                ==>     :voice passive
                                        :logical-subject <venue>
                                        :logical-object <cuisine> )
```

Figure 3.4: An filling rule, English gloss, and illustration

```
 (x0 (:instance /))
 (x1 :rest)
 ->
 (1.0 -> (x2 :semmodal (/ x3) :splice x1))
```

3.4.2 Filling Rules

A filling rule adds missing information to underspecified inputs. This type of rule tests whether a particular feature is absent. If so, it will generate one or more copies of the input, one for each possible value of the feature, and add the feature-value pair to the copy. Each copy is then independently circulated through the mapping rules.

```
RULE:
((x1 :subject)
 (x2 :subject-position default)
 (x0 :rest)
  ->
 (1.0 -> (x1 :nominalize +) (x0 :subject-position taken)))

ENGLISH INTERPRETATION:
IF top level of input contains subject relation,
   and also contains the subject-position feature with value "default"
THEN split input and place value of subject in linear order before the rest
     of the input.

ILLUSTRATION:

( / "serve"                                        /\
  :voice active                ==>                /  \
  :subject <venue>                               /    \
  :object <cuisine> )                           /      \
                                               /        \
                                        <venue>  ( / "serve"
                                                   :voice active
                                                   :object <cuisine> )
```

Figure 3.5: An ordering rule, English gloss, and illustration

3.4.3 Ordering Rules

Ordering rules assign a linear order to the values whose features matched with the rule. Ordering rules typically match with syntactic features at the lowest level of abstraction. An ordering rule splits an input apart into several pieces. The values of the features that matched with the rule are extracted from the input and independently recirculated through the mapping rules. The remaining portion of the original input continues to circulate through the rules where it left off. When each of the pieces finishes circulating through the rules, their results are concatenated in the designated linear order.

```
RULE:
((x4 :lex)
 (not :rhs)
 (x1 :person (s 3s))
 (x3 :rest)
 (x2 :tense present)
 (x6 morph-3singpres x4)
 ->
 (1.0 -> (x6 :splice x3 :person s)))

ENGLISH INTERPRETATION:
IF input contains lex, person=3s and tense=present features,
   but not inflected form,
THEN compute inflection.

ILLUSTRATION:
( :lex "eat"          ==>   (:lex "eat"
  :person 3s                  :person s
  :tense present)             :tense present
                              :rhs "eats")
```

Figure 3.6: An morphological inflection rule, English gloss, and illustration

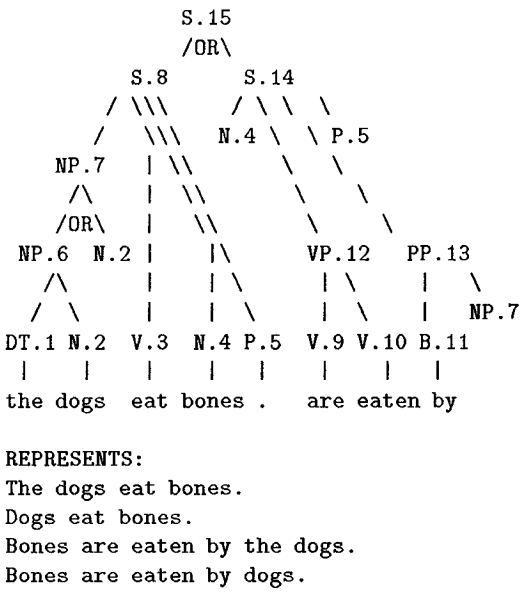
3.4.4 Morphological inflection rules

A morph rule produces a morphological inflection of a base lexeme, based on the property features associated with it.

3.5 Forest Representation

The results of symbolic generation are stored in a *forest* structure. A forest compactly represents a large, finite set of candidate realizations as a *non-recursive context-free grammar*. It can also be thought of as an AND-OR graph where AND nodes represent a sequence of constituents, and OR nodes represent a choice between mutually exclusive alternatives. A forest does not necessarily encode information about linguistic structure of a sentence, but it may.

Figure 3.7 presents a graphical illustration of a simple forest, its internal representation, and a list of the different sentences it represents.




```
REPRESENTS:
The dogs eat bones.
Dogs eat bones.
Bones are eaten by the dogs.
Bones are eaten by dogs.
```

```
TOP --> S.15
S.15 --> S.8
S.15 --> S.14
S.8 --> NP.7 V.3 N.4 P.5
NP.7 --> NP.6
NP.7 --> N.2
NP.6 --> DT.1 N.2
DT.1 --> "the"
N.2 --> "dogs"
V.3 --> "eat"
N.4 --> "bones"
P.5 --> "."
S.14 --> N.4 VP.12 PP.13 P.5
VP.12 --> V.9 V.10
V.9 --> "are"
V.10 --> "eaten"
PP.13 --> B.11 NP.7
B.11 --> "by"
```

Figure 3.7: A Forest: graphical and textual representations with the set of sentences they represent

In HALogen, nodes of a forest are labeled with a symbol consisting of an arbitrary alpha-numeric sequence, then a period, and a number. The alpha-numeric sequence has no significance for HALogen, but can be used to improve readability of the forest. The number identifies a node and must be unique. The TOP node is special, and has simply the label "TOP".

A forest consists of two types of rules, leaf and non-leaf. A leaf rule has only one item on its right-hand side: an output word enclosed in double quotes. A non-leaf node can have any number of items on its right-hand side, all of which are labels for a sequence of child nodes. The presence of multiple rules with the same left-hand side label represents a disjunction, or an OR node.

Alternatively, HALogen also uses a third type of rule to represent OR nodes to simplify implementation. This third type of rule has identical structure to a non-leaf sequence node, except that it contains an OR-arrow symbol ("OR$\longrightarrow$") in place of a simple arrow. This alternate representation of OR nodes is referred to in HALogen as a GF representation, while the first form is called a PF representation. In a GF representation, a label appears on the left-hand side of

a rule only once. In a GF representation, the four rules in Figure 3.7 that represent the two OR-nodes would be represented textually using only two rules:

```
S.15  OR--> S.8  S.14
NP.7  OR--> NP.6  N.2
```

3.6 Realization Algorithm

The symbolic generator proceeds by comparing the top level of an input with each of the mapping rules in turn. The first rule that matches is executed. Rules usually execute successfully by design. The mapping rules transform or decompose the input and recursively process the new input(s). If there is more than one new input, each is independently recirculated through the rules. When matching has finished, base input fragments are converted into elementary forests and then recombined according to the specification of the respective mapping rules as each recursive loop is exited.

When a rule finishes executing, the result is cached together with the input fragment that matched it. Since HALogen extensively overgenerates by design, this caching dramatically impacts efficiency. Each time a matched rule transforms or decomposes an input, the new sub-input(s) are matched against the cache before being recursively matched against the rule set.

If execution of a particular rule is not successful, the original input continues matching against the rest of the rule set, as long as the appropriate flag (rematchspl) in the generator is set to true. Rematching has the same effect that backtracking does in other realization systems, but the mechanism in HALogen is very simple and straightforward. Rematching takes advantage of the cache in the same way first-time matching does. If no match or rematch exists, generation of that particular sub-input fails. (In practice, there is little need for rematching, and so very few rules exist in the rule set that would successfully rematch an input when a previous rule has failed.)

Rules must be ordered so that those dealing with higher levels of abstraction come before those dealing with lower levels. The lowest level of abstraction is the ordering rules. Among ordering rules, those that place constituents farther from the head come before those that place constituents closer to the head. As rule matching continues, ordering rules extract constituents from the input, until only the head is left. Rules that perform morphological inflections thus appear last. Filling rules come before any rule whose left-hand-side matching conditions might depend on the missing feature.

Dependencies between relations thus govern the overall ordering of rules in the rule set. The constraints on rule order define a partial-order, so that within these constraints, it does not matter what order the rules appear in—the output will not be affected. The final resulting forest is given to the statistical ranker for the next stage of processing.

HALogen's mapping rules are hand-written, but developed in part by using the *tgrep* tree grep program distributed with the Treebank. The tgrep program indexes all sections of Treebank, including section 23, the section used for testing in the experiments described later. However, the tgrep program does not indicate the section number of any trees that are retrieved, and appears to display matching trees in sectional order (ie., those from section 1 first). So the use of the test section in development of the mapping rules has been at most indirect.

3.7 Comparison to Nitrogen

While Nitrogen recognizes a few syntactic relations, its focus is on semantic and other more abstract relations. HALogen, on the other hand, adds a full set of deep and shallow syntactic features intended to achieve extensive coverage of English syntax. Semantic rules from Nitrogen were modified in HALogen to map to syntactic ones rather than directly specifying an order on constituents. The syntactic features not only facilitate methodical, thorough coverage of English syntax, they also enable an application to precisely control the output, if desired.

In HALogen, most rules have only one right-hand-side result, with the obvious exception of the filling rules. Filling rules have multiple right-hand-sides in order to create a new input for each potential value of an unspecified feature. To the extent that other rules might have more than one right-hand-side, a client application has no way of controlling in advance the output. This was the usual case in Nitrogen, but HALogen's objective of allowing client control when desired has resulted in a different organization of the rules. Instead, features are defined in HALogen to control the generation of most alternatives.

HALogen handles adjunct relations, both semantic and syntactic, much better than Nitrogen. In HALogen, semantic adjuncts such as :spatial-locating and :temporal-locating can occur multiple times at a given level of nesting in an input, rather than being artificially restricted to just one each per node. HALogen also offers both the capability to try all possible order permutations of adjuncts, as well as the ability to impose partial order constraints on them. Applications concerned with rhetorical structure or coherence across sentences in multi-sentence generation need to have this kind of control. In contrast, Nitrogen arbitrarily assigns a single fixed order to adjuncts according to the order of the respective semantic relations in the set of mapping rules.

Nitrogen uses categorial grammar notations in the mapping rules to constrain generation output. However, HALogen abandons this because it has proved overly restrictive in scaling up to broad coverage. HALogen relies more heavily on the statistical ranker to implement grammatical preferences. When constraints are imposed, it is done by adding features to an input using the recasting mechanism. HALogen uses a feature named :type to impose three gross category constraints on phrases: verbal, nominal, or other. The symbolic processing engine checks the consistency and soundness of each input after recasting.

Other features of the engine include:

- In input: template-like capability using :template and :filler roles with labels
- Efficiency: improved cache and rule matching procedure

- Weights possible in grammar rules, input, and for concept-to-word mappings
- Polished output

Chapter 4

Statistical Ranker

This chapter describes HALogen's statistical ranking module, which computes probabilistic scores for alternate expressions, and outputs the most likely ones. Due to the large number of possible expressions involved, a central issue is representing alternate expressions compactly. The previous chapter briefly introduced the forest representation that HALogen uses. This chapter describes it in greater detail, comparing it to another used by Nitrogen, HALogen's predecessor, called a *lattice*. It also describes a bottom-up dynamic programming algorithm for ranking trees in a forest, and presents empirical results demonstrating exponential improvements in time and space efficiency compared to a lattice. Much of this chapter was previously published in (Langkilde, 2000).

4.1 Representing Alternative Phrases

HALogen leverages corpus-based statistical knowledge in the generation process by itemizing possible alternatives when the knowledge needed to make a decision between them is not available, and later applying a probabilistic model to rank alternatives according to their likelihood.

Representing the alternatives as a simple list is an obvious but naive approach. In practice, the number of alternatives can exceed 10^{30}, making a simple list extremely impractical. Since many

decisions and sub-phrases are common across proposed sentences, a more compact representation that shares structure across alternate sentences is needed.

4.1.1 Lattices

A lattice is one such structure-sharing representation. It is commonly used for the speech recognition task. Nitrogen originally adopted it for generation.

A lattice is a graph where each arc is labeled with a word. One special special node represents the start of a sentence, and another represents the end. A complete path through the lattice from the start-of-sentence node to the end-of-sentence node represents a complete sentence. Multiple arcs leaving a particular node represent alternate paths. An example of a lattice is shown in Figure 4.1. This lattice encodes 576 unique sentences. This is far fewer than typical, but convenient for illustration.

The lattice in Figure 4.1 illustrates several types of decisions that need to be made in generation. For example, there is a choice between the root words "chicken" and "poulet", the choice of whether to use singular or plural forms of these words, the decision whether to use an article or not, and if so, which one—definite or indefinite. There are also other word choice decisions such as whether to use the auxiliary verb "could", "might", or "may", and whether to express the mode of eating with the predicate "have to", "be obliged to", or "be required to". Finally, there is a choice between active voice (bottom half of lattice), and passive voice (top half).

A lattice is able to represent large numbers of alternative phrases without requiring nearly the same amount of space that that a list of individual sentences would require. Inspection of the lattice reveals some duplication, however. For example, the word "chicken" occurs four times, while the sublattice for the noun phrase containing "chicken" is repeated twice. So is the verb phrase headed by the auxiliaries "could", "might", and "may".

Such repetition is common in a lattice representation for text generation, and has a negative impact on the efficiency of the ranking algorithm because the same set of score calculations end

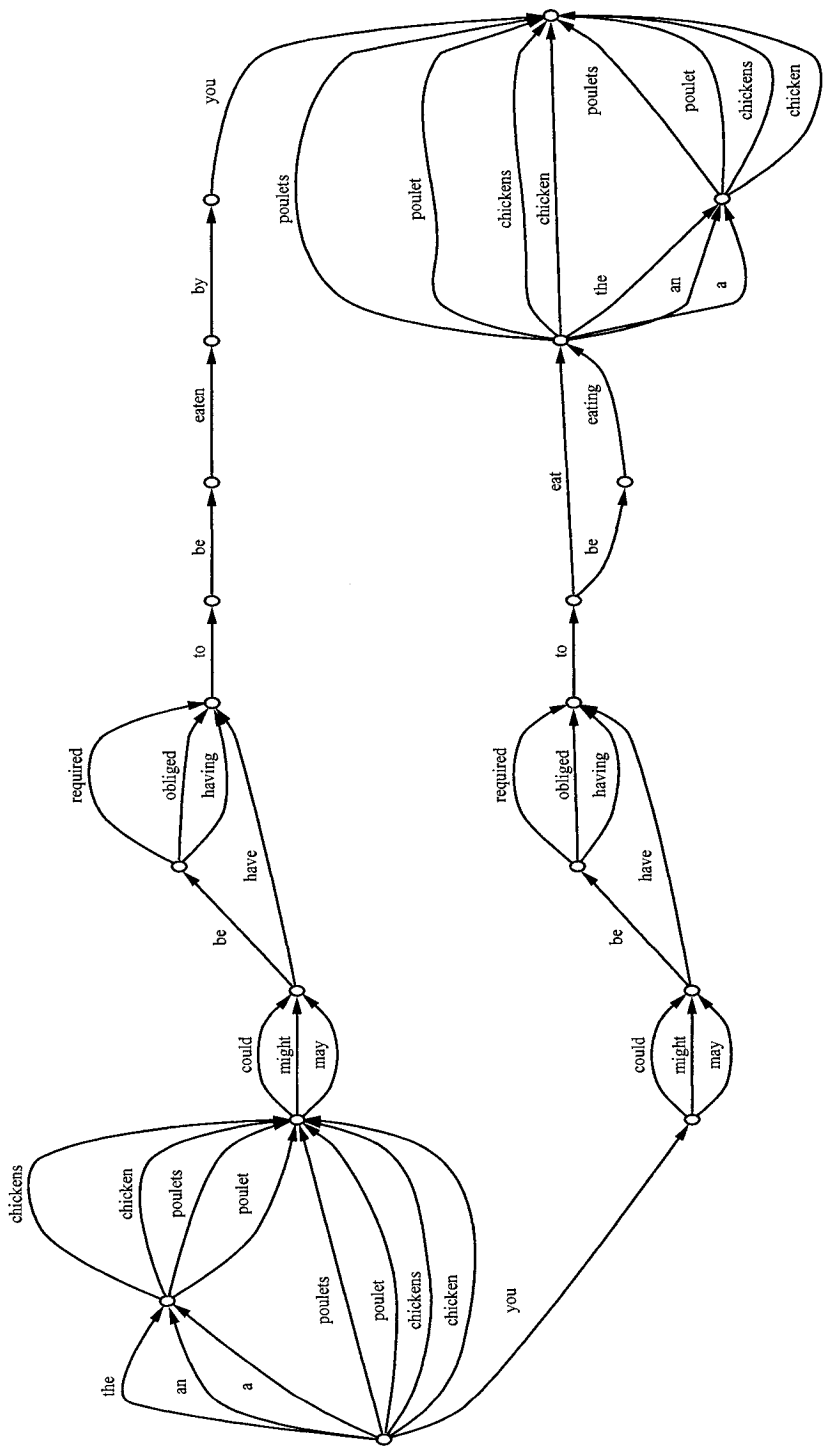


Figure 4.1: A lattice representing 576 different sentences, including "You may have to eat chicken", "The chicken may have to be eaten by you", etc.

up being made several times. Another drawback of the duplication is that the representation consumes much more storage space than necessary.

Yet another drawback of the lattice representation is that the independence between many choices cannot be fully exploited. Stolcke (1997) noted that 55% of all word dependencies occur between adjacent words. This means that most choices that must be made in non-adjacent parts of a sentence are independent. For example, in Figure 4.1, the choice between "may", "might", or "could" is independent of the choice between "a", "an" or "the" to precede "chicken" or "poulet".

Independence reduces the combination of possibilities that must be considered, and allows some decisions to be made without taking into account the rest of the context. Even adjacent words are sometimes independent of each other, such as the words "tail" and "ate" in the sentence "The dog with the short tail ate the bone". A lattice does not offer any way of representing which parts of a sentence are independent of each other, and thus cannot take advantage of this independence. This negatively impacts both the amount of processing needed and the quality of the results. In contrast, a forest representation, which we will discuss shortly, does allow the independence to be explicitly annotated and leveraged.

A final difficulty with using lattices is that the search space grows exponentially with the length of the sentence(s), making an exhaustive search for the most likely sentence impractical for long sentences. Heuristic-based searches offer only a poor approximation. Any pruning that is done renders the solution theoretically inadmissable, and in practice, frequently ends up pruning the mathematically optimal solution.

In Nitrogen, sentences longer than about 10 words suffered noticeably from the degradation that pruning during a 1000-best heuristic beam search caused. The heuristic search algorithm proceeded from left to right, keeping the 1000 best-scoring phrases at each step as it advanced.

4.1.2 Forests

These weaknesses of the lattice representation can be overcome with a forest representation. If we assign a label to each unique arc and to successively larger groups of non-overlapping arcs, a lattice becomes a forest. By aligning label assignments with groups of arcs that are repeated, the problems with duplication in a lattice are eliminated. The resulting structure can be represented as a set of context-free rewrite rules.

A forest representation corresponding to the lattice in Figure 4.1 is shown in Figure 4.2. As described in the previous chapter, a forest structure is an AND-OR graph, where the AND nodes represent sequences of phrases, and the OR nodes represent mutually exclusive alternate phrasings for sub-portion of an input. For example, at the top level of the forest, node S.469 encodes the choice between active and passive voice versions of the sentence. The active voice version is the left child node, labelled S.328, and the passive voice version is the right child node, S.358. There are eight OR-nodes in the forest, corresponding to the eight distinct decisions mentioned earlier that need to be made in deciding the best sentence to output.

The nodes are uniquely numbered, so that repeated references to the same node can be identified as such. In the forest diagram, only the first (left-most) reference to a node is drawn completely. Subsequent references only show the node name written in italics. This eases readability and clarifies which portions of the forest actually need to have scores computed during the ranking process. Nodes N.275, NP.318, VP.225 and PRP.3 are repeated in the forest of Figure 4.2.

Figure 4.3 illustrates how the forest is represented internally, using the PF style notation described in the previous chapter. The figure shows context-free rewrite rules for the top three levels of nodes in the forest. OR-nodes are indicated by the same label occuring more than once on the left-hand side of a rule. This sample of rules includes two examples of multiple references to a node, specifically to nodes NP.318 and PRP.3. They both occur on the right-hand side of more than one rule, and are instances of structure-sharing.

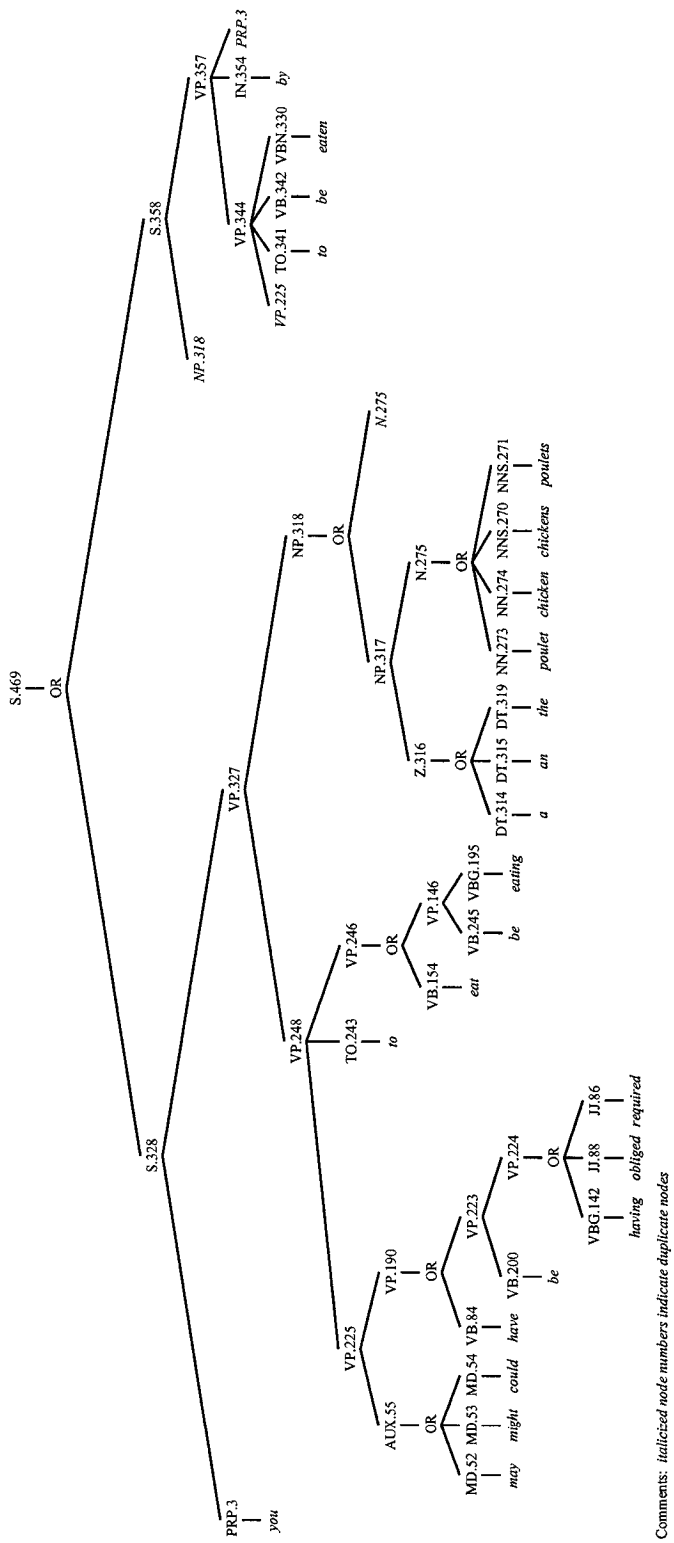


Figure 4.2: A generation forest

S.469	$\Longrightarrow$ S.328
S.469	$\Longrightarrow$ S.358
S.328	$\Longrightarrow$ PRP.3 VP.327
PRP.3	$\Longrightarrow$ "you"
VP.327	$\Longrightarrow$ VP.248 NP.318
S.358	$\Longrightarrow$ NP.318 VP.357
NP.318	$\Longrightarrow$ NP.317
NP.318	$\Longrightarrow$ N.275
VP.357	$\Longrightarrow$ VP.344 IN.354 PRP.3

Figure 4.3: Internal PF representation of top three levels of nodes in the forest shown in this chapter

It takes 42 rewrite rules to represent this forest. This compares favorably to the 48 arcs needed to represent the corresponding lattice. In general, however, a forest has slightly more overhead in representation on a per-unique-word basis than a lattice because it also represents hierarchical structure. However, longer sentences and larger sets of expressions result in exponential duplication of structure representation in a lattice. It is for these larger cases that a forest provides a real benefit, as shown at the end of this section.

A forest need not necessarily be derived from a lattice, but can be constructed directly during the generation process. Its definition also does not require it to represent sentence structure in a way that complies with linguistic theory, though it may. In fact, with a forest representation, it is quite natural to incorporate syntactic information. Syntactic information offers potentially significant advantages for statistical language modeling.

A generation forest differs from a parse forest in that a parse forest represents different possible hierarchical structures that cover a single phrase. In contrast, a generation forest generally represents one (or only a few) hierarchical structures for a given phrase, but represents many different phrases the express the same meaning.

4.1.3 Previous work on packed generation trees

There has been previous work on developing a representation for a packed generation forest structure. Shemtov (1996) describes extensions to a chart structure for generation originally presented in (Kay, 1996) that is used to generate multiple paraphrases from a semantic input. A prominent aspect of the representation is the use of boolean vector expressions to associate each sub-forest with the portions of the input that it covers and to control the unification-based generation process. A primary goal of the representation is to guarantee that each part of the semantic input is expressed once and only once in each possible output phrase.

In contrast, HALogen uses a separate data structure to represent associations between pieces of the input and nodes in the forest. The mappings between the two are a by-product of the caching done by the symbolic generator, as described in the previous chapter. Once-and-only-once coverage of the semantic input in a sentence is an implicit result of the symbolic generation algorithm. HALogen's forest representation is slightly less compact in comparison, but both the representation and the generation/ranking algorithms are much simpler.

4.2 Forest ranking algorithm

The algorithm described here for ranking sentences in a forest is a bottom-up dynamic programming algorithm. It is analogous to a probabilistic chart parser, but performs an inverse kind of comparison. Rather than comparing alternate syntactic structures indexed to the same positions of an input sentence, it compares alternate phrases corresponding to the same pieces of semantic input.

As in a probabilistic chart parser, the key insight of this algorithm is that the score for each of the phrases represented by a particular node in the forest can be decomposed into a context-independent (internal) score, and a context-dependent (external) score. The internal score, once

computed, is stored with the phrase, while the external score is computed in combination with other sibling nodes.

In general, the internal score for a phrase associated with a node p can be defined recursively as:

$$I(p) = \prod_{j=1}^{J} I(c_j) * E(c_j|context(c_1..c_{j-1}))$$

where I stands for the internal score, E the external score, and c_j for a child node of p. The specific formulation of I and E, and the precise definition of the *context* depends on the language model being used. As an example, in a bigram model,[1] I=1 for leaf nodes, and E can be expressed as:

$$E = P(FirstWord(c_j)|LastWord(c_{j-1}))$$

Depending on the language model being used, a phrase will have a set of externally-relevant *features*. These features are the aspects of the phrase that contribute to the context-dependent scores of sibling phrases, according to the definition of the language model. In the case of the bigram model, the features are the first and last words of the phrase. In a trigram model it is the first and last *two* words. In more elaborate language models, features might include elements such as head word, part-of-speech tag, constitutent category, etc. Naturally, the degree to which the language model matches reality, in terms of what features are considered externally relevant, will affect the quality of the output.

A crucial advantage of the forest-based method is that at each node only the best internally scoring phrase for each unique combination of externally relevant features needs to be maintained. The rest can be pruned without sacrificing the guarantee of obtaining the overall optimal solution. This pruning reduces exponentially the total number of phrases that need to be considered. In

[1] A bigram model is based on conditional probabilities, where the likelihood of each word in a phrase is assumed to depend on only the immediately previous word. The likelihood of a whole phrase is the product of the conditional probabilities of each of the words in the phrase.

VP.344 $\Longrightarrow$	VP.225	TO.341	VB.342	VBN.330
might have to be eaten	might have	to	be	eaten
may have to be eaten	may have			
could have to be eaten	could have			
	might be required			
	may be required			
	could be required			
	might be having			
	may be having			
	could be having			
	might be obliged			
	may be obliged			
	could be obliged			

Figure 4.4: Pruning phrases from a forest node, assuming a bigram model

effect, the ranking algorithm exploits the independence that exists between most disjunctions in the forest.

To illustrate this, Figure 4.4 shows an example of how phrases in a node are pruned, assuming a bigram model. The rule for node VP.344 in the forest of Figure 4.2 is shown, together with the set of phrases corresponding to each of the nodes. If every possible combination of phrases is considered for the sequence of nodes on the right-hand side, there are three unique first words, namely "might", "may" and "could", and only one unique final word, "eaten". Given that only the first and last words of a phrase are externally relevant features in a bigram model, only the three best scoring phrases (out of the 12 total) need to be maintained for node VP.344—one for each unique first-word and last-word pair. The other nine phrases can never be ranked higher by definition of the bigram model, no matter what constituents VP.344 later combines with.

In this particular example, the internal words of VP.344 turn out to be identical. However, this is not necessarily the case in general, because the most likely internal phrase depends on the context words, and may vary accordingly.

Pseudocode for the ranking algorithm is shown below. "Node" is assumed to be a record composed at least of an array of child nodes, "Node->c[1..N]," and best-ranked phrases, "Node->p[1..M]." The function ConcatAndScore concatenates two strings together, and computes a new

score for it based on the formula given above. The function Prune guarantees that only the best phrase for each unique set of features values is maintained. The core loop in the algorithm considers the children of the node one-by-one, concatenating and scoring the phrases of the first two children and pruning the results, before considering the phrases of the third child, and concatenating them with the intermediate results from the first two nodes, and so on. From the pseudocode,

```
RankForest( Node)
{
  if ( Leafp( Node)) LeafScore( Node);
  for j=1 to J {
      if ( not( ranked?(Node->c[j])))
          RankForest(Node->c[j]);
  }
  for m=1 to NumberOfPhrasesIn( Node->c[1])
      Node->p[m] = (Node->c[1])->p[m];
  k=0;
  for j=2 to J {
      for m=1 to NumberOfPhrasesIn( Node)
          for n=1 to NumberOfPhrasesIn(
                      Node->c[j])
              temp[k++] = ConcatAndScore(
                      Node->p[m],
                      (Node->c[j])->p[n]);
      Prune( temp);
      for m=1 to NumberOfPhrasesIn( temp)
          Node->p[m] = (temp[m]);
  }
}
```

it can be seen that the complexity of the algorithm is dominated by the number of phrases associated with a node (not the number of rules used to represent the forest, nor the number of nodes on the right-hand side of a node rule). More specifically, because of the pruning, it depends on the number of features associated with the language model, and the average number of unique combinations of feature values that are seen. If f is the number of features, v the average number of unique values seen in a node for each feature, and N the number of N best being maintained for each unique set of feature values (but not a cap on the number of phrases), then the algorithm

has the complexity $O((vN)^{2f})$ (assuming that children of AND nodes are concatenated in pairs). Note that f=2 for the bigram model, and f=4 for the trigram model.

In comparison, the complexity of an exhaustive search algorithm on a lattice is $O((vN)^l)$, where l is approximately the length of the longest sentence in the lattice. The forest-based algorithm thus offers an exponential reduction in complexity while still guaranteeing an optimal solution. A capped N-best heuristic search algorithm on the other hand has complexity O(vNl). However, as mentioned earlier, it typically fails to find the optimal solution with longer sentences.

The tables in Figure 4.5 and Figure 4.6 show experimental results comparing a forest representation to a lattice in terms of the time and space used to rank sentences. These results were generated from 15 test set inputs, whose average sentence length ranged from 14 to 36 words. They were ranked using a bigram model. The experiments were run on a Sparc Ultra 2 machine. Note that the time results for the lattice are not quite directly comparable to those for a forest because they include overhead costs for loading portions of a hash table. It was not possible to obtain timing measurements for the search algorithm alone. We estimate that roughly 80% of the time used in processing the lattice was used for search alone.

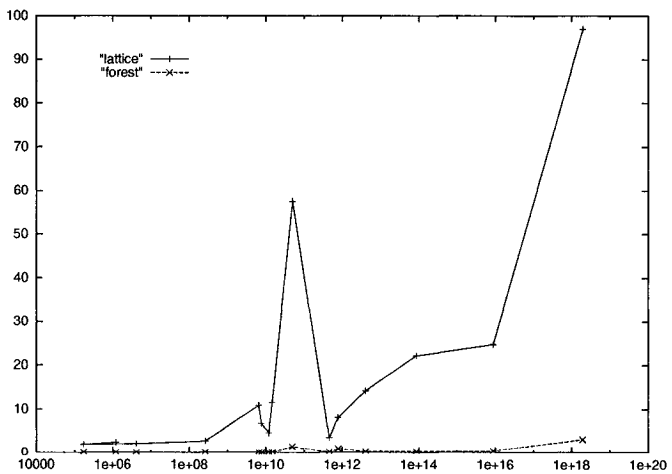


Figure 4.5: Time required for the ranking process using a lattice versus a forest representation. The X-axis is the number of paths (log_{10} scale), and the Y-axis is the time in seconds.

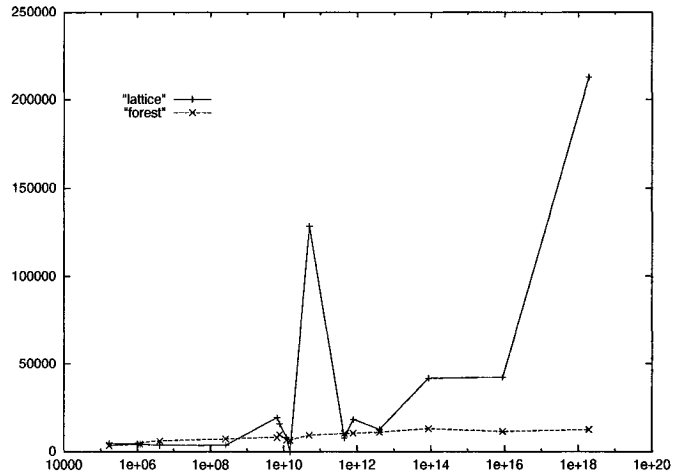


Figure 4.6: Size of the data structure for a lattice versus a forest representation. The X-axis is the number of paths (log_{10} scale), and the Y-axis is the size in bytes.

In that respect, it can be observed from Table 4.5 that the forest ranking program performs at least 3 or 4 seconds faster, and that the time needed does *not* grow linearly with the number of paths being considered as it does with the lattice program. Instead it remains fairly constant. This is consistent with the theoretical result that the forest-based algorithm does not depend on sentence length, but only on the number of different alternatives being considered at each position in the sentence.

From Table 4.6 it can be observed that when there are a relatively moderate number of sentences being ranked, the forest and the lattice are fairly comparable in their space consumption. The forest has a little extra overhead in representing hierarchical structure. However, the space requirements of a forest do not grow linearly with the number of paths, as do those of the lattice. Thus, with very large numbers of paths, the forest offers significant savings in space.

The spike in the graphs deserves particular comment. Our current system for producing forests from semantic inputs generally produces OR-nodes with about two branches. The particular input that triggered the spike produced a forest where some high-level OR-nodes had a much larger number of branches. In a lattice, any increase in the number of branches exponentially increases

the processing time and storage space requirements. However, in the forest representation, the increase is only polynomial with the number of branches, and thus did not produce a spike.

The most significant aspect of these experiments is that calculating the mathematically optimal sentence becomes tractable using a forest representation and dynamic programming algorithm—particularly for longer sentences.

4.3 Implementation Specifics

HALogen's forest ranker can extract the N most likely phrases from a forest, not just the single best. It uses an ngram language model built using Version 2 of the CMU Statistical Language Modeling Toolkit (Clarkson and Rosenfeld, 1997). Unigram, bigram and trigram models are all available. They are trained on 250 million words of Wall Street Journal newspaper text, excluding text from 1989 (from which the Penn Treebank is derived).

Chapter 5

The Practical Value of Ngrams in Generation

This chapter examines the synergy between symbolic and statistical language processing and discusses the performance in practice of an ngram language model. The analysis provides insight into the kinds of linguistic decisions that ngram frequency statistics can make, and how they improve scalability. This chapter also discusses the limits of bigram statistical knowledge. It is organized around specific examples of inputs to the generator system and its outputs.

The analysis was originally performed using the Nitrogen system; thus, the examples are illustrated with lattices. However, the conclusions apply to any system using ngrams, including HALogen. Much of this chapter was previously published as (Langkilde and Knight, 1998b).

5.1 Example 1

The first example I will use is the following input. It represents the idea that a selling/betraying action performed by myself on the trust/reliance belonging to "they" is not workable.

Symbolic generation for this input produces a word lattice containing 270 nodes, 592 arcs, and 155,764 distinct paths. Space permits only two portions (about one-fifth) of the sentence lattice to be shown in Figure 5.1. An example of a random path through the lattice not shown is "Betrayal of an trust of them by me is not having a feasibility." The top 10 paths selected by the statistical extractor are:

```
(A / |workable|
  :DOMAIN (A2 / |sell<cozen|
             :AGENT I
             :PATIENT (T / |trust,reliance|
                         :GPI THEY))
  :POLARITY NEGATIVE)
```

```
I cannot betray their trust .
I will not be able to betray their trust .
I am not able to betray their trust .
I are not able to betray their trust .
I is not able to betray their trust .
I cannot betray the trust of them .
I cannot betray trust of them .
I cannot betray a trust of them .
I cannot betray trusts of them .
I will not be able to betray the trust of them .
```

The statistical extractor prefers common words and word combinations. When a subject and a verb are contiguous, it automatically prefers the verb conjugation that agrees with the subject. When a determiner and its head noun are contiguous, it automatically prefers the most grammatical combination (not "a trusts" or "an trust"). For example, here are the corpus counts for some of the subject-verb bigrams and determiner-noun bigrams:

```
I am 2797              a trust 394    an trust  0    the trust  1355
I are  47              a trusts  2    an trusts 0    the trusts  115
I is   14
```

Note that the secondary preference for "I are" over "I is" makes sense for sentences like "John and I are..." Also note the apparent preference for the singular form of "trust" over the plural form, a subtle reflection of the most common meaning of the word "trust". This same preference is reflected for the bigrams "their trust" versus "their trusts."

```
their trust 28     their trusts 8
```

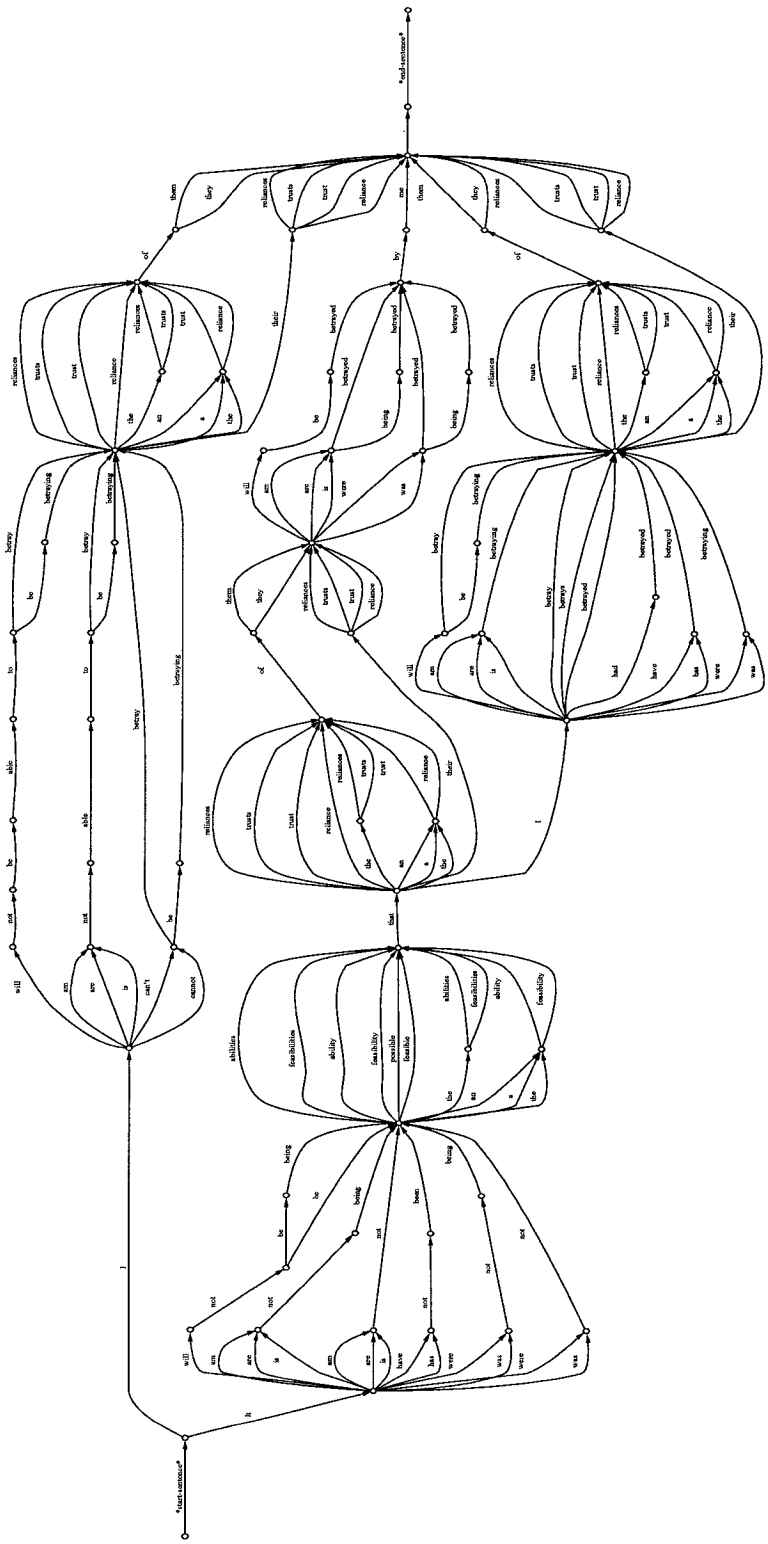


Figure 5.1: Lattice for Example 1

A purely symbolic generator would need a tremendous amount of deep, handcrafted knowledge to infer that the "reliance" meaning of trust must be singular. Our generator handles it automatically, based simply on the evidence of colloquial frequency.

(The plural form is used in talking about monetary trusts, or as a verb. To be more accurate, the generator would need access to ngram frequencies based on word/sense combinations, rather than just words. However, it is worth pointing out that raw ngrams are biased in favor of the most common usage of a word, which in practice compensates to a large extent for the ambiguity. To get the less frequent form instead, the input would need to be more explicitly specified.)

A heuristic for preferring shorter phrases accounts for the preference of sentence 1 over sentence 2, and also for the preference of "their trust(s)" over "the trust(s) of them". As can be seen from the lattice in Figure 2, the generator must also make a word choice decision in expressing the concept `|trust,reliance|`. It has two root choices, "trust" and "reliance."

```
reliance 567    reliances  0    trust  6100      trusts 1083
```

The generator's preference is predicted by these unigram counts. Trust(s) is much more common than reliance(s). Though one could modify the symbolic lexicon to suit their needs more precisely, doing so on a scale of 100,000 concepts is prohibitive. Thus the corpus statistics offer better scalability.

5.2 Example 2

Consider another input:

```
(A / |admire<look|
  :AGENT (V / |visitor|
         :AGENT-OF (C / |arrive,get|
                   :DESTINATION (J / |Nihon|)))
  :PATIENT (M / "Mount Fuji"))
```

Lattice statistics: 245 nodes, 659 arcs, 11,664,000 paths.

Random path: `Visitant which came into the place where it will be`

`Japanese has admired that there was Mount Fuji.`

Top paths extracted:

```
Visitors who came in Japan admire Mount Fuji .
Visitors who came in Japan admires Mount Fuji .
Visitors who arrived in Japan admire Mount Fuji .
Visitors who arrived in Japan admires Mount Fuji .
Visitors who came to Japan admire Mount Fuji .
A visitor who came in Japan admire Mount Fuji .
The visitor who came in Japan admire Mount Fuji .
Visitors who came to Japan admires Mount Fuji .
A visitor who came in Japan admires Mount Fuji .
The visitor who came in Japan admires Mount Fuji .
Mount Fuji is admired by a visitor who came in Japan .
```

This example offers more word choice decisions to the generator than the previous example. There are two root choices for the concept `|visitor|`, "visitor" and "visitant," and two for `|arrive,get|`. Between "visitor" and "visitant", it is easy to guess that the generator will prefer "visitor." However, the choice between singular and plural forms for "visitor" results in a decision opposite to the one made for "trust" above.

```
visitor 575     visitors 1083
```

In this case the generator prefers the plural form.

For relative pronouns, the extractor is given a choice between "that," "which," and "who," and nicely picks "who" in this case, though it has no symbolic knowledge of the grammatical constraint that "who" is only used to refer to people.

```
visitor_who   9     visitors_who  20
visitor_which 0     visitors_which 0
visitor_that  9     visitors_that 14
```

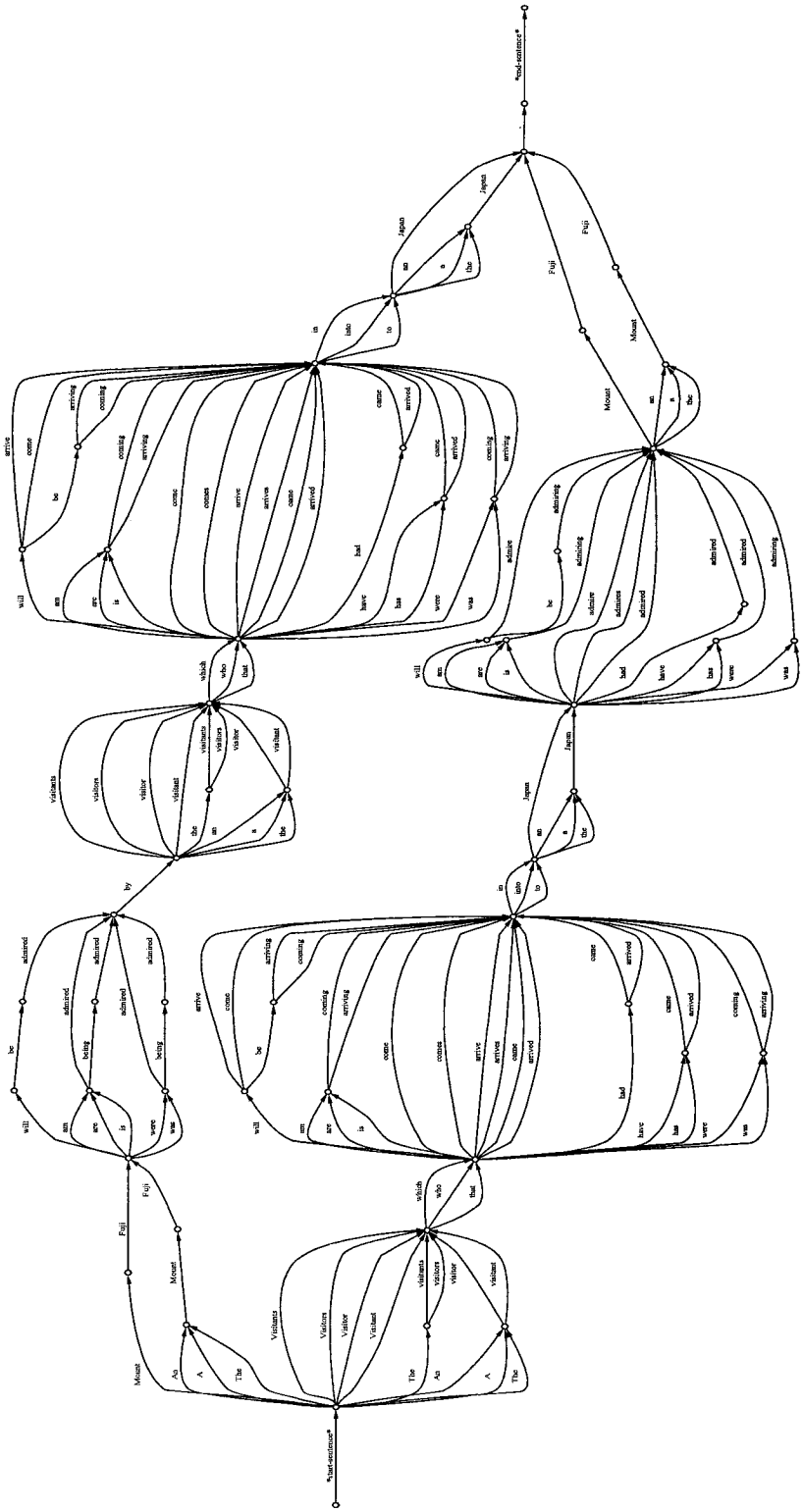


Figure 5.2: Lattice for Example 2

As can be seen from the lattice in Figure 5.2, the generator is given a wide variety of choices for verb tense. The heuristic preferring shorter phrases in effect narrows the choice down to the one-word expressions. Between these, the past tense is chosen, which is typical. In choosing between forms of "come" and "arrive," the generator unsurprisingly prefers the more common word "come" and its inflected forms.

```
who_came 383      who_arrived 86

who_come 142      who_arrive  15

who_comes 89      who_arrives 11
```

A notable error of the generator in this example is in choosing the preposition "in" to precede "Japan."

```
in_Japan 5413      to_Japan 1196
```

Japan is often preceded by the preposition "in." It is the context of the verb "come" that makes one wish the generator would choose "to", because "arrived in Japan" sounds fine, while "came in Japan" does not.

```
came_to  2443     arrived_in 544

came_in  1498     arrived_to 35

came_into 244     arrived_into 0
```

However, "in Japan" is so much stronger than "to Japan" when compared with "came to" versus "came in" that "in" wins. The problem here is that bigrams cannot capture dependencies that exist between more than two words. A look at trigrams shows that this dependency does indeed exist.

```
came_to_Japan   7      arrived_to_Japan   0

came_into_Japan 1      arrived_into_Japan 0

came_in_Japan   0      arrived_in_Japan   4
```

A final aspect of this example worth discussing is the choice of person for the root verb "admire." In this case, all the relevant bigrams are zero (ie. between "Japan" and "admire," and between "admire" and "Mount," so the decision essentially defaults to the unigrams.

```
admire 212      admired 211      admires 107
```

In this example the generator got lucky in its choice, because if there had been non-zero bigrams, they would have most likely caused the generator to make the wrong choice, by choosing a third person singular conjugation to agree with the contiguous word "Japan" rather than a third person plural conjugation to agree with the true subject "visitors".

Note that another weakness of the current extractor in being able to choose the conjugation of any verb is that it weighs the bigrams between the verb and the word previous to it the same as it does the bigrams between the verb and the word that follows, although the former dependency is more important in choosing grammatical output. In other words, for a phrase like "visitors admire(s) Mount", the bigrams "admire(s) Mount" are weighed equally with the bigrams "visitors admire(s)", though obviously "Mount" should be less relevant in choosing between "admire" and "admires" than "visitors".

5.3 Discussion

The strength of a statistical approach to generation is its simplicity and robustness. Basic unigram and bigram frequencies capture much of the linguistic information relevant to generation. Yet there are also inherent limitations. One is that dependencies between non-contiguous words cannot be captured, nor can dependencies between more than two items (in a bigram-based system).

As mentioned above, trigrams are one way of capturing part of these dependencies. However, they only capture the contiguous ones, and moreover, practical experience suggests that switching to a trigram-based system can pose as many problems as it solves. In particular, trigrams exponentially increase the amount of data that must be stored during ranking using a forest-based

approach, reducing the tractability of the approach. Overall, however, the previous discussion demonstrates that simple corpus-based frequencies can solve many of the decision problems that have long bedeviled the generation problem as a whole.

Chapter 6

An Empirical Evaluation of Coverage and Correctness

This chapter describes an empirical evaluation of the syntactic coverage and correctness of the HALogen sentence realizer using a section of the Penn Treebank corpus as a test set. HALogen's ability to handle varying kinds and degrees of underspecification is also evaluated. This evaluation is the first large-scale measurement of coverage reported in the literature. It is also the the first evaluation of a system's ability to handle underspecification, and the first measurement of a system's ability to correctly generate an almost fully-specified input. Much of this chapter was previously published as (Langkilde-Geary, 2002).

6.1 Introduction

Prominent general-purpose realization systems developed to date include FUF/Surge (Elhadad, 1993) and (Elhadad and Robin, 1998), RealPro (Lavoie and Rambow, 1997), Penman/KPML (Bateman, 1996), and Nitrogen (Langkilde and Knight, 1998a), (Knight and Hatzivassiloglou, 1995b). These systems have demonstrated their general usefulness by being deployed in a variety of different applications. However, it is still difficult to ascertain the degree to which they have achieved broad coverage of natural language or high quality output because no empirical evaluation has been performed.

At best, suites of example inputs have been used for regression testing. However, such regression suites are biased towards the capabilities of their respective systems and consist of relatively few inputs compared to the variety of input classes that are possible. For example, there are currently 500 test inputs distributed with Surge, and about 210 for English with KPML. At any rate, no matter how large the regression suite may be, the inherent irregularity of natural language makes regression testing inadequate as a means of assessing coverage or quality.

In practice, there is a seemingly irreconcilable conflict between broad coverage and high quality output. It is usually the case that the rules and class features are simultaneously too general to rule out undesirable combinations, and yet too restrictive to allow some combinations that are valid. High quality output is thus much easier to achieve with smaller-scale applications or in limited domains.

For example, in the text structure representation proposed by Meteer (1990), the labeling of the verb "land" as a completive event and the labeling of the phrase "for ten minutes" as a duration was expected to prevent the awkward clause, "the plane landed for ten minutes" (and ones similar to it) from being generated. However, a search on the web immediately turns up legitimate sentences that contain such constituent combinations, such as "landed for ten minutes and then took off again". A rule against such a combination would thus be overly restrictive for actual large-scale English generation.

Similarly, as another example, the documentation for the Surge grammar indicates that locative adjuncts of sentences are only allowed to be adverbs or prepositional phrases. However, the Penn Treebank corpus contains at least eight sentences that violate this rule.

HALogen's development has been guided in part by the question, "What is the simplest input notation that suffices to allow the generation of all valid sentences, that at the same time minimizes the generation of invalid English, and is yet easy for applications to use?" This question is likely similar to that which has motivated the design of other generators. However, the empirical

evaluation described in this chapter is the first large-scale empirical measurement ever performed to estimate the degree of success achieved.

The next section describes the setup of the empirical evaluation, and the following one discusses the results. Finally, the last section concludes.

6.2 Experimental Setup

The goal of this empirical evaluation of coverage and quality is to measure the extent to which any and every valid English sentence can be represented and generated. Generation quality is usually notoriously difficult to evaluate because grammaticality is difficult to measure automatically, and more than one output can be acceptable. However, although variations in output are usually acceptable in the context of specific applications, different applications can have different constraints on the kinds of variation in output that they accept. By demonstrating the capability to produce any desired sentence exactly, a system can assure that all possible application constraints on the output can be met. Thus, these experiments focus on whether a desired sentence can be produced exactly, though this kind of measurement is harsher than necessary.

An input construction tool, described shortly, was developed and tested using Sections 1-22 of the Penn Treebank. Then HALogen inputs were automatically constructed from Section 23, consisting of about 2400 sentences, to be used as a blind test set for evaluating coverage and quality. The evaluation consists of generating the test set inputs using HALogen, and comparing the output to the original sentence.

The Penn Treebank offers several advantages as a test set. It contains real-world sentences, it is large, and can be assumed to exhibit a very broad array of syntactic phenomena. It is not biased towards system-specific capabilities, since it was collected independently. It also acts as a standard for linguistic representation, offering the potential of interoperability with other natural language programs based on it, such as parsers. At the same time, there are limits to its usefulness.

It only represents the domain of newspaper text, and thus does not test the stylistic, structural, and content variations that can occur in other domains such as question answering or dialogue. It also does not evaluate how the system handles nonsensical inputs or inputs that might not be expressible in grammatical English.

The generator's primary tasks in this evaluation are to determine the linear order of constituents, perform morphological inflections, and insert needed function words. Six experiments, described below, are run to evaluate HALogen's performance on inputs that are underspecified in different ways. The ability to handle underspecification eases the information burden on client applications. It also makes the generator more flexible in meeting the varying needs and constraints of different types of applications.

6.2.1 Automatic Input Construction

Sentences in the Penn Treebank are annotated using phrase structure categorial grammar, as shown in Figure 6.1. In contrast, inputs to HALogen use a dependency-style notation. (A dependency-style notation is more suitable as input to generation since the one of the main tasks to be performed is determining linear constituent order given the relationship between a pair of entities.) Thus the main tasks in constructing inputs automatically from the Treebank annotation are determining the head constituent and labeling the relationship between the head and each other child.

The mapping from Treebank annotation to HALogen inputs is complicated by the flatness of Treebank's annotation of some types of phrases. In particular, the flatness of base noun phrases, of coordinated single-word phrases, of compound prepositions, and of most punctuation leads to complexity in the conversion process, and sometimes ambiguity errors. For example, in Figure 6.1, the flat annotation of the phrase "Boeing Co. 707s" leads to a dependency representation in which "Boeing" is improperly considered a dependent of "707s", rather than a modifier of "Co."

```
(TOP (S (ADVP-TMP (RBR Earlier))
        (NP-SBJ (DT the)
                (NN company))
        (VP (VBD announced)
            (SBAR (-NONE- 0)
                  (S (NP-SBJ (PRP it))
                     (VP (MD would)
                         (VP (VB sell)
                             (NP (NP (PRP$ its)
                                     (VBG aging)
                                     (NN fleet))
                                 (PP (IN of)
                                     (NP (NNP Boeing)
                                         (NNP Co.)
                                         (NNPS 707s))))
                             (PP-PRP (IN because)
                                     (IN of)
                                     (NP (VBG increasing)
                                         (NN maintenance)
                                         (NNS costs))))))))
        (. .)))
```

Figure 6.1: Penn Treebank annotation for the sentence, "Earlier the company announced it would sell its aging fleet of Boeing Co. 707s because of increasing maintenance costs."

Similarly, "because" and "of" need to be grouped together because they are viewed as functioning as a unit rather than independently to describe the relationship between "sell" and "increasing maintenance costs". In the cases of compound prepositions and coordinated phrases at least, the extra nesting can be added automatically with a fair degree of reliabilty, and the conversion tool does so. The flatness of NP's is more challenging however, and for now is left as is.

On the other hand, the degree of nesting used to represent verb phrases causes difficulties of the opposite kind, since HALogen usually uses a flat representation for predicates and their dependents. For example, in the clause fragment "it would sell its aging fleet", HALogen would represent "sell" as the head, with "would", "it", and "fleet" as dependents. A sample simplified flat representation of this is:

```
( / sell
  :subject it
  :modal would
  :object fleet)
```

The extra nesting used in Treebank seems to serve mainly to facilitate the representation of coordinated verb phrases. From a semantic representation and language modeling perspective, however, it would be more useful to represent such coordinated phrases with a flatter notation that included referential null elements in place of ellided dependents. Since inferring such null elements automatically is complex and error-prone, and since HALogen allows a hierarchical dependency representation, the input construction tool compromises by flattening predicates when possible, and leaving a minimal amount of nesting in place in the case of coordination.

The functional roles of dependency relations are inferred mainly from the constituent labels in Treebank. Table 6.1 indicates some of the correspondences between Treebank annotation and the functional roles used by HALogen. The actual mapping is somewhat more complicated, due to the irregularity of natural language and idiosyncracies in the Treebank annotation.

Treebank annotation	HALogen Syntactic Relations
-LGS	:logical-subject
-PRD	:predicate
-DTV	:logical-dative
-BNF	:beneficiary
-CLR	:closely-related
-ADV, -PRP, -LOC, -TMP, -MNR, -EXT, -DIR	:adjunct
-TPC	:topic
-SBJ	:subject
NP	:logical-dative *if followed by an NP and parent is VP* :logical-object *if and no function tags and parent is VP* :adjunct *otherwise*
"not" or "n't"	:polarity
IN	:anchor
TO	:anchor *or* :mood to-infinitive
DT	:anchor *if parent tag is SBAR* :determiner
PDT	:pre-determiner
CC or CONJP	:conj *or* :introconj *or* :premod *sometimes* :coordpunc
punctuation	:coordpunc *if used to separate coordinated constituents* :punc *otherwise*
otherwise	:adjunct

Table 6.1: Some correspondences between Penn Treebank annotation and HALogen syntactic relations

```
:LOGICAL-SUBJECT :INSTANCE   :ANCHOR
:LOGICAL-OBJECT  :POLARITY   :TOPIC
:LOGICAL-DATIVE  :ADJUNCT    :PREMOD
:CLOSELY-RELATED :WITHINMOD  :POSTMOD
:BENEFACTIVE     :INTROCONJ  :CONJ
:PREDICATE       :LEFTPUNC   :PUNC
:DETERMINER      :RIGHTPUNC  :PREDET
```

Figure 6.2: Relations used in experiments

In all, the input construction process involves finding the root forms of words, factoring Treebank categories of open class words into more basic features, heuristically designating constituent heads, inferring syntactic and logical roles for each node, making coordination bracketing more explicit, reorganizing compound prepositions into a single constituent, associating punctuation with a content-bearing constituent, flattening VP's, flattening nodes with only one child, removing null elements, and dropping some function words (ex: simple dative 'to', 'of'; benefactive 'for'; logical-subject 'by', auxiliary verbs, and some punctuation).

The experiments use only a subset of the relations that HALogen actually recognizes. Specifically, they use the mix of deep and shallow syntactic relations shown in Figure 6.2. No semantic relations are used, since they either cannot be straight-forwardly derived from the Penn treebank annotation, or are too ambiguous to be adequately handled on the scale of the experiments in this paper. A deep-syntactic level of abstraction is used because it is the deepest level that can be readily inferred from Treebank, and exercises more system capabilities than a shallower level would.

6.2.2 Underspecified-Input Experiments

In the first experiment, labeled "Almost fully spec" in Figure 6.2, the inputs contain nearly enough detail to fully determine a unique output. The inputs contain as much detail as it was possible to straight-forwardly obtain from the Treebank annotation. An example is shown in

```
ORIGINAL: Earlier the company announced it would sell its aging fleet of Boeing
Co. 707s because of increasing maintenance costs.

(H34911
  :MOOD INDICATIVE
  :PREMOD (H34876 :CAT RB :CAT1 COMPARATIVE :LEX "earlier")
  :LOGICAL-SUBJECT (H34879 :DET (H34877 :CAT DT :LEX "the")
                    / (H34878 :DET NONE :CAT NN :CAT1 COMMON :NUMBER SING
                        :LEX "company"))
  / (H34880 :CAT VV :TENSE PAST :LEX "announce")
  :POSTMOD  (H34908
             :VOICE ACTIVE
             :LOGICAL-SUBJECT (H34883 :DET NONE :CAT NN :CAT1 PRONOUN :LEX "it")
             :MODAL WOULD
             / (H34885 :CAT VV :LEX "sell")
             :LOGICAL-OBJECT (H34896
                              :DET NONE
                              / (H34889
                                  :DET NONE
                                  :PREMOD (H34886 :CAT JJ :CAT1 PRONOUN :LEX "its")
                                  :PREMOD (H34887 :CAT VV :LEX "age")
                                  / (H34888 :DET NONE :CAT NN :CAT1 COMMON :NUMBER SING
                                      :LEX "fleet"))
                              :POSTMOD (H34895
                                        :ANCHOR (H34890 :CAT IN :LEX "of")
                                        / (H34894
                                            :DET NONE
                                            :PREMOD (H34891 :DET NONE :CAT NN
                                                      :CAT1 PROPER :NUMBER SING
                                                      :LEX "Boeing")
                                            :PREMOD (H34892 :DET NONE :CAT NN
                                                      :CAT1 PROPER :NUMBER SING
                                                      :LEX "Co.")
                                            / (H34893 :DET NONE :CAT NN
                                                      :CAT1 PROPER :NUMBER PLURAL
                                                      :LEX "707"))))
             :POSTMOD (H34903
                       :ANCHOR (H34904
                                :PREMOD (H34897 :CAT IN :LEX "because")
                                / (H34898 :CAT IN :LEX "of"))
                       / (H34902
                           :DET NONE
                           :PREMOD (H34899 :CAT VV :MOOD PRESENT-PARTICIPLE
                                     :LEX "increase")
                           :PREMOD (H34900 :DET NONE :CAT NN :CAT1 COMMON
                                     :NUMBER SING :LEX "maintenance")
                           / (H34901 :DET NONE :CAT NN :CAT1 COMMON :NUMBER PLURAL
                               :LEX "cost"))))
  :PUNC PERIOD)
```

BIGRAM and TRIGRAM: (same as original) Earlier the company announced it would sell its aging fleet of Boeing Co. 707s because of increasing maintenance costs.

Figure 6.3: An almost fully specified input, and its output

```
ORIGINAL: Earlier the company announced it would sell its aging fleet of Boeing
Co. 707s because of increasing maintenance costs.

(H37 :ADJUNCT "earlier"
     :LOGICAL-SUBJECT (H5 / "company")
     / "announce"
     :ADJUNCT (H34 :LOGICAL-SUBJECT "it"
                   :MODAL WOULD
                   / "sell"
                   :LOGICAL-OBJECT (H22 / (H15 :ADJUNCT "its"
                                               :ADJUNCT "age"
                                               / "fleet")
                                        :ADJUNCT (H21 :ANCHOR "of"
                                                      / (H20 :ADJUNCT "Boeing"
                                                             :ADJUNCT "Co."
                                                             / "707")))
                   :ADJUNCT (H29 :ANCHOR (H30 :PREMOD "because"
                                              / "of")
                                 / (H28 :ADJUNCT "increase"
                                        :ADJUNCT "maintenance"
                                        / "cost")))
     :PUNC PERIOD)
```

BIGRAM: It would sell its fleet age of Boeing Co. 707s because of maintenance costs increase the company announced earlier.
TRIGRAM: The company earlier announced it would sell its fleet age of Boeing Co. 707s because of the increase maintenance costs.

Figure 6.4: A minimally specified input, and its output

Figure 6.3. In this experiment, adjuncts are represented either as premodifiers, postmodifiers or within-modifiers. (Within-modifiers are verbal modifiers that come between the subject and the object). A flag in the generator is set so that constituents with the same role occurring at the same level of nesting (such as modifiers) will be ordered in the output in the same relative order in which they appear in the input (and adjacent to each other). This order flag allows applications that plan discourse structure before doing sentence realization to control the coherence across sentences. For example, dialogue systems often want old or background information to appear before new information. (Partial order constraints can also be specified by using extra levels of nesting in the input. However, this capability is not exercised in these experiments.)

In the second experiment, "Permute same-roles," the permutation flag set in the first experiment is reversed. Constituents with the same role are permuted in place, and the statistical ranker is expected to choose the most likely order. An exception occurs if there happen to be more than five constituents with the same role. For computational reasons, the constituents are not permuted in this case. Instead, they are placed in reverse order in the output (to avoid unfairly inflating the accuracy results). Everything else remains the same as in the first experiment.

The third experiment, "Permute, no dir" is like the second, but in addition, all modifiers (:premod, :postmod, and :withinmod) are mapped to the :adjunct relation, thus increasing the number of constituents that get permuted. The statistical ranker must not only determine the order of the modifiers with respect to each other, but must determine the direction of each one with respect to the head.

The fourth experiment, "Underspec det", is like the first except that common determiners are left unspecified. Specifically, "the", "a", "an", "any", and "some" are dropped from the input. The null-determiner feature is also dropped from all nominal phrases that had no determiner in the original Treebank annotation. This experiment tests the ability of the generator to supply the appropriate determiner, or figure out that none is needed.

The fifth experiment, "No leaf, clause feats", is also like the first experiment except that all the leaf and clause properties listed in Figure 2.2 are dropped from the input. Only the value of the :lex feature is retained for the input.

The sixth experiment, "Min spec," represents the opposite extreme from the first experiment. All the information dropped in experiments 2-5 is also dropped in this experiment. An example of such an input and its output is shown in Figure 6.4.

For computational reasons, a bigram model, not trigram, was applied by the ranker for these experiments. However, for the sake of comparison, Figures 6.3 and 6.4 show the output from both the bigram and trigram models.

6.3 Results

Results of the experiments are shown in Figure 6.2. Section 23 of the Penn Treebank contains 2416 sentences. The average Penn Treebank sentence consisted of 23.5 tokens–the shortest had two, and the longest 66. The input construction tool produced inputs from 98% of the Penn sentences, or 2377 inputs. HALogen produced output for approximately 80% of the inputs. Assuming the test set is representative of English, and coverage is defined as the percent of syntactic constructions (ie., inputs) for which the generator produces correlative output, then we can estimate HALogen's coverage of English at about 80%.

We applied three different metrics to evaluate quality, the IBM Bleu score (Papineni et al., 2002), the average NIST simple string accuracy, and exact match. The IBM Bleu score is a geometric average of n-gram accuracy with respect to the original Penn Treebank sentence, adjusted by a length penalty factor LP. Namely, BLEU $= \exp(\sum_{n=1}^{N} w_n \log p_n) \times \mathrm{LP}$. LP $= \exp(1 - r/c)$ if $c \leq r$, and LP $= 1$ if $c > r$, where $w_n = 1/N$, $N = 4$, c is the system output length, and r is the reference length.

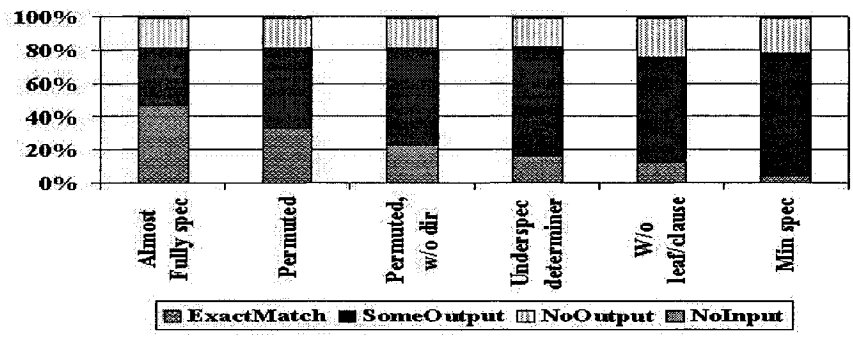


Figure 6.5: Coverage

The average NIST simple string accuracy score reflects the average number of insertion (I), deletion (D), and substitution (S) errors between the output sentence and the original Penn Treebank sentence. Formally, SSA $= 1 - (I + D + S)/R$, where R is the number of tokens in the original sentence.

The Bleu and NIST metrics agree fairly closely with each other in all the experiments. Using these metrics, HALogen's output ranged from about 93% correct when inputs were almost fully specified, to 53% correct with minimally specified inputs. Almost 58% of the outputs were exact matches in the first experiment, dropping to 5% in the sixth. Although the quality in the sixth experiment is substantially worse than that of the first, it requires much less information in the input, and thus is much easier for a client application to produce. For applications like machine translation that can tolerate imperfect output but have difficulty supplying detailed inputs, HALogen can be an appealing tool.

The second and third experiments show that permuting same-role nodes does not have as big an impact on quality as one might expect. This probably reflects the ngram model's strength in capturing order information, especially for single-word modifiers. It may also reflect the relative

Input characteristics:	Almost fully spec	Permute same-roles	Permute, no dir	Under-spec det	No leaf, clause feats	Min spec
Median num of sen gen by an input:	72	576	2e+5	7e+6	9e+9	4e+16
Smallest num of sen gen by an input:	1	1	1	2	2	4
Max num of sen gen by an input:	2e+10	1e+14	8e+19	9e+25	2e+32	2e+53
Average ranking time per input (secs):	0.013	0.023	0.226	0.036	0.207	18.3
Average total time per input (secs):	28.9	27.1	29.8	28.4	30.2	55.5
Average length of gen sentences:	22.4	22.4	22.4	21.9	21.7	21.0
Average length of exact matches:	20.9	18.8	17.0	16.4	16.0	11.5
Num of inputs that produced output:	1968	1968	1966	1981	1812	1884
Num of exact matches:	1132	807	555	391	299	98
Percent inputs that produced output:	82.8%	82.8%	82.7%	83.3%	76.2%	79.3%
Percent exact matches in output:	57.5%	41.0%	28.2%	19.7%	16.5%	5.2%
Ave. NIST simple string accuracy:	94.5%	81.6%	69.6%	85.1%	81.1%	55.3%
IBM Bleu score:	0.924	0.826	0.757	0.776	0.717	0.514

Table 6.2: Experimental Results

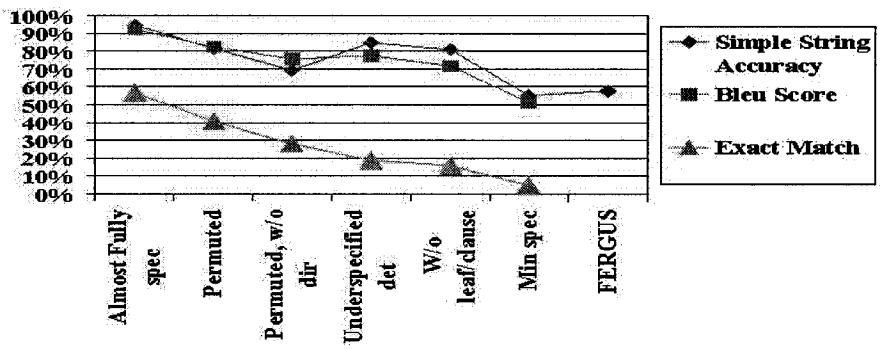


Figure 6.6: Accuracy

infrequence of nodes having multiple modifiers. In the fourth experiment, HALogen doesn't do as well at selecting determiners as one might expect. The divergence in this experiment between the exact match metric and the other metrics probably results from the need to choose determiners in nearly every sentence, while determiners constitute only a fraction of the words in a sentence. The fifth experiment shows the largest drop in accuracy compared to the second through fourth, suggesting that this problem is harder than the others. However, HALogen still achieves about 75% correct on this very frequent problem.

6.4 Causes of incorrect or failed output

One topic of interest is the causes of generation failure, and the causes of inexact matches in the first experiment. The most common cause is malformed inputs due to the difficulty of the automatic input construction process. Another cause is inconsistencies or errors in the original Treebank annotation, particularly with part-of-speech labels and bracketing conventions.

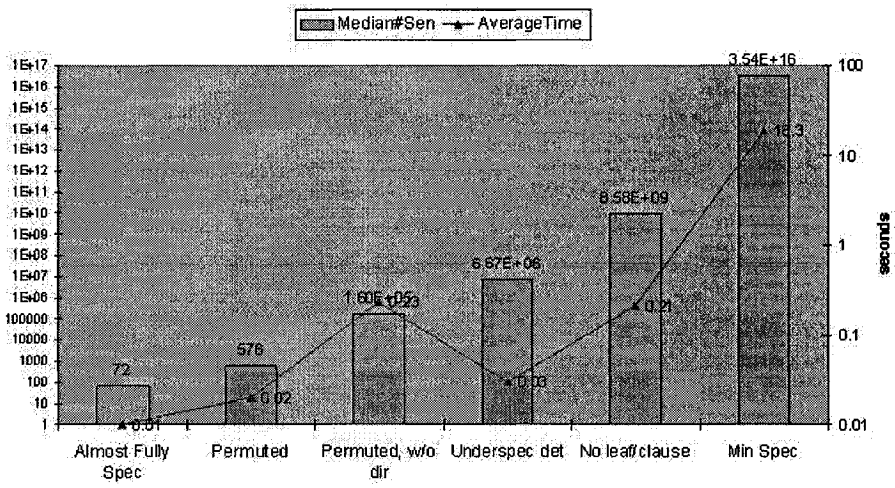


Figure 6.7: Average number of sentences and ranking time

The generator itself is also the source of some failures and errors. For example, HALogen may deliberately fail to generate an input because it violates grammaticality restrictions. Otherwise, information in the knowledge bases HALogen uses may be missing or incorrect. Additionally, there are some syntactic phenomena that are known to not be handled appropriately yet. This includes right-node-raising, dislocated constituents, discontinuous constituents, and headless constituents.

The main causes of failure and errors are highlighted in the following list, in chronological order of possible occurence. The accuracy errors that occur in the other experiments besides the first experiment, of course, are dramatically affected by the language model.

- Original annotation
  - pos labels
  - role tags

  - bracketing
  - sentence fragments
- conversion tool
  - head heuristics
  - role assignments
- HALogen generator
  - dictionary
  - morph
  - expressivity of input
  - grammar restrictions
  - language model

6.5 Related work

The only other system to have done a similar general-domain empirical evaluation is FERGUS. FERGUS applied a statistical tree model, TAG grammar, and ngram language model to 100 inputs consisting of a complete dependency tree labeled with fully inflected words but no roles. The realization problem was limited to determining constituent order. FERGUS achieved 58% correct using the NIST metric, a percentage comparable to that of the "Min spec" experiment just described. However, the differences in the input make the results difficult to compare. In contrast to HALogen, FERGUS offers no means for controlling the generation of a specific output exactly. On the other hand, it seems likely that a tree model such as the one used by FERGUS could improve HALogen's accuracy. See the experiment in Chapter 8 for evidence of this.

Corpus-based empirical coverage evaluations have been performed for limited domains by (Robin and McKeown, 1996) and (Kukich, 1983). The first was primarily concerned with evaluating the coverage and extensibility of a revision-based generation model, as opposed to the coverage of a system implementation. This contrasts with both the experiments described in this chapter, and with (Kukich, 1983). However, the latter was a template-based generator, not a general-purpose sentence realization system.

Robin and McKeown (1996) measured how coverage changed with an increase in training and test set size. The two test sets used for evaluation consisted of 130 and 240 sentences, respectively—an order of magnitude less than the experiments described in this chapter. Realization coverage (based on regular expression estimates, not actual generation) was 78.8% and 79.7% on the two respective test sets, where the first experiment was semi-automatically trained on 190 sentences, and the second on 320 sentences. The inputs were domain-specific propositions rather than sentence plans, and thus encompassed a broader range of generation tasks than just sentence realization.

Previous work has handled underspecification by supplying rigid defaults, ie. assume definite noun phrases, present tense, choose alphabetically first synonym, etc. The work described in this thesis differs by generating outputs for multiple (or all) possible values of the underspecified features, and allowing a statistical model to make a context-sensitive choice. It also differs in allowing a greater degree of underspecification than other systems. In particular, purely symbolic systems are hampered by the need to control overgeneration, which limits the degree of underspecification that they can handle while maintaining high quality output.

6.6 Discussion

The success of this approach for underspecification depends heavily on the strength and relevance of the statistical language model used. The experiments in this chapter quantify the degree of

success using a bigram model, and Chapter 5 offered a more qualitative analysis. In general, the approach works surprisingly well, although with room for improvement in the statistical model. It works because corpus co-occurence statistics implicitly encode a wide variety of linguistic constraints—from syntax to semantics to usage patterns. Although simple ngrams suffer obvious limitations in their ability to model various aspects of syntax, semantics, and usage—it seems that many of these limitations could be overcome with a more sophisticated model derived from suitably annotated corpora, as explored briefly in Chapter 8.

6.6.1 Symbolic/Statistical Integration

One important issue raised by the results of these experiments is the integration of symbolic and statistical methods. How should linguistic responsibility be assigned to each module?

The approach taken thus far is driven by practical concerns—those decisions that can be made satisfactorily by statistics can be left up to the statistical module. Additionally, those decisions for which there is insufficient symbolic knowledge can be left up to the statistical module when some output is better than no output. The rest are left up to the symbolic module.

Since statistics can be gathered automatically given a suitably annotated corpus, it is advantageous to shift as much responsibility as possible to the statistical module. The better the performance of the statistical model, the greater the responsibility it can take on, reducing the labor costs of building and using a realization system.

6.7 Summary

In conclusion, we empirically verified HALogen's coverage and accuracy using section 23 of the Penn Treebank as a test set. On a set of 2400 automatically-derived inputs, 80% produced output that was about 94% correct when the input was almost fully specified. Accuracy was measured using IBM Bleu scores and NIST simple string accuracy scores which compared outputs to the

original sentences. About 57% of the outputs were exact matches with the original. Using minimally specified inputs, accuracy was still more than 51%, 5% of which were exact matches. The flexibility that HALogen offers in the degree of specification of the input adds to its general-purposeness. Tasks like dialogue that require high quality output and need to exert significant control over the output can do so if they wish, while tasks like translation, that generally suffer from an inability to provide much information but that can accept lower quality output, still obtain output without providing many specification details.

Chapter 7

Preserving Ambiguities in Automatic Translation

This chapter discusses the problem of generating text that preserves certain ambiguities, a capability that is useful in applications such as machine translation. A hybrid symbolic/statistical generator like HALogen is relatively simple to extend to handle this task. This chapter was previously published as part of (Knight and Langkilde, 2000b).

7.1 Introduction

This chapter reports on an aspect of the natural language generation system HALogen which has been used as part of the Gazelle machine translation (MT) system (Knight et al., 1995a). In particular, we show how HALogen can generate English text that preserves unresolved ambiguities from non-English documents.

One of the first texts translated by Gazelle was a Japanese article with the following noun phrase:

1a. [IC chippu o seizou-suru noni tsukau]

dorai-echingu soochi ya suteppa

[*silicon chip construct for use*]

dry-etching device and stepper motor

In Japanese, relative clauses precede the nouns they modify. In this sentence, it is not clear whether the relative clause [shown bracketed] modifies the whole conjoined noun phrase (dry-etching device and stepper motor) or only the first noun in the conjunction (dry-etching device). Depending on the correct interpretation, reasonable English translations might be:

1b. ((dry-etching devices and stepper motors)
that are used to construct silicon chips)

1c. ((dry-etching devices that are used to construct
silicon chips) and stepper motors)

Both of these translations are "unsafe" because they commit to a particular interpretation that may turn out to be wrong. Translators who happen to know about computer equipment will prefer (1c) over (1b), but even (1c) has its problems—if we remove the parentheses, the sentence is likely to be misparsed by a reader!

The solution is to find a nice English translation that covers both interpretations:

1d. stepper motors and dry-etching devices
that are used to construct silicon chips

To get this "safe" translation, we simply re-order the nouns in the conjunction. It turns out that this conjunct-flipping technique is often employed by human translators. In this case, the idea is that the reader knows more about stepper motors than the translator does, and can disambiguate more easily. Many structural ambiguities provide opportunities for such ambiguity preservation.

Consider the English sentence:

2a. John saw the man with the telescope.

This sentence has two interpretations, depending on where the prepositional phrase attaches, but both interpretations are covered by the single Spanish translation:

2a. John vió al hombre con el telescopio.

This example shows how word-for-word MT engines frequently perform a simple-minded kind of ambiguity preservation, a fact that explains much of the commercial success of machine translation systems that operate between close language pairs. But even between Spanish and English, syntactic differences suggest careful handling. In the next phrase, the scope of the phrase-initial adjective is unclear:

3a. green eggs and ham

Again, conjunct flipping can help:

3b. (unsafe) huevos verdes y jamón

3c. (unsafe) huevos y jamón verdes

3d. (safe) jamón y huevos verdes

Another usefully vague construction is nominalization. Suppose that I witnessed somebody destroy something, and I only know that Nero was involved. There are safe and unsafe expressions for this situation:

4a. (unsafe) I witnessed Nero being destroyed.

4b. (unsafe) I witnessed the destruction by Nero.

4c. (safe) I witnessed Nero's destruction.

Pronoun reference can also benefit. Consider the English sentence

5. She saw the car in the window and wanted to buy it.

Although we must pick a gender for the German equivalent of "it," we can maintain the ambiguity if we select translations of "car" and "window" that have the same gender:

6a. (unsafe) Sie sah den Wagen$_{masc}$ im

Schaufenster$_{neut}$ und wollte ihn$_{masc}$ kaufen.

6b. (unsafe) Sie sah den Wagen$_{masc}$ im

Schaufenster$_{neut}$ und wollte es$_{neut}$ kaufen.

6c. (safe) Sie sah das Auto$_{neut}$ im

Schaufenster$_{neut}$ und wollte es$_{neut}$ kaufen.

Opportunities for ambiguity preservation occur frequently, but most ambiguities cannot of course be preserved. For example, it is very difficult to preserve lexical part-of-speech ambiguities, as in sentences like "Time flies." Semantic lexical ambiguities are more frequently preservable. In the English sentence "I went to the center," it is not clear whether "center" means "middle" or "an institution devoted to the study of something." When we translate to Spanish, we can simply say "Fui al centro," without resolving the ambiguity.

To sum up, ambiguity preservation is a technique often employed by human translators, among whom it is considered somewhat of an art. There are both opportunities and pitfalls. Ambiguity preservation is particularly interesting for machine translators, because they are not nearly as adept as humans at resolving source-language ambiguities.

7.2 Objective

Ideally, we would like to come up with an ambiguity preservation algorithm that is not tied to a specific language pair or even to specific linguistic constructions. Wedekind and Kaplan (1996) consider ambiguity preservation for generators that use LFG-style grammars. They show that the problem is undecidable: an interesting result, but one that is of little use to the practitioner. Emele and Dorna (1998) give an algorithm for preserving ambiguity that relies on packed LFG f-structure representations produced by the transfer component of an MT system. Shemtov (1998) describes the ambiguity preserving version of Kay (1996) chart generator. Like the above authors, this

chapter discusses ambiguity preservation across semantic inputs, thereby decoupling the problem from translation. For MT applications, we allow source language analysis to generate many possible meanings which are then picked over by an English-only generator.

This chapter contributes an ambiguity-preservation algorithm different from ones already proposed in (Shemtov, 1998; Emele and Dorna, 1998). The algorithm uses structures that are somewhat simpler. The goals are that (1) the method invented is simple, (2) it is efficient, (3) it is a minimal extension to an existing generator, and (4) it is fully implemented and tested.

Suppose we have three possible meaning representations (derived, say, from a single Spanish input sentence):

```
1.  (s1 / see
        :agent I
        :patient (a1 / (m1 / man)
                     / (w1 / woman)
                     :conj and)
        :instrument (t1 / telescope))

     -> I saw the man and woman with the telescope.
     -> With the telescope, I saw the man and woman.
           etc.

2.  (s2 / see
        :agent I
        :patient (a2 / (m2 / man)
                     / (w2 / woman
                           :possesses (t2 / telescope))
                     :conj and))

     -> I saw the man and woman with the telescope.
     -> I saw the woman holding the telescope and the man.
           etc.

3.  (s3 / see
        :agent I
        :patient (p3 / (a3 / (m3 / man)
                           / (w3 / woman)
                           :conj and)
                     :possesses (t3 / telescope))

     -> I saw the man and woman with the telescope.
     -> I saw the man and the woman that were holding the telescope.
```

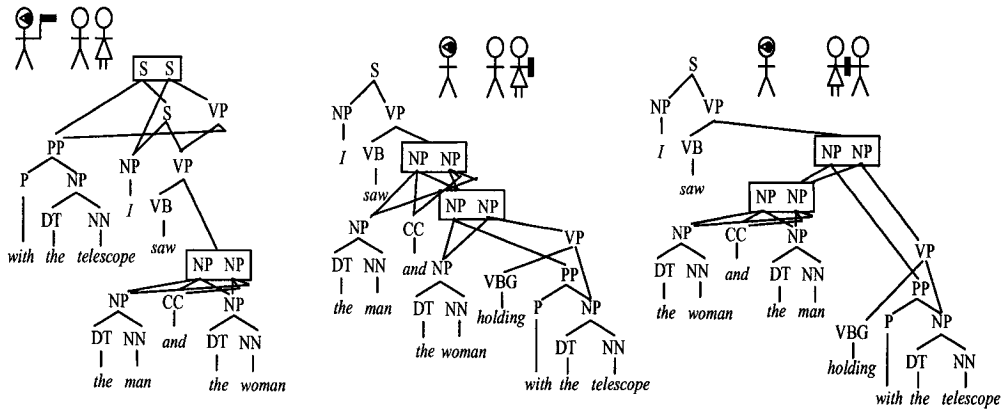


Figure 7.1: Three parse forests built from three semantic representations. The goal of ambiguity preservation is to identify the trees of just those sentences which cover all meanings.

```
etc.
```

HALogen can compute forests for these meanings independently, as illustrated in Figure 7.1. Sentences that express all meanings simultaneously (i.e., preserve ambiguity) are exactly the sentences that occur in all three forests. Here, "I saw the man and woman with the telescope" appears in all three, but "With the telescope, I saw the man" does not, nor does "I saw the man and the woman that were holding the telescope."

More formally, given two forests F1 and F2, the goal of ambiguity preservation is to produce a new forest NF that includes every tree T such that either (1) T is in F1 and T's terminal string is in F2, or (2) T is in F2 and T's terminal string is in F1. For each additional forest given—F3, F4, etc., the ambiguity preservation algorithm is repeated, intersecting each new forest NF with the result of the next forest: F3, then F4, etc. The following algorithm will do this:

7.3 Algorithm

Due to HALogen's implementation there is an efficient algorithm for computing ambiguity-preserving trees. This algorithm is based on two aspects of HALogen's implementation. First, HALogen uses a cache to store the results of mapping sub-pieces of the input to sub-forests. The cache avoids duplicate processing during generation because if a particular sub-piece of an input occurs in multiple inputs, the result can retrieved from the cache rather than being regenerated. The effect of the cache is that different forests will share sub-trees if they have sub-units of semantic representation in common. Such shared trees inherently represent an intersection between forests, and automatically preserve ambiguity between different inputs. Unnecessary processing can be avoided by recognizing this in the ambiguity preserving algorithm, rather than continuing to search within the shared sub-tree.

```
function tree-intersect (A B)
if A and B are equal then return the pair (A,B);
if there are no constituents in A or B then return nil;
if the highest node of A OR the highest node of B is a leaf node,
. then return nil;
if the highest nodes of A and B are the same then
. high-node := highest node of A;
. res1 := tree-intersect(left-of-highest(A),left-of-highest(B));
. res2 := tree-intersect(right-of-highest(A),right-of-highest(B));
. for every tree pair r1 in res1
. . for every tree pair r2 in res2
. . . A1 := append the A tree of res1 to the left of high-node
. . . . and the A tree of res2 to the right of the high node;
. . . B1 := append the B tree of res1 to the left of high-node
. . . . and the B tree of res2 to the right of the high node;
. . . add the pair (A1,B1) to the tree-pair list;
. return the tree-pair list.
if the highest node of A is greater than the highest node of B,
. for each disjunctive set S of children of A
. . A1 := substitute S for high-node in A
. . res := tree-intersect(A1,B);
. for every tree pair r in res
. . A2 := replace children in former positions of S
. . . with new parent node
. return list of tree-pairs (A2,B);
otherwise, do the previous seven steps
. reversing the roles of A and B.
```

Figure 7.2: Tree-Intersection Algorithm

The second aspect of HALogen's implementation that facilitates the preservation of ambiguities is the labeling of nodes in the tree. When a sub-forest is generated, it is identified with a unique numeric label. Successive sub-forests are assigned numerically increasing labels. A side effect of this labeling is that a partial order on the nodes in forest is guaranteed, such that a parent node always has a higher-numbered label than any of its children. The ordering on the nodes can be used to guide the search for an intersection between forests. Higher-numbered nodes are always expanded first, ensuring that a shared-subtree will be discovered, if it exists, without doing any unnecessary processing. When a shared sub-tree is found and it is the highest-numbered node among the nodes being examined, the divide-and-conquer method can be used to split the remaining search into two independent sub-problems, thereby significantly reducing the overall search space.

The algorithm in Figure 7.2 exploits these two aspects of HALogen's implementation to achieve more efficient processing. The A and B arguments to the *tree-intersect* function are lists of sequential nodes. Initially, each list contains only the root node of the respective trees being compared. The algorithm is not limited to comparing whole forests, but equally well handles sub-forests and arbitrary sequences of sibling nodes. This gives it another advantage over independent algorithms since it can be integrated into the generation process, permitting the preservation of ambiguities within inputs as well as across inputs. The algorithm returns a list of paired trees sharing the same sequence of words, with their differing syntactic structure preserved.

If no forest intersection exists that preserves the ambiguity, than the original forests are simply passed along as a disjunction to the statistical ranking module. This algorithm has been implemented and integrated into the HALogen generator.

7.4 Implementation Details

The ambiguity preservation process is potentially triggered whenever an input contains a disjunction, as described in Section 2.4. This is represented as a parenthesized list with "*OR*" as the first element, and two or more additional elements representing inputs or input fragments between which a choice needs to be made.

There are two different ways of controlling the ambiguity preservation process in the implementation. First, there is a general system flag that can be set to true or false to turn on or off the ambiguity preservation procedure. If turned off, the alternative forests generated by the disjoined input fragments are simply packed into the forest as alternatives, like other generation choices are, and no deliberate preservation of ambiguity is performed. The statistical module will decide between the alternatives, as usual. If turned on, only the result forest that remains from intersecting the respective forests produced by the input fragments is passed on to the statistical module, which makes any remaining decisions in generating the input fragment.

As an alternative to the system flag approach, a finer-grained method of controlling the ambiguity preservation process uses a different disjunction symbol "*AOR*", which indicates to the generator that the ambiguity preservation procedure should be used to process the corresponding input fragments. If the general system ambiguity-preservation flag is turned off, this gives a client program the option of two different kinds of disjunctions, and thus finer-grained control over when the ambiguity preservation procedure is triggered.

7.5 Discussion

It was very easy to adapt HALogen to perform a wide range of ambiguity preservations. Partly, this is because of the simple lattice and forest representations it uses. HALogen dispenses with the feature notations that bedevil Wedekind's generator and Shemtov's ambiguity preserving algorithm. Experience indicates that statistically-gathered knowledge can simulate these features

fairly well, while also taking into account the collocational properties of language that are very hard to model with features.

If an ambiguity is not preservable, a union of the parse forests is performed instead of an intersection. This amounts to letting the statistical ranker choose the most fluent sentence regardless of which meaning it expresses. This works better than expected, because strange meanings often make for strange sentences.

Future work remains to empirically evaluate the effect of using the ambiguity preservation procedure in general. Questions to be answered include: "Is it always desirable to preserve ambiguity when possible?" and "How often does an ambiguity-preserving option exist?"

Chapter 8

Statistical Model of Syntax

8.1 Overview

This chapter describes an exploratory experiment to evaluate the effect on sentence generation output quality of using a Charniak-style probabilistic model of syntax. Most of the chapter has previously been published in (Daume et al., 2002).

The motivation for the experiment is the success that probabilistic syntax models have had in parsing (Collins, 1997; Charniak, 2001). At the same time, however, the superiority of statistical syntax models over ngrams in other applications such as speech recognition (Chelba and Jelinek, 1998) or generation has yet to be convincingly demonstrated.

To evaluate the impact of a probabilistic lexicalized syntax model on the quality of generation output, this experiment uses the HALogen system to generate from inputs that were automatically derived from the Penn Treebank (Marcus, Santorini, and Marcinkiewicz, 1993). During generation, a bigram model is used to rank alternate outputs and the most likely sentence is selected for comparison to the original reference sentence in the Penn Treebank. Trigram scores and Charniak-parser scores are then computed for both the output sentence and the original reference sentence. The three models–bigram, trigram, and Charniak-syntax—are then measured

```
(H37 :ADJUNCT "earlier"
     :LOGICAL-SUBJECT (H5 / "company")
     / "announce"
     :ADJUNCT (H34 :ADJUNCT "its"
                   / "plan"))
```

Figure 8.1: An underspecified input for the sentence, "Earlier the company announced its plans."

according to how often they prefer the reference sentence (which serves as a gold standard) over the imperfect output of the sentence generator.

8.2 Experimental setup

From the sentences in section 23 of the Penn Treebank, inputs to the HALogen generator system were automatically derived and then re-generated. The inputs derived were feature-value dependency structures, where the features represent syntactic relationships between values, and the values were either words in root form or a nested feature-value structure. The inputs were underspecified with respect to properties such as part-of-speech category, tense, voice, and number, as well as constituent order and some closed-class words like auxiliary verbs and determiners. Underspecification tests the ability of a language model to pick the best solution from among a set of choices.

HALogen overgenerates possible expressions for an input, in part because of enumerating possible choices for underspecified details. It then ranks potential outputs using an ngram model. Both bigram and trigram models were available, but because the trigram model takes two orders of magnitude more time to use (more than 40 minutes per sentence, versus 1 minute per sentence for the bigram model), the sentences were generated using the bigram model. One hundred sentences were randomly chosen from among the successfully generated outputs. Each was paired with its

reference Treebank sentence, and then trigram and Charniak-parser scores were calculated for all the sentences.

8.3 Results

About 3% of the bigram-output/reference-sentence pairs were exact matches. Sentences within pairs had very similar lengths in all cases. The average sentence length was 24 tokens.

Trigram scores for reference Treebank sentences scored better than the output of the generator system 71% of the time. In comparison, Charniak-parse scores preferred the reference Treebank sentences 83% of the time. This indicates that the generator system would benefit even more from a statistical model of syntax than from trigrams.

Chapter 9

Conclusion

This thesis has addressed the problem of general-purpose sentence realization. It's main contributions are

- a symbolic generator with a core knowledge base of 255 mapping rules that map inputs to a packed set of potential outputs, represented as a forest. The rule formalism allows syntactic constraints to be localized and facilitates broad coverage of syntactic phenomena. It also enables paraphrases to be concisely defined. The approach separates the problem of enumerating the available options from the problem of deciding between them. It does so by passing multiple options forward and delaying decision-making. The effect is a partial-order pipeline architecture that allows flexibility in the order of decision-making and handles more subtask interdependencies, while maintaining modularity and efficiency.
- a flexible definition of inputs to the system, allowing them to vary along three independent dimensions: level of abstraction, degree of specification, and granularity. Inputs can be specified anywhere along an abstraction continuum from the syntactic level to semantic to domain-specific. Clients simultaneously have access to rich paraphrasing power and to complete control over the output. The task of constructing inputs is simplified since inputs can be underspecified, easing the information burden on clients. Inputs can also contain a mix of fine-grained and template specifications.

- a packed forest structure to store the intermediate results from the symbolic generator, and a bottom-up dynamic programming algorithm for ranking individual trees in a forest using word ngram statistics collected over large text corpora. I showed empirically that this combination of representation and algorithm performs exponentially better than word lattices used previously in speech recognition and natural language generation, making the search for an optimal solution tractable.

- a complete system named HALogen integrating the three elements above with a large dictionary (on the order of 100,000 words and concepts) and a 250-million word ngram language model. An online demo is available at http://www.isi.edu/licensed-sw/halogen/index.html, together with a link for downloading the entire system.

9.1 Impact

HALogen's strategy of performing general-purpose realization by first enumerating candidate outputs and later ranking them statistically has inspired similar approaches. Namely, the FERGUS system (Bangalore and Rambow, 2000a; Bangalore, Rambow, and Whittaker, 2000b; Bangalore and Rambow, 2000c) successfully applied the same stategy with a statistical syntax model rather than an ngram model. Malouf (2000) applied the strategy to the specific problem of adjective ordering.

The forest representation has been the subject of additional theoretical analysis in (Nederhof and Satta, 2002), which developed an algorithm for symbolically parsing the sentences represented by a forest. Oberlander and Brew (2000) performed theoretical work on stochastic sentence planning that could leverage a HALogen-style statistical sentence realizer.

HALogen's statistical ranking module has been used to do summarization via sentence compression (Knight and Marcu, 2000), and as a decoder for syntax-based statistical machine translation (Yamada and Knight, 2002). The system as a whole has been adapted for use in machine translation by (Dorr, Habash, and Traum, 1998).

9.2 Limitations and Future Work

Limitations of the system include:

- unsatisfactory quality resulting from reliance on an ngram model when inputs are very underspecified. This makes the system less practical for applications that require high quality output.
- currently performs fewer subtasks than previous reusable generation systems. Specifically, it does only rudimentary mapping from semantics to syntax, and does not perform syntactically-motivated pronomialization or ellipsis.
- does not currently support incremental generation, which is useful in some applications for meeting real-time constraints, or efficiently generating longer, more complex texts through a revision process (Newmann and Finkler, 1990).
- does not currently support non-deterministic generation. The system always produces the same output for a given input. However, some applications would prefer to have variation, to make the system seem more natural.

Plans for future work include applying a probabilistic model of syntax to improve quality, integrating statistical ranking with symbolic generation to improve the efficiency of generation, adding sentence planning capabilities such as syntactic pronomialization and ellipsis, and evaluating the system in the context of specific applications such as machine translation and human-computer dialogue.

References

Bangalore, S. and O. Rambow. 2000a. Using tag, a tree model, and a language model for generation. In *Proceedings of 1st of the International Natural Language Generation Conference.*

Bangalore, S. and O. Rambow. 2000c. Exploiting a probabilistic hierarchical model for generation. In *Proceedings of the International Committee On Computational Linguistics (COLING).*

Bangalore, S., O. Rambow, and S. Whittaker. 2000b. Evaluation metrics for generation. In *Proceedings of 5th Workshop on TAG.*

Bateman, J. 1996. KPML development environment — multilingual linguistic resource development and sentence generation. Technical report, German Centre for Information Technology (GMD).

Cahill, L., C. Doran, R. Evans, C. Mellish, D. Paiva, M. Reape, D. Scott, and N. Tipper. 1999. In search of a reference architecture for NLG systems. In *Proceedings of the European Workshop on Natural Language Generation (EWNLG).*

Charniak, Eugene. 2001. Immediate-head parsing for language models. In *Proceedings of the 39th Annual Meeting of the Association for Computational Linguistics.*

Chelba, C. and F. Jelinek. 1998. Exploiting syntactic structure for language modeling. In *Proceedings of the International Committee On Computational Linguistics/Conference of the Association for Computational Linguistics (COLING/ACL)).*

Clarkson, P.R. and R. Rosenfeld. 1997. Statistical language modeling using the cmu-cambridge toolkit. In *Proceedings of the European Speech Communication Association (ESCA) Eurospeech.*

Collins, M. 1997. Three generative, lexicalised models for statistical parsing. In *Proceedings of the Conference of the Association for Computational Linguistics (ACL).*

Daume, H., K. Knight, I. Langkilde-Geary, D. Marcu, and K. Yamada. 2002. Experiments using a statistical model of syntax. In *Proceedings of the International Natural Language Generation Conference.*

Dorr, B., N. Habash, and D. Traum. 1998. A thematic hierarchy for efficient generation from lexical-conceptual structure. In *Proceedings of the Third Conference of the Association for MT in the America's (AMTA)*, pages 333–343.

Dras, M. 1999. *Tree Adjoining Grammar and the Reluctant Paraphrasing of Text.* Ph.D. thesis, Macquarie University, Australia.

Elhadad, M. 1993. FUF: The universal unifier—user manual, version 5.2. Technical Report CUCS-038-91, Columbia University.

Elhadad, M. and J. Robin. 1998. Surge: a comprehensive plug-in syntactic realization component for text generation. In *http://www.cs.bgu.ac.il/ elhadad/pub.html.*

Emele, M. and M. Dorna. 1998. Ambiguity preserving machine translation using packed representations. In *Proceedings of the Joint Conference of the International Committee On Computational Linguistics and Association of Computational Linguistics (COLING/ACL).*

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

Joshi, A. 1987. The relevance of tree adjoining grammar to generation. In *Natural Language Generation.* Martinus Nijhoff, Dordrecht, Netherlands.

Kay, M. 1996. Chart generation. In *Proceedings Association for Computation Linguistics (ACL).*

Knight, K., I. Chander, M. Haines, V. Hatzivassiloglou, E. Hovy, M. Iida, S. K. Luk, R. Whitney, and K. Yamada. 1995a. Filling knowledge gaps in a broad-coverage MT system. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI).*

Knight, K. and V. Hatzivassiloglou. 1995b. Two-level, many-paths generation. In *Proceedings of the Association for Computation Linguistics (ACL).*

Knight, K. and I. Langkilde. 2000b. Preserving ambiguities in generation via automata intersection. In *Proceedings of the American Association of Artificial Intelligence (AAAI).*

Knight, K. and S. Luk. 1994. Building a large-scale knowlege base for machine translation. In *Proceedings of the American Association of Artificial Intelligence (AAAI).*

Knight, K. and D. Marcu. 2000. Statistics-based summarization — step one: Sentence compression. In *Proceedings of the American Association of Artificial Intelligence (AAAI).*

Kukich, K. 1983. *Knowledge-based report generation: A Knowledge Engineering Approach to Natural Language Report Generation.* Ph.D. thesis, University of Pittsburg.

Langkilde, I. 2000. Forest-based statistical sentence generation. In *Proceedings of the North American Chapter of the Association of Computational Linguistics (NAACL).*

Langkilde, I. and K. Knight. 1998a. Generation that exploits corpus-based statistical knowledge. In *Proceedings of the International Committee On Computational Linguistics-Association of Computational Linguistics (COLING/ACL).*

Langkilde, I. and K. Knight. 1998b. The practical value of n-grams in generation. In *Proceedings of the International Natural Language Generation Workshop (INLGW).*

Langkilde-Geary, I. 2002. An empirical verification of coverage and correctness for a general-purpose sentence generator. In *Proceedings of the International Natural Language Generation Conference (INLG).*

Lavoie, B. and O. Rambow. 1997. RealPro – a fast, portable sentence realizer. In *Proceedings of the Conference on Applied Natural Language Processing (ANLP).*

Malouf, R. 2000. The order of prenominal adjectives in natural language generation. In *Proceedings of the 38th Annual Meeting of the Association of Computational Linguistics (ACL)*, pages 85–92.

Marcus, M., B. Santorini, and M. Marcinkiewicz. 1993. Building a large annotated corpus of english: the Penn treebank. *Computational Linguistics*, 19(2).

Matthiessen, C. and J. Bateman, editors. 1991. *Text Generation and Systemic-Functional Linguistics.* Pinter Publishers, London.

Melcuk, I. and N. Pertsov. 1987. *Surface-Syntax of English–A Formal Model within the Meaning-Text Frame work.* Benjamins, Amsterdam/Philadelphia.

Meteer, M. 1990. *The Generation Gap - the problem of expressibility in text planning.* Ph.D. thesis, University of Massachusetts.

Meteer, M., D. McDonald, S. Anderson, D. Forster, L. Gay, A. Iluettner, and P. Sibun. 1987. Mumble-86: Design and implementation. Technical Report COINS 87-87, University of Massachussets, Amherst, MA.

Miller, G. 1990. Wordnet: An on-line lexical database. *International Journal of Lexicography*, 3(4). (Special Issue).

Nederhof, M. and G. Satta. 2002. Parsing non-recursive cfgs. In *Proceedings of Association for Computation Linguistics (ACL).*

Newmann, G. and W. Finkler. 1990. A head-driven approach to incremental and parallel generation of syntactic structures. In *Proceedings of the International Committee On Computational Linguistics (COLING).*

Oberlander, J. and C. Brew. 2000. Stochastic text generation. *Proceedings of the Royal Society A.*

Papineni, K., S. Roukos, T. Ward, and W. Zhu. 2002. Bleu: a method for automatic evaluation of machine. In *Proceedings Association for Computation Linguistics (ACL).*

Pollard, C. and I. Sag. 1994. *Head Driven Phrase Structure Grammar.* University of Chicago Press.

Quirk, R. and S. Greenbaum. 1973. *A Concise Grammar of Contemporary English.* Harcourt Brace Jovanovich, New York.

Reiter, E. 1994. Has a consensus NL generation architecture appeared, and is it psycholinguistically plausible? In *Proceedings of the International Natural Language Generation Workshop (INLGW).*

Reiter, E. 1995. NLG vs. templates. In *Proceedings of the European Natural Language Generation Workshop (ENLGW).*

Reiter, E. 1999. Shallow vs. deep techniques for handling linguistic constraints and optimisations. In *Proceedings KI Workshop on May I Speak Freely.*

Robin, J. 1994. *Revision-based generation of natural language summaries providing historical background: corpus-based analysis, design, implementation and evaluation.* Ph.D. thesis, Columbia University.

Robin, J. and K. McKeown. 1996. Empirically designing and evaluating a new revision-based model for summary generation. *Artificial Intelligence*, 85(1-2).

Shemtov, H. 1996. Generation of paraphrases from ambiguous logical forms. In *Proceedings of the International Committee On Computational Linguistics'96.*

Shemtov, H. 1998. *Ambiguity Management in Natural Language Generation.* Ph.D. thesis, Department of Linguistics, Stanford University.

Smedt, K. De, H. Horacek, and M. Zock. 1996. Architectures for natural language generation. In *Trends in Natural Language Generation.* Springer, Berlin.

Stolcke, A. 1997. Linguistic knowledge and empirical methods in speech recognition. *AI Magazine*, 18(4):25–31.

Wedekind, J. and R. Kaplan. 1996. Ambiguity-preserving generation with LFG- and PATR-style grammars. *Computational Linguistics*, 22(4).

Yamada, K. and K. Knight. 2002. A decoder for syntax-based statistical mt. In *Proceedings of the Association of Computational Linguistics (ACL).*

Appendix A

Examples of Inputs and Lexical Resources of Sentence Realizers

Examples of inputs from MUMBLE, Penman/KPML, FUF/Surge, RealPro, FERGUS, Nitrogen, and HALogen are shown below for a rough eyeball comparision. These examples are taken from each system's own documentation.

```
(discourse-unit <R1>
  :head (general-clause  <R2>
          :head (chase  <R3>
                  (general-np <R4>
                    :head (np-proper-name ''Fluffy'' <R5>
                    :accessories (:number singular
                                  :determiner-policy no-determiner))
                  (general-np <R6>
                    :head (np-common-noun ''mouse'')  <R7>
                    :accessories (:number singular
                                  :determiner-policy kind)
                    :further-specifications
                      ((:specification
                          (predication_to-be *self*
                                             (adjective ''little''))
                        :attachment-function restrictive-modifier))
                       )))
          :accessories (:tense-modal present
                        :progressive
                        :unmarked)) )
```

Mumble input for the sentence "Fluffy is chasing a little mouse."

Figure A.1: Mumble input

```
((V-591 / (STUDY)
        :SPATIAL-LOCATING
            (V-585 / (MUNICH THREE-D-LOCATION NAMED-OBJECT OBJECT)
                   :NAME MUNICH)
        :TEMPORAL-LOCATING
            (V-587 / (THREE-D-TIME)
                   :NAME |1890|)
        :INSTRUMENTAL
            (V-589 / (KOTSCHENREITER NAMED-OBJECT MALE OBJECT)
                   :NAME KOTSCHENREITER)
        :ACTOR (V-590 / (BEHRENS NAMED-OBJECT MALE OBJECT)
                   :NAME BEHRENS)
        :TENSE PAST))
```

Penman/KPML input for the sentence "He studied in Munich in 1890 with Kotschenreiter."

Figure A.2: Penman/KPML input

```
(lexical-item
  :name FEED
  :spelling "feed"
  :sample-sentence  "The data is fed into the computer."
  :features  (VERB INFLECTABLE UNITARYSPELLING S-IRR PASTFORM
              EDPARTICIPLEFORM LEXICAL NOT-CASEPREPOSITIONS
              NOT-TOCOMP NOT-QUESTIONCOMP NOT-MAKECOMP NOT-ADJECTIVECOMP
              DOVERB DISPOSAL EFFECTIVE NOT-SUBJECTCOMP NOT-PARTICIPLECOMP
              NOT-BAREINFINITIVECOMP OBJECTPERMITTED
              NOT-OBJECTNOTREQUIRED NOT-COPULA PASSIVE INDIRECTOBJECT
              NOT-THATCOMP)
  :properties  ((PASTFORM "fed" )(EDPARTICIPLEFORM "fed" ))
  :date  "Monday the twenty-third of February, 1987; 4:51:40 pm"
  :editor  "Smith")
```

Figure A.3: Example of a KPML lexical entry

```
WORK   [ tense:past ]
(
  I    PROGRAMMER    [ article:indef  number:sg ]  )
  II   PROJECT   [ article:def  number:sg  ]  )
)
END:
```

RealPro input for the sentence "A programmer worked on the project."

Figure A.4: RealPro input

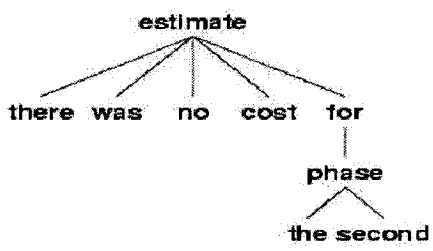
```
LEXEME:          WORK
CATEGORY:        verb
FEATURES:        [ ]
GOV-PATTERN:     [
                    DSYNT-RULE:

                    [ ( WORK  II  X2 ) ]                  |  [  ]
                    <-->
                    [ ( WORK completive1   ON1 )
                      ( ON1     prepositional   X2   )  ]   |  [  ]
                 ]
MORPHOLOGY:      [    ( [ ] work [ reg ] )   ]
```

Figure A.5: Example of a RealPro lexical entry

```
((cat clause)
 (process      ((type material)
                (effect-type creative)
                (lex ''score'')
                (tense past)))
 (participants ((agent ((cat proper)
                        (head ((cat person-name)
                               (first-name ((lex ''Michael'')))
                               (last-name  ((lex ''Jordan'')))))))))
                (created ((cat np)
                          (cardinal ((value 36)))
                          (definite no)
                          (head ((lex ''point'')))))))))
```

FUF/Surge input for the sentence "Michael Jordan scored 36 points."

Figure A.6: FUF/SURGE input



FERGUS input for the sentence "There was no cost estimate for the second phase."

Figure A.7: FERGUS input

```
( A / (B / |admire,look|)
    :AGENT (V / (W / |visitor|)
               :AGENT-OF (C / (D / |arrive,get|)
                             :DESTINATION (J / |Nihon|)))
    :PATIENT (M / "Mount Fuji"))
```

Nitrogen input for the sentence "Visitors who come to Japan admire Mount Fuji."

Figure A.8: Nitrogen input

```
( A / (B / ADMIRE
         :tense present)
    :subject (V / (W / VISITOR
                     :nquant plural
                     :definiteness indef)
                :rel-clause (C / (D / COME)
                                 :destination (J / JAPAN)))
    :object (M / "Mount Fuji"))
   -------------------------------

( A / (B / ADMIRE)
    :ARG (V / (W / VISITOR)
               :ADJUNCT (C / (D / COME)
                            :ARG (J / JAPAN)))
    :ARG (M / "Mount Fuji"))
```

Two more ways, among several possible variations (including the input in Figure A.8), of representing the input in HALogen for the sentence "Visitors who come to Japan admire Mount Fuji."

Figure A.9: HALogen Inputs

Appendix B

Verbal Properties

B.1 Examples of Verbal Properties in Input

```
    (e1 / eat
        :person 3s
        :logical-subject (d1 / dog)
        :logical-object (b1 / bone))

The dog eats bones.

    (e1 / eat
        :person 3p
        :logical-subject (d1 / dog)
        :logical-object (b1 / bone))

The dogs eat bones.

    (e1 / eat
        :taxis perfect
        :logical-subject (d1 / dog)
        :logical-object (b1 / bone))

The dog has eaten bones.

    (e1 / eat
        :aspect continuous
        :logical-subject (d1 / dog)
        :logical-object (b1 / bone))

The dog is eating bones.

    (e1 / eat
        :voice passive
        :logical-subject (d1 / dog)
        :logical-object (b1 / bone))

The bone was eaten by the dog.

    (e1 / eat
        :taxis perfect
        :aspect continuous
        :logical-subject (d1 / dog)
```

```
        :logical-object (b1 / bone))
```

The dog has been eating bones.

```
    (e1 / eat
        :aspect continuous
        :voice passive
        :logical-subject (d1 / dog)
        :logical-object (b1 / bone))
```

The bone was being eaten by the dog.

```
    (e1 / eat
        :taxis perfect
        :voice passive
        :logical-subject (d1 / dog)
        :logical-object (b1 / bone))
```

The bone has been eaten by the dog.

```
    (e1 / eat
        :taxis perfect
        :aspect continuous
        :voice passive
        :logical-subject (d1 / dog)
        :logical-object (b1 / bone))
```

The bone has been being eaten by the dog.

```
    (e1 / eat
        :tense past
        :taxis perfect
        :aspect continuous
        :voice passive
        :logical-subject (d1 / dog)
        :logical-object (b1 / bone))
```

The bone had been being eaten by the dog.

```
    (e1 / eat
        :modal may
        :taxis perfect
        :aspect continuous
        :voice passive
        :logical-subject (d1 / dog)
        :logical-object (b1 / bone))
```

The bone may have been being eaten by the dog.

```
    (e1 / eat
        :mood infinitive
        :taxis perfect
        :aspect continuous
        :voice passive
        :logical-subject (d1 / dog)
```

```
        :logical-object (b1 / bone))

    the bone to have been being eaten by the dog

        (e1 / eat
            :mood present-particple
            :taxis perfect
            :aspect continuous
            :voice passive
            :logical-subject (d1 / dog)
            :logical-object (b1 / bone))

    the bone having been being eaten by the dog
```

B.2 Matrix of Verbal Property Combinations

This table lists the output of every possible combination of the verbal properties and their values, except :polarity and :subject-position. These two are not included because of table size constraints and the regularity of their effect. The table uses the verb "eat" as an example, but also lists the output of the verbal properties when applied to the main verb "be", which is a uniquely irregular case.

- :TAXIS perfect, none
- :ASPECT continuous, simple
- :VOICE active, passive
- :MOOD infinitive, to-infinitive, imperative, present-participle, past-participle, indicative
- :TENSE present, past
- :MODAL should, would, could, may, might, must, can, will
- :PERSON s (3s), p (1s 1p 2s 2p 3p)
- :SUBJECT-POSITION default, post-aux, post-vp

LEGEND

```
s  = 3s
p  = 1s,1p,2s,2p,3p
A  = same as above
-- = not grammatical
X  = ignored

Special cases for verb "be":
If (verb="be" or aspect=continuous, or voice=passive),
  If mood=indicative, taxis=none, person=1s,
    If tense=present,
      IF polarity=negative, and :question=yes,
        THEN inflection of "be" is "are";
      ELSE inflection of "be" is "am";
    ELSE inflection of "be" is "was";
(Basically, anywhere "are" or "were" appears in the table,
 this special case applies.)
```

```
Value of Modal feature can be any of:
  "can" "could" "may" "might" "must" "should" "will" "would"
If modal feature is present, then tense and person are ignored.

Mood=imperative has same table values as mood=infinitive,
(just different syntactic tag).
Mood=to-infinitive has same table values as non-null Modal, with "to"
substituted for the modal.
```

Mood	Modal	Taxis	Aspect	Voice	Tense	Person	BE	EAT
indicative	none	none	simple	active	present	s	is	eats
A	A	A	A	A	A	p	are	eat
A	A	A	A	A	past	s	was	ate
A	A	A	A	A	A	p	were	ate
A	A	A	A	passive	present	s	–	is eaten
A	A	A	A	A	A	p	–	are eaten
A	A	A	A	A	past	s	–	was eaten
A	A	A	A	A	A	p	–	were eaten
A	A	A	continuous	active	present	s	is being	is eating
A	A	A	A	A	A	p	are being	are eating
A	A	A	A	A	past	s	was being	was eating
A	A	A	A	A	A	p	were being	were eating
A	A	A	A	passive	present	s	–	is being eaten
A	A	A	A	A	A	p	–	are being eaten
A	A	A	A	A	past	s	–	was being eaten
A	A	A	A	A	A	p	–	were being eaten
A	A	perfect	simple	active	present	s	has been	has eaten
A	A	A	A	A	A	p	have been	have eaten
A	A	A	A	A	past	s, p	had been	had eaten
A	A	A	A	passive	present	s	–	has been eaten
A	A	A	A	A	A	p	–	have been eaten
A	A	A	A	A	past	s, p	–	had eaten
A	A	A	continuous	active	present	s	has been being	has been eating
A	A	A	A	A	A	p	have been being	have been eating
A	A	A	A	A	past	s, p	had been being	had been eating
A	A	A	A	passive	present	s	–	has been being eaten
A	A	A	A	A	A	p	–	have been being eaten
A	A	A	A	A	past	s, p	–	had been being eaten
A	will	none	simple	active	modal	X	will be	will eat
A	A	A	A	passive	A	A	–	will be eaten
A	A	A	continuous	active	A	A	will be being	will be eating
A	A	A	A	passive	A	A	–	will be being eaten
A	A	perfect	simple	active	A	A	will have been	will have eaten
A	A	A	A	passive	A	A	–	will have been eaten
A	A	A	continuous	active	A	A	will have been being	will have been eating
A	A	A	A	passive	A	A	–	will have been being eaten

Table B.1: Matrix of Verbal Property Combinations, where Mood=Indicative

Mood	Modal	Taxis	Aspect	Voice	Tense	Person	BE	EAT
infinitive	X	none	simple	active	X	X	be	eat
A	A	A	A	passive	A	A	–	be eaten
A	A	A	continuous	active	A	A	be being	be eating
A	A	A	A	passive	A	A	–	be being eaten
A	A	perfect	simple	active	A	A	have been	have eaten
A	A	A	A	passive	A	A	–	have been eaten
A	A	A	continuous	active	A	A	have been being	have been eating
A	A	A	A	passive	A	A	–	have been being eaten
pres-part	X	none	simple	active	X	X	being	eating
A	A	A	A	passive	A	A	–	being eaten
A	A	A	continuous	X	A	A	–	–
A	A	perfect	simple	active	A	A	having been	having eaten
A	A	A	A	passive	A	A	–	having been eaten
A	A	A	continuous	active	A	A	having been being	having been eating
A	A	A	A	passive	A	A	–	having been being eaten
past-part	X	none	simple	active	X	X	been	eaten
A	A	A	A	passive	A	A	–	been eaten
A	A	A	continuous	active	A	A	been being	been eating
A	A	A	A	passive	A	A	–	been being eaten
A	A	perfect	X	X	A	A	–	–

Table B.2: Matrix of Verbal Property Combinations, where Mood is NOT Indicative

Appendix C

Example Inputs

C.1 Alphabetized Examples of Input Relations

NOTE: These examples are intended to help define the meaning of the relations. They are not intended to be samples of actual output, since actual output depends heavily on the statistical language model used. Instead, the sentence shown for each input should be interpreted as an ideal output.

```
---------------------------------------------------------------------------
:ACCOMPANIER (= :INCLUSIVE)

        (g1 / go
            :agent (b1 / boy)
            :accompanier (d1 / dog))

    The boy went with the dog.

        (e1 / eat
            :AGENT (p1 / person
                       :PRO she)
            :PATIENT (n1 / pasta
                         :ACCOMPANIER (m1 / meatballs))
            :MEANS (f1 / fork)
            :ACCOMPANIER (g1 / gentleman
                             :GENERAL-POSSESSION (s1 / sunglasses)))

    She ate pasta with meatballs with a fork with the gentleman with sunglasses.

---------------------------------------------------------------------------
:ADJUNCT (maps to :premod, :postmod, and :withinmod)

       (g1 / go
           :agent (b1 / boy)
           :adjunct (q1 / quickly))

    The boy went quickly. Quickly, the boy went. The boy quickly went.

       (g1 / go
           :agent (b1 / boy)
           :adjunct (q1 / house
                        :anchor (h1 / into)))
```

```
    Into the house, the boy went. The boy went into the house,.

       (g1 / play
           :agent (b1 / boy
                      :adjunct (q1 / young))

    The young boy played.

       (b1 / boy
           :adjunct (g1 / corner
                        :anchor (c1 / in)))

    the boy in the corner

       (b1 / red
           :adjunct (v1 / very)
           :adjunct (b1 / bright)

    very bright red

--------------------------------------------------------------------------------
:AGENT (= :SAYER, :SENSER)

       (g1 / go
           :agent (b1 / boy))

    The boy went.

--------------------------------------------------------------------------------
:AGENT-OF

        (b1 / boy
            :agent-of (s1 / sing))
    The boy who sang

        (c1 / citizen
            :agent-of (o1 / oppose
                    :patient (t1 / tobacco))
            :mood fragment)

    Citizens Against Tobacco

--------------------------------------------------------------------------------
:ANCHOR

        (l1 / live
            :agent (b1 / boy)
            :spatial-locating (d1 / dock
                                  :anchor (n1 / near)))

    The boy lives near the dock.

        (l1 / leave
            :agent (b1 / boy)
            :temporal-locating (a1 / attack
```

```
                              :anchor (d1 / during)))
```

The boy left during the attack

```
   (s1 / stay
       :agent (g1 / girl)
       :premod (g3 / (g2 / go
                           :agent (b1 / boy))
                   :anchor (e1 / ``even if'')))
```

Even if the boy goes, the girl will stay.

:BCPP (stands for Before Conjunction Punctuation--Predicate)

```
    (c1 / (c2 / California)
        / (a1 / Arizona)
        / (a1 / Texas)
        / (a1 / Nevada)
        / (a1 / Washington)
        :CONJ ``as well as''
        :BCPP +)
```

California, Arizona, Texas, Nevada, as well as Washington

```
    (c1 / (c2 / California)
        / (a1 / Arizona)
        / (a1 / Texas)
        / (a1 / Nevada)
        / (a1 / Washington)
        :CONJ ``as well as''
        :BCPP -)
```

California, Arizona, Texas, Nevada as well as Washington

:CLOSELY-RELATED (= :CLR)

```
    (j1 / join
        :subject HE
  :object (b1 /  board)
        :clr (e1 / director
                 :anchor as))
```

He joined the board as director.

:COMPLEMENT (= :COMPL)

```
    (s1 / want
        :subject (b1 / boy)
        :compl (e1 / eat
                   :patient (b1 / hamburger)))
```

The boy wants to eat a hamburger.

```
        (s1 / say
            :subject (b1 / boy)
            :compl (e1 / eat
                       :patient (b1 / bug)))
```

The boy said that bugs were eaten.

```
       (s1 / wait
           :subject (g1 / girl)
           :compl (n1 / go
                      :agent (b1 / boy))))
```

The girl waited for the boy to go.

```
:CONDITION

        (c1 / cancel
            :PATIENT (t1 / trip)
            :CONDITION (w1 / weather
                           :pred (b1 / bad)))
```

The trip will be canceled if the weather is bad.

```
:CONJ

        (c1 / apple
            / orange
            :CONJ and)
```

apples and oranges

```
        (c1 / (c2 / California)
            / (a1 / Arizona)
            / (a1 / Texas)
            / (a1 / Nevada)
            / (a1 / Washington)
            :CONJ 'as well as')
```

California, Arizona, Texas, Nevada, as well as Washington

```
:COORDPUNC

        (c1 / (c2 / California)
            / (a1 / Arizona)
            / (a1 / Texas)
            / (a1 / Nevada)
            / (a1 / Washington)
            :CONJ ''as well as''
            :COORDPUNC ,)
```

California, Arizona, Texas, Nevada, as well as Washington

```
        (c1 / (c2 / California)
            / (a1 / Arizona)
            / (a1 / Texas)
            / (a1 / Nevada)
            / (a1 / Washington)
            :CONJ ``as well as''
            :BCPP -
            :COORDPUNC or)

    California or Arizona or Texas or Nevada as well as Washington

--------------------------------------------------------------------------------
:DATIVE (= :dtv :ind-obj :indirect-object :bnf :benefactive)

        (g1 / give
            :subject (j1 / John)
            :object (b1 / book)
            :dative (m1 / Mary))

    John gave the book to Mary. John gave Mary the book.

--------------------------------------------------------------------------------
:DESTINATION

        (g1 / go
            :agent (b1 / boy)
            :destination (j1 / japan))

    The boy went to Japan.

        (t1 / tell
            :patient (s1 / story)
            :destination (b1 / boy))

    The story was told to the boy.

--------------------------------------------------------------------------------
:DETERMINER

    Values: indefinite, definite, none, demonstrative_this, demonstrative_that,
            interrogative, ``a,'' ``an,'' ``each,'' etc.

        (t1 / teacher
            :DETERMINER indefinite)

    a teacher
    some teachers
    teachers

        (t1 / apple
            :DETERMINER indefinite)

    an apple
    some apples
```

```
    apples

        (t1 / teacher
            :DETERMINER definite)

    the teacher

        (t1 / teacher
            :DETERMINER demonstrative_this)

    this teacher
    these teachers

        (t1 / teacher
            :DETERMINER demonstrative_that)

    that teacher
    those teachers

        (t1 / teacher
            :DETERMINER interrogative)

    which teacher
    what teacher

--------------------------------------------------------------------------------
:DESTINATION

        (i1 / travel
            :AGENT (l1 / visitor)
            :DESTINATION (f1 / beach))

    Visitors travel to the beach.

--------------------------------------------------------------------------------
:DOMAIN

        (b1 / blue
            :domain (c1 / car))

    The car is blue.

        (l1 / lawyer
            :domain (m1 / man))

    The man is a lawyer.

        (p1 / |possible>workable|
            :domain (e1 / eat
                        :patient (w1 / worm)))

    Worms can be eaten.

--------------------------------------------------------------------------------
:DOMAIN-OF
```

```
        (c1 / car
            :domain-of (b1 / blue))

    the blue car
    the car that is blue

        (m1 / man
            :domain-of (l1 / lawyer))

    the man who is a lawyer

        (l1 / lawyer
     :DOMAIN-OF (a1 / age
                    :RANGE (y1 / year
                                        :QUANT 34)))

    a 34 year old lawyer

-------------------------------------------------------------------------------
:FILLER

       (a1 :template (f1 / flight
                         :postmod (c1 / l1
                            :anchor from))
           :filler (l1 / Los Angeles))

    flights from Los Angeles

-------------------------------------------------------------------------------
:GAP

        (c1 / (p1 / patent
           :premod (u1 / |U.S. government|))
     / (c2 / copyright
           :premod (u2 / |U.S. government|
                     :GAP +))
             :conj AND)

    United States patents and copyrights

        (W9 / |desire,want|
     :AGENT (J9 / |boy<male|)
     :PATIENT (R9 / |read>reread|
                  :PATIENT (B9 / |book<publication|)
                  :AGENT (G1 / J9
                          :GAP +)))

    The boy wants to read the book.

-------------------------------------------------------------------------------
:GENERICALLY-POSSESSES (= :GP)

        (b1 / boy
            :GENERICALLY-POSSESSES (n1 / nose
```

```
                                        :agent-of (b1 / bleed)))

    the boy whose nose was bleeding

-------------------------------------------------------------------------------
:GENERICALLY-POSSESSED-BY (= :GPI, :GENERAL-POSSESSION-INVERSE)

        (h1 / handle
            :gpi (d1 / door))

    the handle of the door

        (o1 / officer
            :gpi (n1 / navy))

    naval officials

        (h1 / hair
            :gpi (s1 / is_prounoun))

    my hair

-------------------------------------------------------------------------------
:INSTANCE

        (d1 / dog)

    a dog
    the dog
    some dogs
    dogs

        (d1 / (p1 / pen
                :premod (i1 / ink))
                :premod (n1 / new))

    a new ink pen
    the new ink pens

-------------------------------------------------------------------------------
:LOGICAL-SUBJECT (= :LOG-SBJ)
:LOGICAL-OBJECT (= :LOG-OBJ)
:LOGICAL-DATIVE (= :LOG-DAT)

        (b1 / give
            :logical-subject (m2 / man)
            :logical-object (t1 / ring))
            :logical-dative (t1 / woman))

    The man gave a ring to the woman.
    The woman was given a ring by a man.
    The ring was given to the woman by a man.

-------------------------------------------------------------------------------
:LOGICAL-DATIVE-OF (= :LOG-DAT-OF)
```

```
        (t1 / woman
            :log-dative-of (b1 / give
                        :log-sbj (m2 / man)
                        :logical-dative (t1 / ring))

    the woman that the man gave the ring to

-------------------------------------------------------------------------------
:LOGICAL-OBJECT-OF (= :LOG-OBJ-OF)

        (t1 / ring
            :log-obj-of (b1 / give
                            :log-sbj (m2 / man)
                 :logical-dative (t1 / woman))

    the ring that the man gave to the woman

-------------------------------------------------------------------------------
:LOGICAL-SUBJECT-OF (= :LOG-SBJ-OF)

        (m2 / man
            :log-sbj-of (b1 / give
                        :log-obj (t1 / ring)
                        :logical-dative (t1 / woman)))

    the man that gave a ring to the woman

-------------------------------------------------------------------------------
:MANNER

        (b1 / build
            :patient (t1 / tunnel)
            :manner (b2 / excavate
                         :patient (m1 / mine)))

    The tunnel was built by excavating the mine.

-------------------------------------------------------------------------------
:MEANS (= :INSTRUMENT)

        (r1 / reach
            :patient (h1 / house)
            :means (c1 / car))

    The house is reached by car.

-------------------------------------------------------------------------------
:MOD (= :premod)

        (c1 / car
            :mod (s1 / shift
                     :mod (s2 / stick)))

    stick shift car
```

```
--------------------------------------------------------------------------------
:MODAL (Examples of semantic inputs that map to modal verbs in next section.)

        (e1 / eat
          :agent (d1 / dog)
          :patient (b1 / bone)
     :modal (m1 / may)
     :mood indicative)

    Dogs may eat bones.

--------------------------------------------------------------------------------
:MOOD

        (e1 / eat
            :agent (d1 / dog)
            :patient (b1 / bone)
            :mood indicative)

    Dogs eat bones.

        (e1 / eat
            :agent (d1 / dog)
            :patient (b1 / bone)
            :modal (m1 / may)
            :mood indicative)

    Dogs may eat bones.

        (i1 / increase
            :agent (w1 / wedding)
            :postmod (e1 / exchange
                         :anchor (w2 / with)
                         :agent (c1 / couple
                             :premod (m1 / more))
                         :patient (r1 / ring)
          :temporal-locating (y1 / 1988)
                         :mood present-participle)

    Weddings increased, with more couples exchanging rings in 1988.

        (h1 / help
            :agent I
            :compl (w1 / walk
                :agent (b1 / baby)
                :mood infinitive)

    I helped the baby walk.

        (w1 / want
            :agent I
            :compl (b1 / go
                       :mood infinitive-to)
```

```
    I want to go.

        (e1 / eat
            :agent YOU
            :patient (b1 / bug)
            :mood imperative)

    Eat bugs.

        (s1 / want
            :agent (c1 / country)
            :compl (f1 / forgive
                       :patient (d1 / debt)
            :mood past-participle)

    The country wants the debt forgiven.

--------------------------------------------------------------------------------
:NAME

        (s1 / sing
            :agent (p1 / |someone|
                       :name John))

    John sang.

--------------------------------------------------------------------------------
:OBJECT

        (e1 / eat
            :subject (b1 / boy)
            :object (b1 / bug)))

    The boy eats bugs.

--------------------------------------------------------------------------------
:OP (= :OP1, :OP2, :OP3, :OP4)

        (a1 / and
            :op (s1 / sing
                    :agent  (b1 / boy))
            :op (d1 / dance
                    :agent  (g1 / girl)))

    The boys sang, and the girls danced.

--------------------------------------------------------------------------------
:PATIENT (= :PHENOMENON, :SAYING, :GOAL, :CREATED-ENTITY)

        (e1 / eat
            :patient (b1 / bug))

    Bugs were eaten.

        (s1 / say
```

```
            :agent (b1 / boy)
            :patient (e1 / eat
                        :patient (b1 / bug)))
```

The boy said that bugs were eaten.

```
:PATIENT-OF

        (b1 / bug
            :patient-of (e1 / eat
                            :agent (b1 / boy)))
```

Bugs that the boy ate

```
:POLARITY

        (g1 / go
            :agent (b1 / boy)
            :polarity -)
```

The boy did not go.

```
        (n1 / |necessary<inevitable|
            :domain (g1 / go
                        :polarity -
                        :agent (b1 / boy)))
```

The boy must not go.

```
        (n1 / |necessary<inevitable|
            :polarity -
            :domain (g1 / go
                        :agent (b1 / boy)))
```

The boy need not go.

```
:PREDICATE (= :PRED)

        (w1 / seem
            :subject (p1 / she)
            :pred (b1 / blue))
```

She seems blue.

```
        (w1 / |has the quality of being|
            :subject (p1 / man)
            :pred (b1 / lawyer))
```

The man is a lawyer.

```
:POSTMOD
```

```
        (b1 / break
            :postmod (t1 / tradition
                         :anchor (f1 / from)))

    a break from tradition

        (b1 / rise
            :agent (p1 / price)
            :postmod (y1 / year
                  :mod (l1 / last)))

    Prices rose last year.

--------------------------------------------------------------------------------
:PREDET

        (b1 / dog
            :predet (a1 / all))

    all the dogs

        (b1 / dog
            :predet (e1 / every))

    every dog

--------------------------------------------------------------------------------
:PREMOD

        (b1 / bright
            :premod (v1 / very))

    very bright

        (b1 / rise
            :agent (p1 / price)
            :premod (y1 / year
                 :mod (l1 / last)))

    Last year prices rose.

--------------------------------------------------------------------------------
:PRO

        (w1 / win
            :agent (p1 / |someone|
                       :pro i))

    I won.

        (w1 / win
            :agent (p1 / team
                       :pro 3p_pronoun))
```

```
    They won.

    Note: Inputs can also contain the pronoun directly, as in:

        (h1 / help
            :subject (w1 / WE)
            :compl (q1 / win
                :subject THEY)

    We helped them win.

-------------------------------------------------------------------------------
:PUNC (maps to :rightpunc, :leftpunc, :sandwichpunc)

    VALUES: comma, colon, semi-colon, period, question_mark, exclamation_mark,
            dash, longdash, singlequotes, doublequotes, curlybraces,
            squarebrackets, parentheses, left_single_quote,
            right_single_quote, ``,'', ``.'', ``!'', etc.

        (s1 / say
            :punc period
            :agent (b1 / boy)
            :patient (g1 / (h1 / hello
                               :punc exclamation_mark)
                  :punc doublequotes))

    The boy said, ``hello!''

-------------------------------------------------------------------------------
:PURPOSE

        (s1 / shout
            :agent (b1 / boy)
            :purpose (g1 / get
                         :patient (h1 / help)))

    The boy shouted to get help.

-------------------------------------------------------------------------------
:QUANT

        (t1 / toy
            :quant 5)

    5 toys

        (s1 / school
            :anchor (b1 / beyond)
            :quant (bl / block
                :quant 5))

    5 blocks past the school

-------------------------------------------------------------------------------
:QUESTION (maps to :topic with (:punc question_mark
```

```
                                :subject-position post-aux))

        (e1 / say
            :agent (w1 / researcher)
            :question (b1 / what))

    What did the researcher say?

        (e1 / happen
            :agent (w1 / it)
            :question (b1 / often
                          :premod (h1 / how)))

    How often did it happen?

--------------------------------------------------------------------------------
:QUOTED

        (s1 / say
            :agent (b1 / boy)
            :patient (r1 / rain
                         :quoted +))

    The boy said, ``It rained.''

--------------------------------------------------------------------------------
:RANGE

        (b1 / become
            :domain (b2 / boy)
            :range (f1 / fish))

    The boy became a fish.

--------------------------------------------------------------------------------
:REASON

        (g1 / go
            :agent (b1 / boy)
            :reason (h1 / hurricane))

    The boy went because of the hurricane.

--------------------------------------------------------------------------------
:RECIPIENT

        (g1 / give
            :agent (j1 / John)
            :patient (b1 / book)
            :recipient (m1 / Mary))

    John gave the book to Mary.
    John gave Mary the book.

--------------------------------------------------------------------------------
```

```
:RECIPIENT-OF

        (m1 / Mary
            :recipient-of (g1 / give
                              :patient (b1 / book)))

    Mary, who was given the book

--------------------------------------------------------------------------------
:RESTATEMENT

        (a1 / abdicate
            :agent (p1 / |someone|
                       :name Nicholas
                       :restatement (c1 / czar
                                        :gpi (r1 / russia))))

    Nicholas, Czar of Russia, abdicated.

--------------------------------------------------------------------------------
:ROLE-OF-AGENT (= :ROLE)

        (s1 / speak
            :agent (m1 / man)
            :role-of-agent (p1 / president))

    The man spoke as president. (= :ROLE)

--------------------------------------------------------------------------------
:ROLE-OF-PATIENT

        (u1 / use
            :patient (b1 / barn)
            :role-of-patient (g1 / garage))

    The barn was used as a garage.

--------------------------------------------------------------------------------
:SANS

        (c1 / car
            :sans (w1 / wheel))

    a car without wheels

--------------------------------------------------------------------------------
:SOURCE

        (c1 / come
            :agent (b1 / boy)
            :source (b2 / beyond
                        :anchor (g1 / galaxy)))

    The boy came from beyond the galaxy.
```

```
---------------------------------------------------------------------------
:SPATIAL-LOCATING

        (r1 / rain
            :polarity -
            :spatial-locating (m1 / mars))

    It does not rain on Mars.

---------------------------------------------------------------------------
:SUBJECT

       (g1 / go
           :SUBJECT (b1 / boy))

    The boy went.

       (g1 / eat
           :voice passive
           :SUBJECT (b1 / bug))

    Bugs were eaten.

---------------------------------------------------------------------------
:SUBJECT-POSITION (VALUES: default, post-aux, post-vp)

        (e1 / say
            :subject (w1 / researcher)
            :subject-position post-aux
            :punc question_mark
            :topic (b1 / what))

    What did the researcher say?

        (e1 / say
            :agent (w1 / researcher)
            :subject-position post-vp
            :topic (b1 / be
                       :subject (f1 / fiber)
                       :pred (r1 / resilient)))

    "The fiber is resilient," said the researcher.

        (e1 / say
            :agent (w1 / researcher)
            :subject-position default
            :complement (b1 / be
                            :subject (f1 / fiber)
                            :pred (r1 / resilient)))

    The researcher said that the fiber is resilient.

---------------------------------------------------------------------------
:TEMPLATE
```

```
       (a1 :template (f1 / flight
                         :postmod (c1 / l1
                        :anchor from))
           :filler (l1 / Los Angeles))

    flights from Los Angeles

-------------------------------------------------------------------------------
:TEMPORAL-LOCATING

        (s1 / snow
            :temporal-locating (n1 / november))

    It snowed in November.

-------------------------------------------------------------------------------
:THEME

        (e1 / eat
            :patient (w1 / worm)
            :topic (b1 / boy))

    As for the boy, worms were eaten.

-------------------------------------------------------------------------------
:TOPIC

        (e1 / say
            :agent (w1 / researcher)
            :topic (b1 / be
                       :subject (f1 / fiber)
                       :pred (r1 / resilient)))

    ``The fiber is resilient,'' the researcher said.

        (e1 / say
            :subject (w1 / researcher)
            :subject-position post-aux
            :punc question_mark
            :topic (b1 / what))

    What did the researcher say?

-------------------------------------------------------------------------------
:WITHINMOD

        (e1 / eat
            :withinmod (o1 / often)
            :patient (w1 / worm)
            :agent (b1 / boy))

    The boy often eats worms.

        (e1 / eat
            :withinmod (s1 / still)
```

```
          :aspect continuous
          :patient (w1 / worm)
          :agent (b1 / boy))
```

The boy is still eating worms.

C.2 Possible, Obligatory, Likely, Permitted, etc.

The concepts |possible>workable|, |possible<latent|, |permitted|, |obligatory<necessary|, and |necessary>inevitable| are treated specially by the English generator to sometimes introduce modal verbs.

```
      (h1 / |possible>workable|
           :domain (a1 / |eat,take in|
                       :agent she
                       :patient (C1 / |poulet|)))
```

She can eat chicken.

Note: If ''can'' is used in the sense of ''might,'' please use the concept |possible<latent| as shown in the following example.

```
       (h1 / |possible<latent|
            :domain (a1 / |eat,take in|
                        :agent she
                        :patient (C1 / |poulet|)))
```

She might eat chicken.

```
      (h1 / |obligatory<necessary|
          :domain (a1 / |eat,take in|
                      :agent she
                      :patient (C1 / |poulet|)))
```

She must eat chicken.

```
      (h1 / |necessary>inevitable|
          :domain (a1 / |eat,take in|
                      :agent she
                      :patient (C1 / |poulet|)))
```

She needs to eat chicken.

```
      (h1 / |permitted|
          :domain (a1 / |eat,take in|
                      :agent she
                      :patient (C1 / |poulet|)))
```

She may eat chicken.

```
      (h1 / |likely>apt|
          :domain (a1 / |eat,take in|
                      :agent she
                      :patient (C1 / |poulet|)))
```

```
    She is likely to eat chicken.
```

Note: Be careful when mixing modals and negation. Consider the placement of polarity in the following two cases.

```
        (h1 / |possible>workable|
            :polarity -
            :domain (a1 / |eat,take in|
                        :agent she
                        :patient (C1 / |poulet|)))

    She can not eat chicken.

        (h1 / |obligatory<necessary|
            :domain (a1 / |eat,take in|
                         :polarity -
                         :agent she
                         :patient (C1 / |poulet|)))

    She must not eat chicken.


        (h1 / |exist,be|
            :polarity -
            :domain (i1 / interlingua
                         :mod (p1 / perfect)))

    There is no perfect interlingua.
```