# A Uniform Architecture for Parsing and Generation

Stuart M. SHIEBER
Artificial Intelligence Center
SRI International
Menlo Park, California, USA*

## Abstract

The use of a single grammar for both parsing and generation is an idea with a certain elegance, the desirability of which several researchers have noted. In this paper, we discuss a more radical possibility: not only can a single grammar be used by different processes engaged in various "directions" of processing, but one and the same language-processing architecture can be used for processing the grammar in the various modes. In particular, parsing and generation can be viewed as two processes engaged in by a single parameterized theorem prover for the logical interpretation of the formalism. We discuss our current implementation of such an architecture, which is parameterized in such a way that it can be used for either purpose with grammars written in the PATR formalism. Furthermore, the architecture allows fine tuning to reflect different processing strategies, including parsing models intended to mimic psycholinguistic phenomena. This tuning allows the parsing system to operate within the same realm of efficiency as previous architectures for parsing alone, but with much greater flexibility for engaging in other processing regimes.

## 1 Introduction

The use of a single grammar for both parsing and generation is an idea with a certain elegance, the desirability of which several researchers have noted. Of course, judging the correctness of such a system requires a characterization of the meaning of grammars that is independent of their use by a particular processing mechanism—that is, the formalism in which the grammars are expressed must have an abstract semantics. As a paradigm example of such a formalism, we might take any of the various logic- or unification-based grammar formalisms.

As described by Pereira and Warren [1983], the parsing of strings according to the specifications of a grammar with an independent logical semantics can be thought of as the constructive proving of the string's grammaticality: parsing can be viewed as logical deduction. But, given a deductive framework that can represent the semantics of the formalism abstractly enough to be independent of processing, the generation of strings matching some criteria can equally well be thought of as a deductive process, namely, a process of constructive proof of the existence of a string that matches the criteria. The difference rests in which information is given as premises and what the goal is to be proved. This observation opens up the following possibility: not only can a single grammar be used by different processes engaged in various "directions" of processing, but one and the same language-processing architecture can be employed for processing the grammar in the various modes. In particular, parsing and generation can be viewed as two processes engaged in by a single parameterized theorem prover for the logical interpretation of the formalism.

We will discuss our current implementation of such an architecture, which is parameterized in such a way that it can be used either for parsing or generation with respect to grammars written in a particular grammar formalism which has a logical semantics, the PATR formalism. Furthermore, the architecture allows fine tuning to reflect different processing strategies, including parsing models intended to mimic psycholinguistic phenomena. This tuning allows the parsing system to operate within the same realm of efficiency as previous architectures for parsing alone, but with much greater flexibility for engaging in other processing regimes.

*This research was sponsored by the Nippon Telegraph and Telephone Corporation under a contract with SRI International.

## 2 Language Processing as Deduction

Viewed intuitively, natural-language-utterance generation is a nondeterministic top-down process of building a phrase that conforms to certain given criteria, e.g., that the phrase be a sentence and that it convey a particular meaning. Parsing, on the other hand, is usually thought of as proceeding bottom-up in an effort to determine what properties hold of a given expression. As we have mentioned, however, both of these processes can be seen as variants of a single method for extracting certain *goal* theorems from the deductive closure of some given *premises* under the rules or constraints of the grammar. The various processes differ as to what the premises are and which goal theorems are of interest. In generation, for instance, the premises are the lexical items of the language and goal theorems are of the form "expression $\alpha$ is a sentence with meaning $M$" for some given $M$. In parsing, the premises are the words $\alpha$ of the sentence to be parsed and goal theorems are of the form "expression $\alpha$ is a sentence (with properties $P$)". In this case, $\alpha$ is given a priori.

This deductive view of language processing clearly presupposes an axiomatic approach to language description. Fortunately, most current linguistic theory approaches the problem of linguistic description axiomatically, and many current formalisms in use in natural-language processing, especially the logic grammar and unification-based formalisms follow this approach as well. Consequently, the results presented here will, for the most part, be applicable to any of these formalisms. We will, however, describe the system schematically—without relying on any of the particular formalisms, but using notation that schematizes an augmented context-free formalism like definite-clause grammars or PATR. We merely assume that grammars classify phrases under a possibly infinite set of structured objects, as is common in the unification-based formalisms. These structures—terms in definite-clause grammars, directed graphs in PATR, and so forth—will be referred to generically as nonterminals, since they play the role in the augmented context-free formalisms that the atomic nonterminal symbols fulfill in standard context-free grammars. We will assume that the notion of a *unifier* of such objects and *most general unifier* (*mgu*) are well defined; the symbol $\theta$ will be used for unifiers.

Following Pereira and Warren, the lemmas we will be proving from a grammar and a set of premises will include the same kind of conditional information encoded by the *items* of Earley's parsing algorithm. In Earley's algorithm, the existence of an item (or *dotted rule*) of the form

$$[N \rightarrow V_1 \cdots V_{m-1} \bullet V_m \cdots V_n, i]$$

in state set $j \geq i$ makes a claim that, for some string position $k \geq j$, the substring between $i$ and $k$ can be classified as an $N$ if the substring between $j$ and $k$ can be decomposed into a sequence of strings classified, respectively, under $V_m, \ldots, V_n$. We will use a notation reminiscent of Pereira and Warren's[1] to emphasize the conditional nature of the claim and its independence from $V_1, \ldots, V_{m-1}$, namely,

$$[i, N \leftarrow V_m \cdots V_n, j]$$

### 2.1 Terminology

We digress here to introduce some terminology. If $n = 0$, then we will leave off the arrow; $[i, N, j]$ then expresses the fact that a constituent admitted as a nonterminal $N$ occurs between positions $i$ and $j$. Such items will be referred to as *nonconditional* items; if $n > 0$, the item will be considered *conditional*. In the grammars we are interested in, rules will include either all nonterminals on the right-hand side or all terminals. We can think of the former as grammar rules proper, the

[1]Later, in the sections containing examples of the architecture's operation, we will reintroduce $V_1, \ldots, V_{m-1}$ and the dot marker to aid readability.

latter as lexical entries. Nonconditional items formed by immediate inference from a lexical entry will be called *lexical* items. For instance, if there is a grammar rule $NP \rightarrow sonny$, then the item $[0, NP, 1]$ is a lexical item. A *prediction* item (or, simply, a prediction) is an item with identical start and end positions.

### 2.2 Rules of Inference

The two basic deduction steps or rules of inference we will use are—following Earley's terminology—*prediction* and *completion*.[2]

The inference rule of prediction is as follows:

$$\frac{[i, A \leftarrow B C_1 \cdots C_m, j] \qquad B' \rightarrow D_1 \cdots D_n \qquad \theta = mgu(B, B')}{[j, B'\theta \leftarrow D_1\theta \cdots D_n\theta, j]}$$

This rule corresponds to the logically valid inference consisting of instantiating a rule of the grammar as a conditional statement.[3]

The inference rule of completion is as follows:

$$\frac{[i, A \leftarrow B C_1 \cdots C_m, j] \qquad [j, B', k] \qquad \theta = mgu(B, B')}{[i, A\theta \leftarrow C_1\theta \cdots C_m\theta, k]}$$

This rule corresponds to the logically valid inference consisting of linear resolution of the conditional expression with respect to the nonconditional (unit) lemma.

## 3 Parameterizing a Theorem-Proving Architecture

This characterization of parsing as deduction should be familiar from the work of Pereira and Warren. As they have demonstrated, such a view of parsing is applicable beyond the context-free grammars by regarding the variables in the inference rules as logical variables and using unification of $B$ and $B'$ to solve for the most general unifier. Thus, this approach is applicable to most, if not all, of the logic grammar or unification-based formalisms.

In particular, Pereira and Warren construct a parsing algorithm using a deduction strategy which mimics Earley's algorithm. We would like to generalize the approach, so that the deduction strategy (or at least portions of it) are parameters of the deduction system. The parameterization should have sufficient generality that parsers and generators with various control strategies, including Pereira and Warren's Earley deduction parser, are instances of the general architecture.

We start the development of such an architecture by considering the unrestricted use of these two basic inference rules to form the deductive closure of the premises and the goals. The exhaustive use of prediction and completion as basic inference rules does provide a complete algorithm for proving lemmas of the sort described. However, several problems immediately present themselves.

First, proofs using these inference rules can be redundant. Various combinations of proof steps will lead to the same lemmas, and combinatorial havoc may result. The traditional solution to this problem is to store lemmas in a table, i.e., the well-formed-substring table or chart in tabular parsing algorithms. In extending tabular parsing to non-context-free formalisms, the use of subsumption rather than identity in testing for redundancy of lemmas becomes necessary, and has been described elsewhere [Pereira and Shieber, 1987].

Second, deduction is a nondeterministic process and the order of searching the various paths in the proof space is critical and differs among processing tasks. We therefore parameterize the theorem-proving process by a priority function that assigns to each lemma a priority. Lemmas are then added to the table in order of their priority. As they are added, further lemmas that are consequences of the new lemma and existing ones in the table may be deduced. These are themselves assigned priorities, and so forth. The technique chosen for implementing this facet of the process is the use of an agenda structured as a priority queue to store the lemmas that have not yet been added to the table.

Finally, depending on the kind of language processing we are interested in, the premises of the problem and the types of goal lemmas we are searching for will be quite different. Therefore, we parameterize the theorem prover by an initial set of axioms to be added to the agenda and by a predicate on lemmas that determines which are to be regarded as satisfying the goal conditions on lemmas.

The structure of the architecture, then, is as follows. The processor is an agenda-based tabular theorem prover over lemmas of the sort defined above. It is parameterized by

- The initial conditions,
- A priority function on lemmas, and
- A predicate expressing the concept of a successful proof.

By varying these parameters, the processor can be used to implement language parsers and generators embodying a wide variety of control strategies.

## 4 Instances of the Architecture

We now define some examples of the use of the architecture to process with grammars.

### 4.1 Parser Instances

Consider a processor to parse a given string built by using this architecture under the following parameterization:

- The *initialization* of the agenda includes axioms for each word in the string (e.g., $[0, sonny, 1]$ and $[1, left, 2]$ for the sentence 'Sonny left') and an initial prediction for each rule whose left-hand side matches the start symbol of the grammar (e.g., $[0, S \leftarrow NP\ VP, 0]$).[4]
- The *priority function* orders lemmas inversely by their end position, and for lemmas with the same end position, in accordance with their addition to the agenda in a first-in-first-out manner.
- The *success criterion* is that the lemma be nonconditional, that its start and end positions be the first and last positions in the string, respectively, and that the nonterminal be the start nonterminal.[5]

Under this parameterization, the architecture mimics Earley's algorithm parsing the sentence in question, and considers successful those lemmas that represent proofs of the string's grammaticality with respect to the grammar.[6]

Alternatively, by changing the priority function, we can engender different parsing behavior. For instance, if we just order lemmas in a last-in-first-out manner (treating the agenda as a stack) we have a "greedy" parsing algorithm, which pursues parsing possibilities depth-first and backtracks when dead-ends occur.

An interesting possibility involves ordering lemmas as follows:

1. Highest priority are prediction items, then lexical items, then other conditional items, then other nonconditional items.
2. If (1) does not order items, items ending farther to the right have higher priority.
3. If (1) and (2) do not order items, items constructed from the instantiation of longer rules have higher priority.

This complex ordering implements a quite simple parsing strategy. The first condition guarantees that no nonconditional items will be added until conditional items have been computed. Thus, items corresponding to the *closure* (in the sense of LR parsing) of the nonconditional items are always added to the table. Unlike LR parsing,

[2]Pereira and Warren use the terms *instantiation* and *reduction* for their analogs to these rules.

[3]As noted previously [Shieber, 1985], this rule of inference can lead to arbitrary numbers of consequents through repeated application when used with a grammar formalism with an infinite [structured] nonterminal domain. The solution proposed in that paper is to restrict the information passed from the predicting to the predicted item, corresponding to the rule

$$\frac{[i, A \leftarrow B C_1 \cdots C_m, j] \qquad B' \rightarrow D_1 \cdots D_n \qquad \theta = mgu(B\!\upharpoonright\!\Phi, B')}{[j, B'\theta \leftarrow D_1\theta \cdots D_n\theta, j]}$$

where $B\!\upharpoonright\!\Phi$ is a nonterminal with a bounded subset of the information of $B$. This inference rule is the one actually used in the implemented system. The reader is directed to the earlier paper for further discussion.

[4]For formalisms with complex structured nonterminals, the start "symbol" need only be unifiable with the left-hand-side nonterminal. That is, if $S$ is the start nonterminal and $S' \rightarrow C_1 \cdots C_n$ is a rule and $\theta = mgu(S, S')$, then $[0, S'\theta \leftarrow C_1\theta \cdots C_n\theta, 0]$ is an initial prediction.

[5]Again, for formalisms with complex structured nonterminals, the start symbol need only subsume the item's nonterminal.

[6]Assuming that the prediction inference rule uses the restriction mechanism, the architecture actually mimics the variant of Earley's algorithm previously described in [Shieber, 1985].

however, the closure here is computed at run time rather than being precompiled. The last two conditions correspond to disambiguation of shift/reduce and reduce/reduce conflicts in LR parsing respectively. The former requires that shifts be preferred to reductions, the latter that longer reductions receive preference.

In sum, this ordering strategy implements a sentence-disambiguation parsing method that has previously been argued [Shieber, 1983] to model certain psycholinguistic phenomena—for instance, right association and minimal attachment [Frazier and Fodor, 1978]. However, unlike the earlier characterization in terms of LR disambiguation, this mechanism can be used for arbitrary logic or unification-based grammars, not just context-free grammars. Furthermore, the architecture allows for fine tuning of the disambiguation strategy beyond that described in earlier work. Finally, the strategy is complete, allowing "backtracking" if earlier proof paths lead to a dead end.[7]

## 4.2 A Parsing Example

As a demonstration of the architecture used as a parser, we consider the Earley and backtracking-LR instances in parsing the ambiguous sentence:

Castillo said Sonny was shot yesterday.

Since the operation of the architecture as a parser is quite similar to that of previous parsers for unification-based formalisms, we will only highlight a few crucial steps in the process.

The Earley parser assigns higher priority to items ending earlier in the sentence. The highest-priority initialization items are added first.[8]

| | |
|---|---|
| $[0, S \rightarrow \bullet NP\ VP, 0]$ | '' |
| $[0, NP \rightarrow castillo \bullet, 1]$ | 'Castillo' |

By Completion, the item

| | |
|---|---|
| $[0, S \rightarrow NP \bullet VP, 1]$ | 'Castillo' |

is generated, which in turn predicts

| | |
|---|---|
| $[1, VP \rightarrow \bullet VP\ X, 1]$ | '' |
| $[1, VP \rightarrow \bullet V, 1]$ | '' |
| $[1, VP \rightarrow \bullet VP\ AdvP, 1]$ | '' |

The highest-priority item remaining on the agenda is the initial item

| | |
|---|---|
| $[1, V \rightarrow said \bullet, 2]$ | 'said' |

Processing progresses in this manner, performing all operations at a string position before moving on to the next position until the final position is reached, at which point the final initial item corresponding to the word 'yesterday' is added. The following flurry of items is generated by completion.[9]

| | | |
|---|---|---|
| | ... | |
| | $[5, AdvP \rightarrow yesterday \bullet, 6]$ | 'yesterday' |
| (2) | $[1, VP \rightarrow VP\ AdvP \bullet, 6]$ | 'said Sonny was shot yesterday' |
| (3) | $[3, VP \rightarrow VP\ AdvP \bullet, 6]$ | 'was shot yesterday' |
| | $[4, VP \rightarrow VP\ AdvP \bullet, 6]$ | 'shot yesterday' |
| | $[1, VP \rightarrow VP \bullet AdvP, 6]$ | 'said Sonny was shot yesterday' |
| (4) | $[0, S \rightarrow NP\ VP \bullet, 6]$ | 'Castillo said Sonny was shot yesterday' |
| | $[3, VP \rightarrow VP \bullet AdvP, 6]$ | 'was shot yesterday' |
| (5) | $[2, S \rightarrow NP\ VP \bullet, 6]$ | 'Sonny was shot yesterday' |
| | $[4, VP \rightarrow VP \bullet AdvP, 6]$ | 'shot yesterday' |
| (6) | $[1, VP \rightarrow VP\ S \bullet, 6]$ | 'said Sonny was shot yesterday' |
| | $[1, VP \rightarrow VP \bullet AdvP, 6]$ | 'said Sonny was shot yesterday' |
| (7) | $[0, S \rightarrow NP\ VP \bullet, 6]$ | 'Castillo said Sonny was shot yesterday' |

Note that the first full parse found (4) is derived from the high attachment of the word 'yesterday' (2) (which is composed from (1) directly), the second (7) from the low attachment (6) (derived from (5), which is derived in turn from (3)).

By comparison, the shift-reduce parser generates exactly the same items as the Earley parser, but in a different order. The crucial ordering difference occurs in the following generated items:

| | | |
|---|---|---|
| | ... | |
| (1) | $[5, AdvP \rightarrow yesterday \bullet, 6]$ | 'yesterday' |
| | ... | |
| (3) | $[3, VP \rightarrow VP\ AdvP \bullet, 6]$ | 'was shot yesterday' |
| | $[3, VP \rightarrow VP \bullet AdvP, 6]$ | 'was shot yesterday' |
| (5) | $[2, S \rightarrow NP\ VP \bullet, 6]$ | 'Sonny was shot yesterday' |
| (6) | $[1, VP \rightarrow VP\ S \bullet, 6]$ | 'said Sonny was shot yesterday' |
| | $[1, VP \rightarrow VP \bullet AdvP, 6]$ | 'said Sonny was shot yesterday' |
| (7) | $[0, S \rightarrow NP\ VP \bullet, 6]$ | 'Castillo said Sonny was shot yesterday' |
| (8) | $[2, S \rightarrow NP\ VP \bullet, 5]$ | 'Sonny was shot' |
| | $[1, VP \rightarrow VP\ S \bullet, 5]$ | 'said Sonny was shot' |
| | $[1, VP \rightarrow VP \bullet AdvP, 5]$ | 'said Sonny was shot' |
| (2) | $[1, VP \rightarrow VP\ AdvP \bullet, 6]$ | 'said Sonny was shot yesterday' |
| | $[1, VP \rightarrow VP \bullet AdvP, 6]$ | 'said Sonny was shot yesterday' |
| (4) | $[0, S \rightarrow NP\ VP \bullet, 6]$ | 'Castillo said Sonny was shot yesterday' |
| | ... | |

Note that the reading of the sentence (7) with the low attachment of the adverb—the so-called "right association" reading—is generated before the reading with the higher attachment (4), in accordance with certain psycholinguistic results [Frazier and Fodor, 1978]. This is because item (3) has higher priority than item (8), since (3) corresponds to the shifting of the word 'yesterday' and (8) to the reduction of an *NP* and *VP* to *S*. The second clause of the priority definition orders such shifts before reductions. In summary, this instance of the architecture develops parses in right-association/minimal-attachment preference order.

## 4.3 Generator Instances

As a final example of the use of this architecture, we consider using it for generation by changing the initialization condition as follows:

- The *initialization* of the agenda includes axioms for each word in the lexicon at each position (e.g., $[0, sonny, 1]$ and $[0, left, 1]$ and $[1, left, 2]$, and so on) and an initial prediction for each rule whose left-hand side is the start symbol of the grammar (e.g., $[0, S \leftarrow NP\ VP, 0]$). In the case of a grammar formalism with more complex information structures as nonterminals, e.g., definite-clause grammars, the "start symbol" might include information about, say, the meaning of the sentence to be generated. We will refer to this as the *goal meaning*.
- The *success criterion* is that the nonterminal be subsumed by the start nonterminal (and therefore have the appropriate meaning).

Under this parameterization, the architecture serves as a generator for the grammar, generating sentences with the intended meaning. By changing the priority function, the order in which possibilities are pursued in generation can be controlled, thereby modeling depth-first strategies, breadth-first strategies, and so forth.

Of course, as described, this approach to generation is sorely inadequate for several reasons. First, it requires that we initially insert the entire lexicon into the agenda at arbitrary numbers of string positions. Not only is it infeasible to insert the lexicon so many times (indeed, even once is too much) but it also leads to massive redundancy in generation. The same phrase may be generated starting at many different positions. For parsing, keeping track of which positions phrases occur at is advantageous; for generation, once a phrase is generated, we want to be able to use it in a variety of places.

A simple solution to this problem is to ignore the string positions in the generation process. This can be done by positioning all lemmas at a single position. Thus we need insert the lexicon only once, each word being inserted at the single position, e.g., $[0, sonny, 0]$.

Although this simplifies the set of initial items, by eliminating indexing based on string position we remove the feature of tabular parsing

[7]Modeling of an incomplete version of the shift-reduce technique is also possible. The simplest method, however, involves eliminating the chart completing, and mimicking closure, shift, and reduction operations as operations on LR states (sets of items) directly. Though this method is not a straightforward instantiation of the architecture of Section 3 (since the chart is replaced by separate state sets), we have implemented a parser using much of the same technology described here and have successfully modeled the garden path phenomena that rely on the incompleteness of the shift-reduce technique.

[8]The format used in displaying these items reverts to one similar to Earley's algorithm, with a dot marking the position in the rule covered by the string generated so far, so as to describe more clearly the portion of each grammar rule used. In addition, the string actually parsed or generated is given in single quotes after the item for convenience.

[9]The four instances of 'said Sonny was shot yesterday' arise because of lexical ambiguity in the verb 'said' and adverbial-attachment ambiguity. Only the finite version of 'said' is used in forming the final sentence.

methods such as Earley's algorithm that makes parsing reasonably efficient. The generation behavior exhibited is therefore not goal-directed; once the lexicon is inserted many phrases might be built that could not contribute in any way to a sentence with the appropriate meaning. In order to direct the behavior of the generator towards a goal meaning, we can modify the priority function so that it is partial; not every item will be assigned a priority and those that are not will never be added to the table (or agenda) at all. The filter we have been using assigns priorities only to items that might contribute semantically to the goal meaning. In particular, *the meaning associated with the item must subsume some portion of the goal meaning.*[10] This technique, a sort of indexing on meaning, replaces the indexing on string position that is more appropriate for parsing than generation.

As a rule, filtering the items by making the priority function partial can lead to incompleteness of the parsing or generation process.[11] However, the subsumption filter described here for use in generation does not yield incompleteness of the generation algorithm under one assumption about the grammar, which we might call *semantic monotonicity.* A grammar is semantically monotonic if, for every phrase admitted by the grammar, the semantic structure of each immediate subphrase subsumes some portion of the semantic structure of the entire phrase. Under this condition, items which do not subsume part of the goal meaning can be safely ignored, since any phrase built from them will also not subsume part of the goal meaning and thus will fail to satisfy the success criterion. Thus the question of completeness of the algorithm depends on an easily detectable property of the grammar. Semantic monotonicity is, by intention, a property of the particular grammar we have been using.

## 4.4 A Generation Example

As an example of the generation process, we consider the generation of a sentence with a goal logical form

*passionately(love(sonny,kait))*

The example was run using a toy grammar that placed subcategorization information in the lexicon, as in the style of analysis of head-driven phrase-structure grammar (HPSG). The grammar ignored tense and aspect information, so that, for instance, auxiliary verbs merely identified their own semantics with that of their postverbal complement.[12]

The initial items included the following:

| | | |
|---|---|---|
| (1) | $[0, NP \to sonny\bullet, 0]$ | 'Sonny' |
| (2) | $[0, NP \to kait\bullet, 0]$ | 'Kait' |
| | $[0, V \to to\bullet, 0]$ | 'to' |
| | $[0, V \to was\bullet, 0]$ | 'was' |
| | $[0, V \to were\bullet, 0]$ | 'were' |
| | ... | |
| | $[0, V \to loves\bullet, 0]$ | 'loves' |
| | $[0, V \to love\bullet, 0]$ | 'love' |
| | $[0, V \to loved\bullet, 0]$ | 'loved' |
| | $[0, AdvP \to passionately\bullet, 0]$ | 'passionately' |
| (3) | $[0, S \to \bullet NP\ VP, 0]$ | '' |

Note that auxiliary verbs were included, as the semantic structure of an auxiliary is merely a variable (coiindexed with the semantic structure of its postverbal complement), which subsumes some part (in fact, every part) of the goal logical form.[13] Similarly, the noun phrases 'Sonny' and 'Kait' (with semantics *sonny* and *kait*, respectively) are added, as these logical forms each subsume the respective innermost arguments of the goal logical form. Several forms of the verb 'love' are considered, again because the semantics in this grammar makes no tense/aspect distinctions. But no other proper nouns or verbs are considered (although the lexicon that was used contained them) as they do not pass the semantic filter.

The noun phrase 'Sonny' can be used as the subject of the sentence by combining items (1) and (3) yielding

| | | |
|---|---|---|
| (4) | $[0, S \to NP \bullet VP, 0]$ | 'Sonny' |

(The corresponding item with the subject 'Kait' will be generated later.) Prediction yields the following chain of items.

| | |
|---|---|
| $[0, VP \to \bullet VP\ AdvP, 0]$ | '' |
| $[0, VP \to \bullet V, 0]$ | '' |

The various verbs, including the forms of 'love', can complete this latter item.

| | | |
|---|---|---|
| | $[0, VP \to V\bullet, 0]$ | 'to' |
| | $[0, VP \to V\bullet, 0]$ | 'is' |
| | $[0, VP \to V\bullet, 0]$ | 'was' |
| | $[0, VP \to V\bullet, 0]$ | 'were' |
| (5) | $[0, VP \to V\bullet, 0]$ | 'loves' |
| | $[0, VP \to V\bullet, 0]$ | 'love' |
| | $[0, VP \to V\bullet, 0]$ | 'love' |
| | $[0, VP \to V\bullet, 0]$ | 'loved' |

The passive form of the verb 'loved' might be combined with the adverb.

| | |
|---|---|
| $[0, VP \to VP \bullet AdvP, 0]$ | 'loved' |
| $[0, VP \to VP\ AdvP \bullet, 0]$ | 'loved passionately' |
| ... | |

The latter item might be used in a sentence 'Kait was loved passionately.' This sentence will eventually be generated but will fail the success criterion because its semantics is insufficiently instantiated.

Prediction from item (4) also yields the rule for adding complements to a verb phrase.

| | |
|---|---|
| $[0, VP \to \bullet VP\ X, 0]$ | '' |

Eventually, this item is completed with items (5) and (2).

| | |
|---|---|
| ... | |
| $[0, VP \to VP \bullet NP, 0]$ | 'loves' |
| $[0, VP \to VP\ NP \bullet, 0]$ | 'loves Kait' |

The remaining items generated are

| | |
|---|---|
| $[0, VP \to VP \bullet AdvP, 0]$ | 'loves Kait' |
| $[0, VP \to VP\ AdvP \bullet, 0]$ | 'loves Kait passionately' |
| $[0, S \to NP\ VP \bullet, 0]$ | 'Sonny loves Kait passionately' |

This final item matches the success criterion, and is the only such item. Therefore, the sentence 'Sonny loves Kait passionately' is generated for the logical form *passionately*(*love*(*sonny*, *kait*)).

Looking over the generation process, the set of phrases actively explored by the generator included 'Kate is loved', 'Kate is loved passionately', 'were loved passionately' and similar passive constructions, 'Sonny loves Kait', and various subphrases of these. However, other phrases composed of the same words, such as 'Kait loves Kait', 'Sonny is loved', and so forth, are eliminated by the semantics filter. Thus, the the generation process is, on the whole, quite goal-directed; the subphrases considered in the generation process are "reasonable".

## 5 The Implementation

The architecture described above has been implemented for the PATR grammar formalism in a manner reminiscent of object-oriented programming. Instances of the architecture are built as follows. A general-purpose processor-building function, taking a priority function and success criterion function as arguments, returns an object that corresponds to the architecture instance. The object can be sent initialization items as arbitrary lemmas of the usual form. Whenever a successful lemma is constructed (according to the success criterion) it is returned, along with a continuation function that can be called if further solutions are needed. No processing is done after a successful lemma has been proved unless further solutions are requested.

Using this implementation, we have built instances of the architecture for Earley parsing and the other parsing variants described in this paper, including the shift/reduce simulator. In addition, a generator was built that is complete for semantically monotonic grammars. It is interesting to note that the generator is more than an order of magnitude faster than our original PATR generator, which worked purely by

[10] Since the success criterion requires that a successful item be subsumed by the start nonterminal and the priority filter requires that a successful item's semantics subsume the start nonterminal's semantics, it follows that successful items match the start symbol exactly in semantic information; overgeneration in this sense is not a problem.

[11] Indeed, we might want such incompleteness for certain cases of psycholinguistically motivated parsing models such as the simulated LR model described above.

[12] For reference, the grammar is similar in spirit to the third sample grammar in [Shieber, 1986].

[13] It holds in general that closed-class lexical items—case-marking prepositions, function verbs, etc.—are uniformly considered initial items for purposes of generation because of their vestigial semantics. This is as desired, and follows from the operation of semantic filtering, rather than from any ad hoc techniques.

top-down depth-first backtracking search, that is, following the Prolog search strategy.

The implementation is in Common Lisp and runs on Symbolics 3600, Sun, and Macintosh computers. It is used (in conjunction with a more extensive grammar) as the generation component of the GENESYS system for utterance planning and production.

## 6 Precursors

Perhaps the clearest espousal of the notion of grammar reversability was made by Kay [1975], whose research into functional grammar has been motivated by the desire to "make it possible to generate and analyze sentences with the same grammar." Other researchers have also put such ideas into effect. Jacobs's PHRED system [Jacobs, 1985] "operates from a declarative knowledge base of linguistic knowledge, common to that used by PHRAN", an analyzer for so-called phrasal grammars. Jacobs notes that other systems have shared at least part of the linguistic information for parsing and generation; for instance, the HAM-ANS [Wahlster *et al.*, 1983] and VIE-LANG [Steinacker and Buchberger, 1983] systems utilize the same lexical information for both tasks. Kasper has used a system for parsing grammars in a unification-based formalism (SRI's Z-PATR system) to parse sentences with respect to the large ISI NIGEL grammar, which had been previously used for generation alone.

Nonetheless, all of these systems rely on often radically different architectures for the two processes. Precedent for using a single architecture for both tasks is much more difficult to find. The germ of the idea can be found in the General Syntactic Processor (GSP) designed for the MIND system at Rand. Kaplan and Kay proposed use of the GSP for parsing with respect to augmented transition networks and generation by transformational grammars [Kaplan, 1973]. However, detailed implementation was apparently never carried out. In any case, although the proposal involved using the same architecture, different formalisms (and hence grammars) were presupposed for the two tasks. Running a definite-clause grammar (DCG) "backwards" has been proposed previously, although the normal Prolog execution mechanism renders such a technique unusable in practice. However, alternative execution models might make the practice feasible. As mentioned above, the technique described here is just such an execution model, and is directly related to the Earley deduction model of Pereira and Warren [1983]. Hasida and Isizaki [1987] present another method for generating and analyzing using a DCG-like formalism, which they call dependency propagation. The technique seems to entail using dataflow dependencies implicit in the grammar to control processing in a coroutining manner. The implementation status of their method and its practical utility are as yet unclear.

The use of an agenda and scheduling schemes to allow varying the control structure of a parser also finds precedent in the work of Kaplan [1973] and Kay [1967]. Kay's "powerful parser" and the GSP both employed an agenda mechanism to control additions to the chart. However, the "tasks" placed on the agenda were at the same time more powerful (corresponding to unconstrained rewrite rules) and more procedural (allowing register operations and other procedural constructs). This work merely applies the notion in the context of the simple declarative formalisms presupposed, and provides it with a logical foundation on which a proof of correctness can be developed.[14] Because the formalisms are simpler, the agenda need only keep track of one type of task: addition of a chart item.

## 7 Further Research

Perhaps the most immediate problem raised by the methodology for generation introduced in this paper is the strong requirement of semantic monotonicity, which serves as yet another instance of the straitjacket of compositionality. The semantic-monotonicity constraint allows the goal logical form to be systematically decomposed so that a dynamic-programming generation process can be indexed by the parts of the decomposition (the subformulas), just as the constraint of string concatenation in context-free grammars allows a goal sentence to be systematically decomposed so that a dynamic-programming parsing process can be indexed by the subparts of that decomposition (the

[14]Such a proof is currently in preparation.

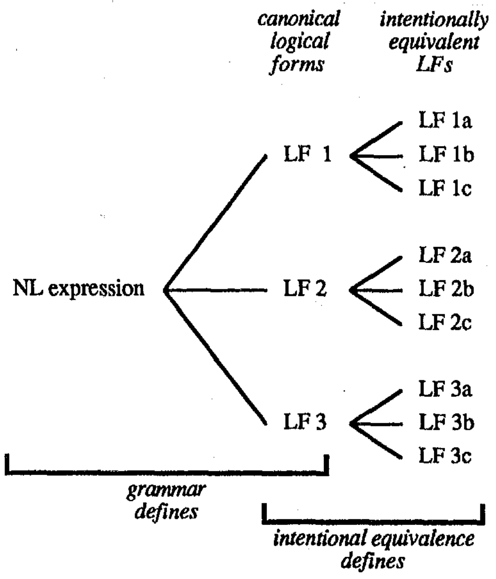

Figure 1: Canonical Logical Forms

substrings). And just as the concatenation restriction of context-free grammars can be problematic, so can the restriction of semantic monotonicity. Finding a weaker constraint on grammars that still allows efficient processing is thus an important research objective.

Even with the semantic-monotonicity constraint, the process of indexing by the highly structured logical forms is not nearly so efficient as indexing by simple integer string positions. Partial match retrieval or similar techniques from the Prolog literature might be useful here.

Nothing has been said about the important problem of guaranteeing that the syntactic and semantic goal properties will actually be realized in the sentence generated. The success criterion for generation described here would require that the logical form for the sentence generated be identical to the goal logical form. However, there is no guarantee that the other properties of the sentence include those of the goal; only compatibility is guaranteed. Researchers at the University of Stuttgart have proposed solutions to this problem based on the type of existential constraint found in lexical-functional grammar. We expect that their methods might be applicable within the presented architecture.

Finally, on a more pessimistic note, we turn to a widespread problem in all systems for automatic generation of natural language, which Appelt [1987] has discussed under the rubric "the problem of logical-form equivalence". The mapping from logical forms to natural-language expressions is in general many-to-one. For instance, the logical forms $red(x) \wedge ball(x)$ and $ball(x) \wedge red(x)$ might both be realized as the nominal 'red ball'. However, most systems for describing the string-LF relation declaratively do so by assigning a minimal set of logical forms to each string, with each logical form standing proxy for all its logical equivalents. The situation is represented graphically as Figure 1.

The problem is complicated further in that, strictly speaking, the class of equivalent logical forms from the standpoint of generation is not really closed under logical equivalence. Instead, what is actually required is a finer-grained notion of *intentional equivalence*, under which, for instance, $p$ and $p \wedge (q \vee \neg q)$ would not necessarily be intentionally equivalent; they might correspond to different utterances, one about $p$ only, the other about both $p$ and $q$.

In such a system, merely using the grammar per se to drive generation (as we do here) allows for the generation of strings from only a subset of the logical-form expressions that have natural-language relata, that is, LF1, LF2, and LF3 in the figure. We might call these the *canonical logical forms*. Even if the grammar is reversible, the problem remains, because logical equivalence is in general not computable. And even in restricted cases in which it is computable, for instance a logic with a confluence property under which all logically equivalent expressions do have a canonical form, the problem is not solved *unless* the notion of canonical form implicit in the logic corresponds exactly to that of the natural-language grammar.

It should be noted that this kind of problem is quite deep. Any system that represents meanings in some way (not necessarily with logical expressions) must face a correlate of this problem. Furthermore, although the issue impinges on syntax because it arises in the realm of grammar, it is primarily a semantic problem, as we will shortly see.

There are two apparent possible approaches to a resolution of this problem. We might use a logic in which logical equivalence classes of expressions are all trivial, that is, any two distinct expressions mean something different. In such a logic, there are no artifactual syntactic remnants in the syntax of the logical language. Furthermore, expressions of the logic must be relatable to expressions of the natural language with a reversible grammar. Alternatively, we could use a logic for which canonical forms, corresponding exactly to the natural language grammar's logical forms, do exist.

The difference between the two approaches is only an apparent one, for in the latter case the equivalence classes of logical forms can be identified as logical forms of a new logical language with no artifactual distinctions. Thus, the second case reduces to the first. The central problem in either case, then, is discovery of a logical language which exactly and uniquely represents all the meaning distinctions of natural language utterances and no others. This holy grail, of course, is just the goal of knowledge representation for natural language in general; we are unlikely to be able to rely on a full solution soon.

However, by looking at approximations of this goal, suitably adapted to the particular problems of generation, we can hope to achieve some progress. In the case of approximations, it does not hold that the two methodologies reduce one to another; in this case, we feel that the second approach—designing a logical language that approximates in its canonical forms those needed for grammatical applications—is more likely to yield good incremental results.

## References

[Appelt, 1987] Douglas E. Appelt. Bidirectional grammars and the design of natural language generation systems. In *Theoretical Issues in Natural Language Processing—3*, pages 185-191, New Mexico State University, Las Cruces, New Mexico, 7-9 January 1987.

[Frazier and Fodor, 1978] Lyn Frazier and Janet Dean Fodor. The sausage machine: a new two-stage parsing model. *Cognition*, 6:291-325, 1978.

[Hasida and Isizaki, 1987] Kôiti Hasida and Syun Isizaki. Dependency propagation: a unified theory of sentence comprehension and generation. In *Proceedings of AAAI-87*, pages 664-670, Seattle, Washington, 13-17 July 1987.

[Jacobs, 1985] Paul S. Jacobs. PHRED: a generator for natural language interfaces. *Computational Linguistics*, 11(4):219-242, October-December 1985.

[Kaplan, 1973] Ronald M. Kaplan. A general syntactic processor. In Randall Rustin, editor, *Natural Language Processing*, pages 193-241, Algorithmics Press, New York, 1973.

[Kay, 1967] Martin Kay. Experiments with a powerful parser. In *Proceedings of the Second International Conference on Computational Linguistics*, August 1967.

[Kay, 1975] Martin Kay. Syntactic processing and functional sentence perspective. In *Theoretical Issues in Natural Language Processing—Supplement to the Proceedings*, pages 12-15, Cambridge, Massachusetts, 10-13 June 1975.

[Pereira and Warren, 1983] Fernando C. N. Pereira and David H. D. Warren. Parsing as deduction. In *Proceedings of the 21st Annual Meeting of the Association for Computational Linguistics*, pages 137-144, Massachusetts Institute of Technology, Cambridge, Massachusetts, 15-17 June 1983.

[Pereira and Shieber, 1987] FernandoC. N. Pereira and Stuart M. Shieber. *Prolog and Natural-Language Analysis.* Volume 10 of *CSLI Lecture Notes*, Center for the Study of Language and Information, Stanford, California, 1987.

[Shieber, 1983] Stuart M. Shieber. Sentence disambiguation by a shift-reduce parsing technique. In *Proceedings of the 21st Annual Meeting of the Association for Computational Linguistics*, pages 113-118, Massachusetts Institute of Technology, Cambridge, Massachusetts, 15-17 June 1983.

[Shieber, 1985] Stuart M. Shieber. Using restriction to extend parsing algorithms for complex-feature-based formalisms. In *Proceedings of the 23rd Annual Meeting of the Association for Computational Linguistics*, pages 145-152, University of Chicago, Chicago, Illinois, 8-12 July 1985.

[Shieber, 1986] Stuart M. Shieber. *An Introduction to Unification-Based Approaches to Grammar.* Volume 4 of *CSLI Lecture Notes*, Center for the Study of Language and Information, Stanford, California, 1986.

[Steinacker and Buchberger, 1983] Ingeborg Steinacker and Ernst Buchberger. Relating syntax and semantics: the syntactico-semantic lexicon of the system VIE-LANG. In *Proceedings of the First Conference of the European Chapter of the Association for Computational Linguistics*, pages 96-100, Pisa, Italy, 1-2 September 1983.

[Wahlster *et al.*, 1983] Wolfgang Wahlster, Heinz Marburger, Anthony Jameson, and Stephan Busemann. Overanswering yes-no questions: extended responses in a natural language interface to a vision system. In *Proceedings of the Eighth International Joint Conference on Artificial Intelligence*, pages 643-646, Karlsruhe, West Germany, 8-12 August 1983.