NATURAL LANGUAGE PROCESSING AND LOGIC PROGRAMMING

VERONICA DAHL

▷ This paper examines the main points of contact between logic programming and natural language processing, and covers some of the important issues that arise using logic programming techniques in natural language processing. It emphasizes the importance of taking into account the most general analyses from linguistic theory in building NLP systems, while also pointing out the need to adapt and combine the different theories to our ends, given that no single one can offer all-encompassing solutions. We concentrate on syntax, the most studied aspect of language processing. Our presentation of different approaches is centered whenever possible around treatment of one particular phenomenon, which serves as an axis of discussion throughout the article: long-distance dependencies between constituents. ◁

1. INTRODUCTION

Of all the enterprises that have been attempted through logic programming, in particular, and by artificial intelligence, in general, processing natural language is arguably one of the most ambitious.

Much has been written about the marvel and wonder of human language, that intricate tool for communicating thoughts, feelings, and emotions, for expressing human nature and society, and for transmitting and clarifying knowledge and belief. Unlike all other tools used by humankind, it is also, in a very real sense, a living creature in constant change. Even if it remained static, or when we slice (as we are forced to do) precise and simplified subsets of it for the purpose of scientific exploration, we are still left with a formidable and intriguing set of linked systems: vocal systems which use sounds as building blocks, systems for combining phonemes into words, systems for combining words and systematic

Address correspondence to Veronica Dahl, Computing Sciences Department, Simon Fraser University, Burnaby, B.C. V5A 1S6 Canada. Email: veronica@cs.sfu.ca.

Received May 1993; accepted December 1993.

traits such as stress and intonation into utterances, systems for interpreting the utterances as symbols for meaning, and systems for representing utterances in written form.

Linguistic theories offer useful but incomplete organizing frameworks for dealing with this complexity, and are by no means in agreement on all points. In addition, the activity of *computationally* processing language often stresses different aspects than those stressed by linguists. Thus, language processing problems have by necessity been addressed without clear guidance from formal linguistics, with some people developing computational linguistic theories of their own, and others adapting various pure linguistic theories for computational use. Interesting mutual feedback between formal and computational linguistics has ensued.

With the advent of logic programming circa 1972 [25], and of logic grammars in 1975 [24], concise logic-based prototypes for language processing became possible [24, 31]. The "parsing as deduction" [90] paradigm had been born, in which a parser is a specialized theorem-prover that can deduce information about utterances in a language from a set of axioms expressing knowledge about that language. Although stressing the parsing aspect for historical reasons, it is clear that the parsing-as-deduction framework can be extended into *language-processing*-as-deduction, in which other aspects of natural language processing are treated through deduction as well (e.g., a specialized theorem-prover can generate sentences in a language by deducing, from a set of axioms expressing knowledge about that language, the *form* of the utterances in that language, that correspond to a given semantic representation).

The attraction in using logic programming for processing language is threefold. In the first place, many of its features are naturally adapted to deal with natural language processing needs: knowledge about language is, in many linguistic frameworks, expressed as some sort of deduction from general principles which can be expressed as axioms; partial, incremental information is also typical in linguistics, and can be expressed through the logical variable; the aims of conciseness and generality are also common to both the field of modern linguistics and of logic programming; and the declarativeness inherent in logic programming is also one of the aims of modern linguistics, important in particular for reversibility (i.e., for using the same program/grammar for analysis as for synthesis). In the second place, logic can provide a natural underpinning for natural language semantics. Third, different types of logics have played important roles in linguistics and in natural language processing. By choosing a logic-based formalism to process language, the implementation means become closer to some of the representations manipulated in processing, thus minimizing the need for interfaces by providing a uniform methodology: logic throughout, in one form or another. Formal characterizations of the logic programming tools developed can also be done in terms of some kind of logic.

This uniformity of methodology was evident from the first Prolog-based natural language systems [25, 24, 31]: logic was used in Horn-clause form (via logic grammars) for programming, and sentences were translated into some kind of logical representation, e.g., into a three-valued logic system, the semantics of which corresponded to answer extraction from an also logic-programmed relational database. Specialized logic systems have been incorporated into several linguistic formulations, particularly regarding semantic representations for natural language (e.g., situation semantics [9] and Montague semantics [42]). Higher-order logics have been used for semantic interpretation [88]. Lambek calculus has been used for inferring the syntactic categories of phrases [73, 74] and for handling gaps[1](e.g.,

[1]In linguistic jargon, a gap denotes the position a moved constituent would have occupied had it not moved, e.g., when the object noun phrase in "Jack built the house" moves to the front through relativization,

[54, 53]). An intuitionistic logic treatment of gaps was presented in [84]. Relevance logic has been used for a formal account of logic grammars [5].

In this article we present an overall view of the evolution and different expressions of the language-processing-as-deduction framework, which as shall be seen, has manifestations other than logic programming ones. We focus on approaches to the description of syntax, while also giving some flavor of semantics and other aspects of processing language. Our presentation of different approaches is centered, whenever possible, around the treatment of one particular phenomenon, which serves as an axis of discussion throughout the article: long-distance dependencies between constituents. We are more attentive to intuitive and comparative explanations than to formalizations which can be found in the literature, and we necessarily resort to simplifications, and sometimes incomplete pictures, for expository purposes. We assume no previous knowledge of the subject.

The paper is organized as follows. Section 2 provides some background as well as some preview for the rest of the paper. Section 3 exemplifies the kinds of problems faced. Section 4 surveys some formalisms and methodologies. Section 5 examines linguistically principled approaches. Section 6 discusses applications, and Section 7 presents our concluding remarks.

2. BACKGROUND

The first language processing application of logic programming, written by Alain Colmerauer and his colleagues [25], was also the very first application of logic programming. Indeed, it was around the needs of processing language that the very idea of Prolog emerged. Its first version was developed not so much as a general purpose AI language, but as a means for solving the deductive problems of a "system of man-machine communication in French."

Colmerauer next developed a grammatical formalism that could compile into Prolog, *metamorphosis grammars*, with the aim of joining thc advantages of the new language with those of clarity and simplicity, where treatment of syntax is concerned, as exhibited in his previous formalism, q-systems [23]. q-systems automate the analysis or synthesis of structures according to a user-defined grammar based on complex-symbol rewriting.

Metamorphosis grammars, or MGs [24], admit type-0-like rules[2]for rewriting symbols (both terminal and nonterminal) which are logic terms. Right-hand sides of rules can also contain Prolog calls. The automatic analysis and synthesis of sentences in the language defined by a metamorphosis grammar are, in principle, both possible (although in practice, procedural concerns dictated separate formulations for analysis than for synthesis, except for very simple grammars).

The first natural language application of metamorphosis grammars was a conversation system in French on social and kin relationships [24]. Then a Spanish and a French front end for the first database system written in Prolog was developed using metamorphosis grammars [31]. These first results were encouraging: an English version of this front end, which was coded in a total of six pages, compared favorably in efficiency with the considerably larger (in the sense of code length) Lunar system [91]. Soon this Spanish front end was adapted to several other languages, other logic-programming-based systems for

as in "the house that Jack built," it leaves a gap behind in the phrase structure representing the sentence.

[2]That is, rules with the same format as type-0 rules, but in which symbols may include arguments, and Prolog calls may be added. Type-0 rules are rewriting rules of the form `X1, . . . , Xm -->Y1, . . . , Ym`, which, as is well known, can be used to define any recursively enumerable language [55].

language processing applications started to emerge, and metamorphosis grammars inspired the development of other types of what are now known as logic grammars [2].

Because the early logic grammars ultimately relied on Prolog's depth-first strategy, which created problems such as nontermination for left-recursive grammars, and in order to solve other specific computational linguistics problems, other proposals emerged, e.g., those based on tabular parsing [118, 4] or on bottom-up parsing [125]. Many evolving logic programming techniques, such as higher-order logics, partial execution, memoing, sharing techniques, constraint logic programming, concurrency, parallelism, and so on, have also been finding application in language processing. For instance, memoization [121] allows us to avoid infinite loops due to left recursion, given that subsumption-equivalent subgoals are not evaluated. Another interesting approach proposed the elimination of left-recursion from DCG rules through successive transformations into a generalized Greibach normal form [43].

In parallel with these developments, many computational linguistics researchers were turning more and more toward the models provided by formal linguistics, after a long period of relying on relatively ad hoc frameworks for constructing their linguistic descriptions.

The transformational or generative paradigm [21] had provided an initial step toward *computationally* usable linguistic models by viewing grammars as highly formalized entities. These entities consisted of two components: a base component of context-free rewriting rules, which described a "canonical" version of sentences (i.e., in the active voice, affirmative form, etc.), and a transformational component, which contained general rules to convert these canonical representations into other possible variants (passive voice, interrogative, or negative form, etc.).

While it was the most formalized linguistic paradigm until then, transformational theory was not easily amenable to computational treatments, mainly due to the myriad of specific rules that it engendered. New theories emerged, all with the objective of brevity of description in mind. Lexical functional grammar [14] was born under the explicit goals of computational preciseness and psychological realism, and replaces transformations by dealing with them in the lexicon. Generalized phrase structure grammars [49] aimed at succinctness by providing higher level grammars that could be mechanically converted into context-free grammars. Government and binding theory [22], as well as its successor, barriers [20], replaced thousands of specific rules by the deductive interaction of a small number of principles and constraints. HPSG [92], with similar aims, takes elements from many of these theories, as well as from semantic and computational theories. Categorial grammars analyze language expressions as the functional product of a functor applied to a suitable set of simpler argument expressions [83]. The categorial grammar approach lends itself very nicely for studying the relationship between the syntactic structures and the semantics of language expressions. All these linguistic models strive in different ways for the same objectives of principledness and succinctness, and in so doing have developed similarities between themselves and also with logic programming. As an example, some notion of unification is also present, although less crucially than in logic programming, in most contemporary linguistic models.

Despite considerable progress made by modern linguistic theories toward formalized accounts of human language, their adaptation for computational use remains difficult, for reasons such as the following:

- Modern linguistics stresses competence (the tacit knowledge that a speaker has of the structure of his/her language) over performance (how language is processed in real time, why speakers say what they say, how language is used in various social groups, etc.), while the latter considerations are more prominent in building natural language systems.

- Linguistic efforts for accounting for competence have, particularly in the past, yielded mostly explanations of language synthesis, whereas computational linguistics is often more interested in analyzing language than in synthesizing it.[3]
- Formalizations of linguistics to the point that it is conceivable to use them for automatic processing are relatively recent and in constant evolution.

Thus, natural language processing is still an art, whose intersection with logic programming is that of two highly promising and complementary, but also rapidly changing scenarios. Cross-fertilization with each other and with other fields is only to be expected, and is indeed happening.

In trying to convey the complexities of this art, we shall aim at an intuitive understanding of some of the main problems and solutions rather than at exhaustive coverage.

3. WHAT KINDS OF PROBLEMS DO WE FACE?

The most explored natural language processing application has been analysis or parsing of individual sentences, to provide natural language front ends to knowledge-based systems.

Given a grammar and a presumed sentence in the language defined by that grammar, the *parsing problem* is to obtain some representative structure(s) (e.g., a tree-form record of the rules applied in order to obtain the sentence) if the sentence is indeed in the language, and nothing if not (representations for the sentence are worth deriving since they will be key to semantics). For instance, Figure 1 shows a simple phrase-structure grammar and the tree structure for the sentence: "Maria laughs." Terminal symbols are enclosed in square brackets.

```
sentence --> noun_phrase, verb_phrase.

    noun_phrase --> name.

    verb_phrase --> verb.

    name    --> [maria].
    verb    --> [laughs].
```

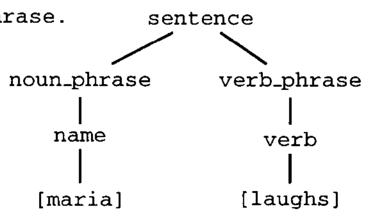


FIGURE 1. A sample grammar and parse tree.

The main problem in parsing is how to describe the infinite possible sentences of a natural language through a finite device, such as a grammar, in as concise and regularity-capturing a way as possible. Context-free grammars are generally believed to be too restricting to deal with natural language syntax, and context-sensitive and type-0 ones, are too computationally intractable, in general [55].

Another problem is that of the ambiguity that plagues natural language, and is exacerbated in written interactions with a computer by the lack of pragmatic clues, intonation clues, etc. A well-known example is "I saw the boy in the park with a telescope", in which it is not clear which phrases the prepositional phrases modify.

[3]Although this is changing, in that modern linguistic theories are intent upon declarativeness and lack of bias toward one processing direction, the change is not as swift as would be desirable (cf. for instance [11])

The linguistically important problem of overgeneration (how to make a grammar describe only sentences in the language, and reject incorrect sentences) can and has been overlooked in many analysis applications, but becomes important for generation, translation, and learning, as well as for descriptive adequacy in the more linguistically principled approaches, and for plain efficiency.

Obtaining semantic representations for sentences is a crucial concern as well. Figure 2 shows a definite clause grammar [91] extension of the grammar in Figure 1, in which nonterminal symbols have been augmented with one to two arguments which build semantic structure. For "Maria laughs," the structure "laughs(maria)" will be obtained—perhaps to be used directly to query a database.

```
sentence(Sem) -->
        noun_Phrase(X), verb_phrase(X,Sem).

noun_phrase(X) --> name(X).

verb_phrase(X,P) --> verb(X,P).

name(maria) --> [maria].
verb(X,laughs(X)) --> [laughs].
```

FIGURE 2. A DCG that builds semantic structures.

More specifically linguistic problems appear immediately when the linguistic coverage becomes reasonably ambitious. For instance, when for emphasis we say "Logic, we love," rather than "We love logic," a parser needs to somehow attach the dislocated constituent "logic" to the gap, i.e., to the missing noun phrase that would have come after the verb. This phenomenon, called the unbounded or long-distance dependency phenomenon, involves constructions such as interrogatives ("Who do you think I saw?"), relatives ("the house that Jack built"), and topicalization ("Friendships like this, one should cultivate," "Logic, we love"), in which some phrase is missing from its standard position in the main clause and a corresponding "extra" phrase appears outside of it. The relation of dependency between the "missing" and the "extra" positions is potentially unbounded, i.e., a string of any length can intervene (e.g., consider "Logic, we love," "Logic, I know we love," "Logic, I suspected he knew we love," and so on). Study of such relations in different languages (e.g., in the Scandinavian languages, where more than one constituent needs to be extracted [49]) suggests that a purely context-free analysis of such relations will not be sufficient.[4]

We will return to these kinds of problems from various different viewpoints, while surveying different formalisms and approaches.

[4]The formal power needed to treat long-distance dependencies has long been a point of controversy. For many years they were thought to be beyond the scope of a nontransformational (i.e., type-0) grammar. However, context-free-based approaches, such as GPSG, provide some treatment of this phenomenon, through augmenting a phrase-structure grammar such that it still remains a phrase-structure grammar, but it can handle long-distance dependencies.

4. FORMALISMS AND METHODOLOGIES

4.1. Earlier Approaches

The earlier formalisms for logic-programming-based computational linguistics incorporated a number of ad hoc methodologies, which were basically clever ways of exploiting the features of logic programming. For instance, some cases of ambiguity and of semantic anomaly were dealt with by attaching semantic types to the logic terms induced by a grammar, and letting unification decide whether two types were compatible or not. Thus, by a parser attaching the type "human" to the subject of "speak," a sentence such as *Which dogs speak Latin?* could be rejected on the basis of semantic anomaly detected through syntactic (un)matching of types. Similarly, the ambiguity in *Where does Georgia live?*, where Georgia denotes either a woman or a place, could be similarly resolved by requiring an "animate" type for the subject of "live." Types were represented by terms such that even set inclusion verifications could basically reduce to unification. Further extensions, such as logic grammar formalisms for automating semantic structure buildup, treatments of coordination, or quantifier scoping [76, 36, 1, 57] provided further perspicuity.

These initial, ad hoc approaches were succeeded by more linguistically and/or logically justified formulations, such as the type treatments of [79, 78, 3, 34, 17] or the higher-order logic formulation of semantic structure buildup in [80].

In the remainder of this section, we present some early formalisms around the axis of one particular type of linguistic phenomenon: that of relating long-distance constituents.

4.1.1. DEFINITE CLAUSE GRAMMARS (DCGs) [91]. DCGs have already been informally introduced through the example in Figure 2. DCG rules have the general form

```
        s1 --> s2,..., sn.
```

where s1 is a complex grammar symbol (i.e., may be a term) and s2,...,sn are either complex grammar symbols or Prolog calls. If the latter is true, they are enclosed in curly brackets. Terminal symbols are noted between square brackets and can also contain arguments (although this is unusual). DCG rules compile into Prolog by adding two arguments to each symbol: one with the string to be analyzed, and another one with what remains of it after the predicate corresponding to the symbol has perhaps consumed a part of it.

While following the general context-free format, the arguments allowed in the grammar symbols endow them with type-0 power: they can be used to carry any context-sensitive or movement information that, in a grammar with only simple symbols, would appear through additional left-hand-side symbols instead.

Figure 3, for instance, shows a DCG (taken from [85]) which describes the long-distance dependency between a pronoun and a missing noun phrase in a relative clause, through carrying in an extra argument information on when the noun phrase will be missing. It moreover illustrates the difficult art of arriving at a grammar formulation that maintains generality and conciseness while ruling out incorrect sentences. The rule for s blocks movement out of subject noun phrases (which would, in a larger grammar, result in incorrect noun phrases such as **the lost puppy that a reward for was offered*,[5]) by requiring subject noun phrases to be complete. However, it has the undesirable side effect of also blocking relative clauses where the whole subject np is missing, as in: *the puppy that was found*

[5]We precede ill-formed sequences of words by an *, as is common practice in linguistic literature.

yesterday. The second rule for relative is introduced solely to correct this side effect. The same analysis can also be found in some versions of GPSG, and represents a reasonable compromise solution for the problem.

```
sentence --> s(complete).
s(E) --> noun_phrase(complete), verb_phrase(E).

noun_phrase(complete) --> determiner, noun, relative.
noun_phrase(complete) --> name.
noun_phrase(missing_np) --> [].

verb_phrase (complete) --> verb.
verb_phrase (E) --> verb, noun_phrase(E).

relative --> pronoun, s(missing_np).
relative --> pronoun, verb_phrase(complete).
```

FIGURE 3. A DCG for relativization.

In most contemporary implementations, DCGs also allow for more than one grammar symbol in their left-hand side, provided that the first left-hand-side symbol is a nonterminal. This full version was first developed by Alain Colmerauer under the name of metamorphosis grammars (MGs) [24], but the name DCG has become prevalent since. Using metamorphosis grammars, movement rules can be directly expressed rather than conveyed through argument manipulation. A subject-verb inversion, as occurs in interrogative sentences in some romance languages, can be described by MG rules such as

```
interrogative_marker, noun_phrase, verb --> verb, noun_phrase.,
```

where "interrogative_marker" has been introduced by rules such as

```
sentence(interrogative) --> interrogative_marker, s.
sentence (affirmative) --> s.
```

Notice that records of rule application for MGs are no longer trees, as in DCGs, but graphs, owing to the right-hand-side symbols having more than one parent. Any MG has an equivalent "normalized" formulation, in which all nonleading left-hand-side symbols are (perhaps pseudo-) terminals (most implementations only accept normalized formulations). However, this equivalence is weak, in the sense that while both grammars will analyze the same set of sentences, in normalized MGs the rules added for normalization will alter the structural description corresponding to those sentences. Normalized MGs may also generate spurious sentences besides those of the intended language, owing to the pseudoterminals introduced. Normalized MG rules compile into Horn clauses in the same way as those rules with only one left-hand side symbol, with the nonleading terminal symbols being recorded inside the string-manipulation arguments.

4.1.2. EXTRAPOSITION GRAMMARS (XGs) [85]. XGs allow us to refer to unspecified strings of symbols in a rule, thus making it easier to describe left extraposition of constituents. Left extraposition refers to a view of the genesis of a sentence as a process during which, for instance, a relative pronoun introducing a relative clause is seen as having been moved from the position of a subject or an object in the clause. Thus, in

The house [that$_i$ Jack built [t$_i$]] was comfortable. , we have bracketed the relative clause and indicated the gap (the position in which the object would have appeared in a main sentence) with a trace, t_i. The trace is coindexed with the pronoun to express that they both refer to the same entity.

The following sample XG rule allows for a noun phrase which would normally be expected somewhere at the right of its subject to be left-extraposed to the position of a pronoun representing that subject, and subsumed by it, its index I becoming that of the pronoun[6]:

```
rel_marker(I), skip(X), trace(I) --> rel_pronoun(I), skip(X).
```

For uniformity of exposition, we adopt above the notation skip(X) for any unspecified string of symbols X, instead of the original notation "... ." `rel_marker` is introduced for identifying a sentence as a relative clause, and a trace is introduced by one of the noun phrase rules:

```
relative --> rel_marker, sentence.
np --> trace(_).
```

In XGs, the skipped substrings must always follow the remaining right-hand-side symbols in their original order; thus they are left implicit on the right-hand side in the original notation. XGs are furthermore constrained in the nesting of skipped substrings: two skips must either be independent, or one skip must lie entirely within the other.

Nonlogic-based approaches to long-distance dependencies have also been explored. For instance, following Woods' treatment [124] using augmented transition networks [123], material corresponding to "the house" in "the house that Jack built was comfortable" would be remembered in a global register called HOLD, from which the relevant information would be retrieved upon detecting a missing noun phrase.

4.2. A Cross-Disciplinary Methodology—Chart Parsing and Earley Deduction

Perhaps one of the most interesting examples of cross-disciplinary feedback in our areas of concern is the relationship between parsing, logic programming, and deductive databases.

Chart parsers [69, 71] remember substructures parsed by noting them in a chart which remains available even after failure and backtracking. This is useful for instance when trying to parse "Alan Robinson discovered the resolution principle in the sixties." For the verb phrase, the following two rules may be tried in order:

```
vp --> v, np.
vp --> v, np, pp.
```

If in trying the first rule, we record in a chart the fact that "discovered" is a verb and "the resolution principle" is a noun phrase, when that rule fails, we will still be able to use that immediate knowledge upon trying the second rule, rather than parsing these subphrases

[6]Notice that in order to obtain coindexing in the DCG formulation of Section 4.1.1, one would have to unnaturally pass on the argument containing the index from the pronoun to the noun phrase constituent, through other symbols such as s.

again.

An ingenious chart parsing algorithm was developed by Earley [44], in which the chart can store unfinished constituents as well as completely parsed ones. Just as in the magic set models of deductive databases, it uses a combination of top-down processing (by constructing from old chart entries, new chart entries in which constituents are expanded, e.g., "vp" is expanded into "v, np") and bottom-up processing (by looking for constituents that have just been completely recognized and using them to complete higher-level constituents). Loops are avoided by ensuring that no entries in the chart are duplicated. Backtracking never occurs—instead, the parser pursues all alternatives concurrently, and at the end, all alternative parses are in the chart. For grammars in which constituents have no arguments, sentences of length n are parsed in, at most, time proportional to n^3.

This theoretically encouraging result is, however, less interesting in practice, owing to the overhead of the necessary bookkeeping and loop-checking operations, and to the fact that we are often interested in allowing constituents with arguments. Spacewise, chart parsing techniques have been developed which can encode all possible parses as a data structure with size polynomial in the length of the sentence. However, as noted in [13], sophistication in chart parsing schemata may reduce time and space efficiency instead of improving it.

In an unpublished 1975 note, David D. H. Warren proposed the transfer of the Earley algorithm ideas about parsing into logic programming proper. Memoization, or the storing of lemmas in order to avoid repetition of proofs upon backtracking, has since been investigated within "Earley deduction" [90, 89] and incorporated into other logic programming settings, such as constraint logic programming [62, 59, 121].

Just as its parsing analogue, Earley deduction avoids loops because it never stores a lemma which is already on the chart, or which is subsumed by a lemma already stored. Similarly, although avoiding exponential search for restricted classes of programs (the task of deduction, of course, remaining undecidable in general), it does so at the cost of subsumption checking and of managing the alternative binding environments associated to stored lemmas.

These kinds of problems, of course, are not exclusive to Earley deduction or parsing, but arise in all logic programming frameworks that depart from the standard WAM-based implementation by using structure sharing to simultaneously maintain several computation branches (e.g., parallel models [40, 52, 119] and magic set models [7, 100, 112]). Work by Lang and his collaborators [75, 117, 13] proposes interesting solutions to those problems. For instance, the structure-sharing framework called *layer sharing* [117] allows multiple environment management, and can cleanly join two computation paths (the current one and the reused one) while renaming to avoid variable clashes. Variables are accessed and bound in constant time, and time overhead seems to be kept minimal. Although these results are preliminary, done with a prototype implementation, it would be interesting to investigate how well they transfer to natural language applications of logic programming.

5. LINGUISTICALLY PRINCIPLED APPROACHES

As mentioned in the introduction, computational linguistics concepts are merging more and more with concepts from linguistic theories. This is sometimes the surprisingly converging result of an independent evolution, as in the case of the unification-based formalisms that have mushroomed in linguistic theories independently of, but closely resembling, the logic programming notion of unification.

Together with the idea of inferencing, the ideas of unification and constraints are perhaps the most powerful connection between the areas of natural language processing through logic programming and linguistic theory. However, these notions often have different connotations in each of these fields.

This section presents three important families of linguistically principled approaches to natural language processing: unification-based (also known as constraint-based), logico-mathematical, and principles-and-parameters. This classification is far from universally agreed upon, being just our own modest attempt to organize the vast material in the literature around the language-processing-as-deduction axis, for expository purposes. It is moreover not a clearly disjoint classification: as shall be seen, some of the approaches described under a specific family also partake of some features of another family, but for our purposes here, this rough classification will hopefully suffice.

5.1. Unification-Based Approaches

Unification in linguistic theories often refers to the combination of compatible feature structures rather than of first-order terms. Feature structures are representations of partial information made in terms of features or attributes and their values. A value can be either undefined (roughly corresponding to a variable in logic programming) or another feature structure. Unification combines two feature structures into another if the information conveyed in both is consistent. For instance, the unification of feature structures (1) and (2) below produces the new feature structure (3):

$$\begin{bmatrix} \text{agreement} : \boxed{1}\,[\,\text{number} : \text{singular}\,] \\ \text{subject} : [\,\text{agreement} : \boxed{1}\,] \end{bmatrix}, \tag{1}$$

$$[\,\text{subject} : [\,\text{agreement} : [\,\text{person} : \text{third}\,]\,]\,], \tag{2}$$

$$\begin{bmatrix} \text{agreement} : \boxed{1} \begin{bmatrix} \text{number} : \text{singular} \\ \text{person} : \text{third} \end{bmatrix} \\ \text{subject} : [\text{agreement} : \boxed{1}\,] \end{bmatrix}. \tag{3}$$

Shared information is marked by numeric labels rather than by shared variables; thus the value of the *agreement* feature of the subject is understood to be the same as that of the whole structure's agreement feature (i.e., the feature-value pair "number:singular"), because they share label $\boxed{1}$. As in logic programming, the information flow is two-way (e.g., one of the structures unified contributes the value for the *number* feature, while the other one does so for *person*). However, unlike logic programming, the handling of partial information does not necessitate explicit arguments or any other explicit position for the unkown information: it is simply omitted when not there and added on when unification calls for its addition.

We next show a sample feature-based grammar fragment in which, for convenience, we use variables instead of labels, and we associate each syntactic category with the list of its feature-value pairs:

```
S -->       NP              VP
         [num:X,         [num:X]
         case:nom]

NP                 -->     name
```

```
[num:X,                    [num:X,
case:Y]                    case:Y]

NP               -->     Pronoun
[num:X,                  [num:X,
case:Y]                  case:Y]

VP               -->       V
[num:X]                  [num:X,
                         subcat:1]

VP               -->       V              NP
[num:X]                   [num:X,      [case:acc]
                          subcat:2]

Name             -->  Maria              V                -->  laughs
[num:sing]                               [num:sing,
                                          subcat:1]

Pronoun          -->  them               V                -->  designs
[num:plu,                                [num:sing,
case:acc]                                 subcat:2]
```

With respect to this grammar, we can analyze the sentence "Maria designs them," for instance, as in Figure 4 (note that the direction of analysis—top-down or bottom-up—is irrelevant).

Because feature structures have a natural representation as graphs, the linguistic notion of unification is often that of *graph* unification.

A variety of unification-based formalisms has independently sprung from different fields—linguistics, computational linguistics, and artificial intelligence. This diversity resulted in a variety of names and definitions. Thus, the unification-based framework can also be found in the literature under the names *constraint-based* or *information-based.* The most specific definitions (e.g., [26]) characterize these formalisms as those in which:

- Complex feature structures encode partial information about constraints.
- Features are given values only through unification, and not through any other kind of computation.

A more abstract characterization has been given in [103], in terms of information and constraints on it. In this view, information is described through desired properties rather than through their concrete incarnations. These properties are:

- Modularity (information must be partitioned into different, declarative modules)
- Partiality (information does not need to be complete at all times, but can arise from the combination of partial information available from a variety of sources)
- Equationality (informational constraints interact to place strong limits on the distribution of a phrase, rather than giving information about that phrase directly)

A grammar formalism can then be described through logical constraints over information associated with phrases.

Attribute value-based grammars have been formally described as systems of logic by Johnson [60]. More recently, Carpenter provided theoretical foundations for the specification and implementation of systems using feature structures, and examined their applications to unification-based grammars, logic programming, and constraint resolution [17]. John-

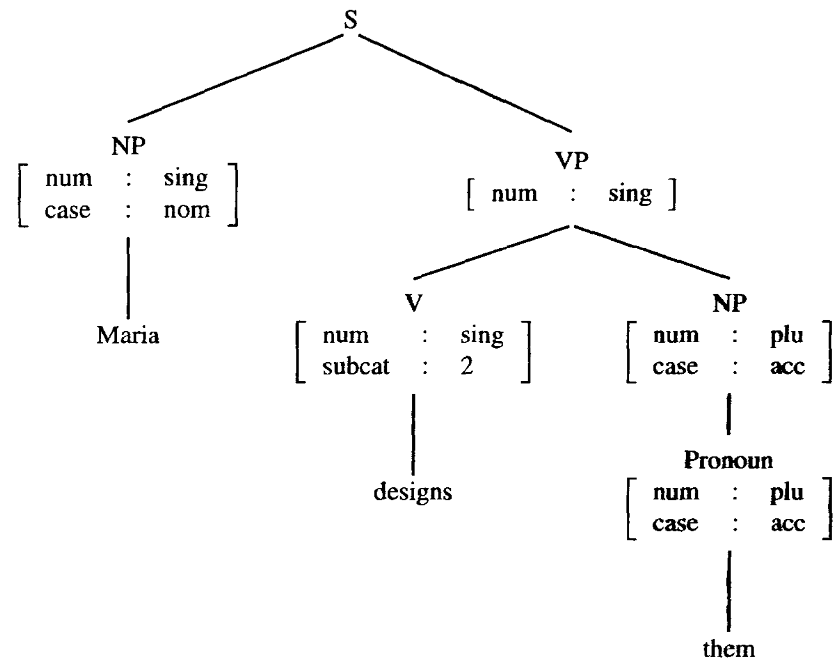


FIGURE 4. A sample feature-based analysis.

son has also shown how to express a variety of different types of feature structures and constraints using predicate logic, or standard nonmonotonic extensions thereof [61, 59]. Shieber has developed foundations for constraint-based or unification-based formalisms with a strong pivotal point on PATR-II [103].

Just as is the case with unification in logic programming, feature-structure unification can be seen as a specific kind of constraint enforcement. Also as in logic programming, other methods for constraint enforcement are used as well. Some of these are adaptations of constraint logic programming frameworks, tailored to natural language applications, including, for instance, equational constraints, or constraints requiring the existence of values, set membership constraints, etc. Such constraints are in one way or another present in formalisms like augmented transition networks [123], lexical functional grammars [64], functional unification grammar [70], logic grammars, grammars in the generalized phrase-structure grammar framework, as evolved from [48], and the PATR-II formalism [95]. It has recently been shown that feature structures may be essentially interpreted as terms allowing freedom of order and number of the arguments (e.g., [81, 17]). Natural language tailored CLP techniques have been studied, such as memoization within CLP [59], constraint extensions of Earley deduction [62], and the incorporation of feature-structure constraints into logic programming [19].

Other linguistically motivated constraints that have been argued as necessary include constraints on parse histories. An axiomatization of contraints in first-order predicate calculus is provided as part of Stabler's axiomatization of Chomsky's barriers theory [107]. Disjunction, negation, and conditional descriptions in unification systems have also been argued as necessary, and have been incorporated in various approaches [66, 88, 17, 67, 68, 45].

We next briefly present two unification-based approaches to natural language processing.

5.1.1. Lexical Functional Grammar (LFG). LFG was developed in the late 1970s by Joan Bresnan and Ron Kaplan, with the specific goal of providing a computationally precise and psychologically realistic model of language. It includes an enriched *lexicon* which characterizes, for instance, the relationship between active and passive constructions as well as that information which other theories view as strictly lexical information.

Lexical items from this enriched lexicon are inserted into *c-structures*, or constituent structures, which express language-dependent properties such as word order and phrasal structure. The invariant grammatical constraints, e.g., on agreement, are mostly stated on a third component of LFG called *f-structures*, or functional structures. These different levels have different kinds of representations and obey their own constraints.

This organization into just three levels of representation has interesting implications. For instance, long distance dependencies, or the relating of two positions which may be arbitrarily separated from one another, were previously accounted for in terms of two successive structures. Thus, to relate "Which principle" with"__" in

Which principle did John believe __ had been discovered?,

the transformational analysis has one level of structure in which "which principle" is in the position of " __ ", and another one in which it has moved by transformation into its final position.

Because LFG postulates a single phrase structure level (namely, c-structure), this analysis is not possible. In the 1982 version of LFG, a mechanism called *functional control* was used instead, which allows "which principle" to belong in the *f*-structure representations of *both positions* simultaneously, once under the FOCUS function, and next under the TOPIC function. Recent proposals for the treatment of unbounded dependency in LFG make use of functionality uncertainty, which allows the expression of regular paths in functional descriptions (see, for example, [65]).

Functions are also useful to resolve the conflict between the intuition that coordination takes place between phrases "of the same kind" and the existence of counterexamples, such as "Mario was asleep and causing a traffic stoppage" [101], in which an adjective phrase (AP) is conjoined with a verb phrase (VP). The LFG analysis retains identity of the conjoined phrases, but identity of function, not of category. Both conjoined phrases have a function called XCOMP, which may be realized by either an AP or a VP.

A more detailed discussion of these issues can be found for instance in [101].

5.1.2. Generalized Phrase Structure Grammar (*GPSG*) [49], GPSG takes categories to be sets of feature-value pairs. Their rules are of two kinds:

1. ID/LP rules, which factor out information on *immediate dominance* (ID) and of *linear precedence* (LP). A rule's right-hand side is a multiset rather than a list of symbols, and LP rules indicate precedence requirements. For instance, the ID rule s --> a, b, c together with the linear precedence rule a < c ("a precedes c") is a shorthand for the context-free rules: s --> a,b,c; s --> b,a,c; s --> a,c,b.
2. Metarules, which derive alternate forms of sentences from a basic core of description.

For instance, a metarule for passivization will generate, from a rule for VP in the active form, a corresponding passive form rule.

A related framework, head-driven phrase-structure grammar (HPSG), expresses metarules as lexical rules, eliminating lexical ID rules as such and stating subcategorization as a property of lexical heads (i.e., lexical items that head a phrase make explicit in the lexicon what modifiers they require) [92]. It includes such general principles as the head feature principle (HFP)[7] which requires the head features of a phrase to be shared with its head daughter (e.g., the case of a noun phrase is determined by the case of its head noun). The HPSG approach could be grouped with the unification-based approaches, but probably fits better under the principles-and-constraints family (see Section 5.3.1).

5.2. Logico-Mathematical Approaches

5.2.1. HIGHER ORDER LOGICS. Intuitionistic and linear logic treatments of computational linguistics issues such as long-distance dependencies have been proposed for instance in [84, 80, 88, 53], mostly within the general GPSG framework.

The relativization process, for instance, can be expressed in intuitionistic logic through the inference rule of implication introduction

```
G, D --> G        => R
-----------
G --> D =>G
```

in which clauses in D are only available during the proof of G.

Pareschi and Miller [84] propose using this rule, rather than the extra argument technique of Figure 3, in order to control when the empty noun phrase is used. Instead of having an independent rule that makes a noun phrase empty, we can simply incorporate it as the D part of an implication introduction rule which specifically introduces a relative clause, so that a noun phrase can only be made empty while parsing a relative clause.

For instance, the first grammar rule for *relative* in Figure 3 can be represented as follows in λ-Prolog (input and output string arguments, which logic grammars usually make invisible, need to be explicit in this formulation—they are noted with no commas):

```
relative (that::L1) L2 :-  (np Z Z) => s L1 L2.
```

The np clause stretching between the string Z and the same string Z (i.e., describing an empty np) is only available during the proof that there is a sentence between strings L1 and L2, within the proof that there is a relative clause between a string L1 fronted by that and a string L2. This prevents empty noun phrases from being generated outside relative clauses.

This approach, however, still admits incorrect sentences, the avoidance of which would involve cumbersome additions. In particular, the freely available rule of weakening, which allows unused assumptions to be simply discarded, results in relative clauses with no empty noun phrase (as in * the house [that Jack built the house]). Linear logic [53] has been proposed to remedy this problem, but it does not remedy other problems, involving both over- and undergeneration.

Other treatments of long-distance dependencies include that of SDGs [33] (cf. Section 5.3.3), which has been given a relevance logic characterization [5]. Variants of linear logic have been used for other computational linguistics purposes, such as inferring the syntactic

[7]The HFP is an adaptation of a similar principle which can be found in GPSG.

categories of phrases [73, 74].

5.2.2. Categorial Grammars. The line of work starting with Lambek [73, 74] and leading to modern categorial grammar [114, 82, 115], which has close relations with lambda calculus and with noncommutative linear logic, is an important instance of the "parsing as deduction" paradigm. Categorial grammars are grammars in which information about all possible combinations of constituents is embedded in their categories. They are based on the idea that language expressions can be analyzed as the functional product of a functor applied to a suitable set of simpler argument expressions [83].

Categories are divided into *basic* and *derived*. Basic categories are just category symbols. Derived categories, noted A/B, where A and B are (simple or derived) categories, may be identified with functions which map expressions of category B into the set of expressions of category A. The concatenation of an expression E_1 of category A/B with an expression E_2 of category B is an expression of category A. When E_2 is expected at the left of the function rather than the right, the function is noted A\B.

For instance, if our basic categories are N and S, we can define the category for "likes" as the derived category (S\N)/N. Then if we attach the category N to the lexical items "Maria" and "Caitlin," we can analyze "Maria likes Caitlin" as an S, as follows:

```
Maria          likes          Caitlin
-----          --------       -------
  N            (S\N)/N           N
               ----------------------
                         S\N
-------------------------------------
                   S
```

Thus, a categorial grammar is defined by specifying the categories of basic expressions. The language generated by such a grammar is the closure of the set of basic expressions under functional product. Our ubiquitous example, long-distance dependencies, can be analyzed as involving categories produced under a composition operator [108].

Since functions and arguments need not be restricted to those that have only syntactic properties, categorial grammars can be used to study the composition of grammatical expressions across a variety of domains: syntactic, semantic, phonological, etc.

Various extensions of categorial grammar have been proposed. Flexible categorial grammars, including rules for "type change" of expressions have been studied for instance in [114]. Ways of merging the categorial and the unification grammar framework have been studied, for instance, in [113, 126]. We next briefly exemplify such merges by describing the latter approach.

Unification categorial grammar (UCG) [126] is a computationally motivated extension of categorial grammar which incorporates elements from PATR-II. It evolved from a concern to integrate syntax and semantics as tightly as possible, and to use results from Kamp's work on discourse representation [63] while preserving compositionality.

UCG defines a *sign* as a (complete or incomplete) list of the following representations for an expression:

- W (the expression's phonology)
- C (its syntactic category)
- S (a semantic representation)
- O (its order)

also written: W:C:S:O.

Function application involves two steps, exemplified below in the application of a sign, as a functor, to an argument sign. Unspecified attributes are omitted (e.g., "order"), and left-associativity of "/" is assumed:

```
FUNCTOR:          walks
                  sent[fin]/np[nom]:x:pre
                  [e] WALK(e,x)

ARGUMENT:         john
                  np
JOHN
```

Step 1: Unify the active sign of the functor (its portion after "/") with the argument. In our example, the active sign of the functor becomes

```
john:np[nom]:JOHN:pre .
```

Step 2: Obtain a new sign by:

(a) Stripping its category down, from A/B to just A. In our example, we get:

```
walks
sent[fin]
[e] WALK(e,john) .
```

and

(b) Replacing its phonology by that of the functor concatenated with that of B. In our example, this yields

```
walks john
sent[fin]
[e] WALK(e,john) .
```

In practice, a more complex lexical entry for "John," combined with an appropriate use of the variable O (order), yields the correct ordering "John walks."

Even in this very simplified example, it is clear that semantic restrictions are, just as the syntactic ones, imposed through unification: if it is not possible to construct a new semantics through unification, the derivation is blocked. Thus different levels of representation—semantic, syntactic, and phonological—are built up simultaneously by the uniform device of unification.

Feature percolation from implementing universal principles as the head-feature structure, it is argued, is no longer needed when using UCG: these principles can be hard-wired in a categorial setting, with no need for additional stipulations. UCG allows simplification of the set of categories used, since the category identity of a function application typically depends on the makeup of the argument: the same functor applied to two different arguments will yield different results.

5.3. Principle-and-Constraints Approaches

Approaches based on principles and constraints, such as government-binding, barriers, and HPSG, move away from construction-specific and language-particular rules, replacing them by a small set of universally valid principles plus a set of language-specific constraints, from whose deductive interactions the necessary constructions follow.

5.3.1. HEAD-DRIVEN PHRASE-STRUCTURE GRAMMAR (HPSG) [92]. HPSG is based on the intuition that each phrase contains a central word, called its lexical head, which determines many of the syntactic properties of that phrase (e.g., for a verb phrase, the verb; for a prepositional phrase, the preposition; etc.).

This framework brings together ideas from various syntactic, semantic, and computational theories.

From situation semantics [9] it derives an adaptation of the Saussurean definition for a sign [98]: a *sign* is a meaning relation (constraint) between two components called the *signifier* or *utterance situation* (e.g., one in which the word "moon" is uttered) and a certain property of things in the world (e.g., the property of being the celestial object referred to by the word "moon").

A sign such as the lexical sign *moon* can be described by a feature structure with attributes PHON (phonology), SYN (syntax), and SEM (semantics). Syntactic features are further classified as LOC (local) and BIND (binding).

Local features, in general, specify inherent syntactic properties of a sign (e.g., part of speech, inflection, case, etc.) and lexicality (whether a sign is lexical or phrasal). *Binding* features are nonlocal in the sense that they provide information about long-distance dependencies.

Local syntactic features are further classified into *head features*, *subcategorization features* (SUBCAT), and *lexical features* (LEX). Head features specify syntactic properties that a lexical sign shares with its projections (i.e., the phrasal signs headed by a lexical sign). Subcategorization features express what kinds of phrasal signs the sign in question typically combines with (e.g., for *walk*, the SUBCAT list is $<NP$[NOM]$>$, indicating that *walk* must combine with a single NP in the nominative case). LEX is a binary feature which distinguishes between lexical and nonlexical signs. Feature structures of nonlexical signs have a fourth attribute, daughters (DTRS), which is further described below. Binding features include SLASH, which provides information about gaps and their binding to an appropriate dislocated constituent (as in "Which principle did John believe _ had been discovered," in which "Which principle" must be bound to the gap, represented '_]"), REL, which give informat

ion about unbound relative elements in the sign, and QUE, which performs the same function about interrogative elements.

Thus the overall structure of a sign can be depicted as

$$\begin{bmatrix} \mathit{PHON} & \\ \mathit{SYN} & \begin{bmatrix} \mathit{LOC} & \begin{bmatrix} \mathit{HEAD} \\ \mathit{SUBCAT} \\ \mathit{LEX} \end{bmatrix} \\ \mathit{BIND} & \begin{bmatrix} \mathit{SLASH} \\ \mathit{REL} \\ \mathit{QUE} \end{bmatrix} \end{bmatrix} \\ \mathit{SEM} & \end{bmatrix}.$$

Subcomponents of such structures can be indicated through paths along them, e.g., SYN/LOC/HEAD indicates the phrase's head. DTRS in phrasal signs are further classified into HEAD-DTR (head daughter—these share their head features with the mother), COMP-DTR (complements—these discharge subcategorization requirements on the head), FILLER_DRT (fillers—these discharge binding requirements on the head), etc.

Principles of universal grammar (those which are language-independent, i.e., apply on all human languages) can be expressed in terms of feature structure descriptions. For instance the head feature principle (HFP) which, as we have seen, requires for the head features of a phrase to be shared with its head daughter, can be stated as

$$[\ DTRS_{\text{headed\_structure}}\ [\ ]\ ] \Longrightarrow \begin{bmatrix} SYN/LOC/HEAD\ \boxed{1} \\ DTRS/HEAD\text{-}DTR/SYN/LOC/HEAD\ \boxed{1} \end{bmatrix}$$

(if a phrase has a head daughter, then they share the same head features). The feature information can be thought of as flowing from the head daughter to the phrase, through unification.

HPSG treatments of long-distance dependencies also use this "flowing" mechanism, through transmitting a gap's feature specifications more or less freely from arbitrary daughters (not just head daughters) to their mothers, until they can be unified with the appropriate dislocated constituent.

For instance, in "Logic, I know we like _", the gap position is described with the aid of a feature SLASH with value NP (where SLASH stands for "missing"). The description NP[*SLASH*NP] (noun phrase missing a noun phrase) is associated with the gap, after being determined by the subcategorization restriction exerted on the gap position by the lexical head *like*. The "missing NP" information encoded as NP[*SLASH*NP] is transmitted up to the larger signs that contain it (thus for instance "we like -" is characterized as VP[*SLASH*NP] (a verb phrase missing a noun phrase), and so on until we reach a point in the structure in which a suitable mechanism (the grammar rule responsible for topicalization) recognizes *logic* as the missing noun phrase and identifies its local syntactic features with those of the missing noun phrase.

A simplified HPSG sample analysis for this type of sentences [92] is shown in Figure 5. HPSG-inspired approaches include those discussed in the following paragraphs.

Constraint logic grammars (CLGs) [6, 39] were inspired in the CLP paradigm [58]. They are implemented in Prolog and use delayed evaluation of nonequational constraints which are represented in a slightly restricted form of first-order predicate calculus. Two prototypes have been extensively tested with nontrivial grammars of several European languages.

Constraints are of two types: (i) complex constraints, which are nonatomic formulas of the constraint language, and (ii) global constraints, which encode HPSG types of linguistic principles under the general form

```
Partial specification --> constraints .
```

These facilities allow us to code linguistic principles in a fairly direct and high-level manner. For instance, the head feature principle, described in linguistic terms above, when encoded in CLG, looks like

```
[head_dtr= [_]] --> syn.local.head= head.dtr.syn.local.head .
```

Partial descriptions for lexical signs have the form <DAG,CS>, where the first argument is a directed acyclic graph representing a feature structure and the second argument is a set of complex constraints. Partial descriptions of phrasal signs are expressed through CLG(2)

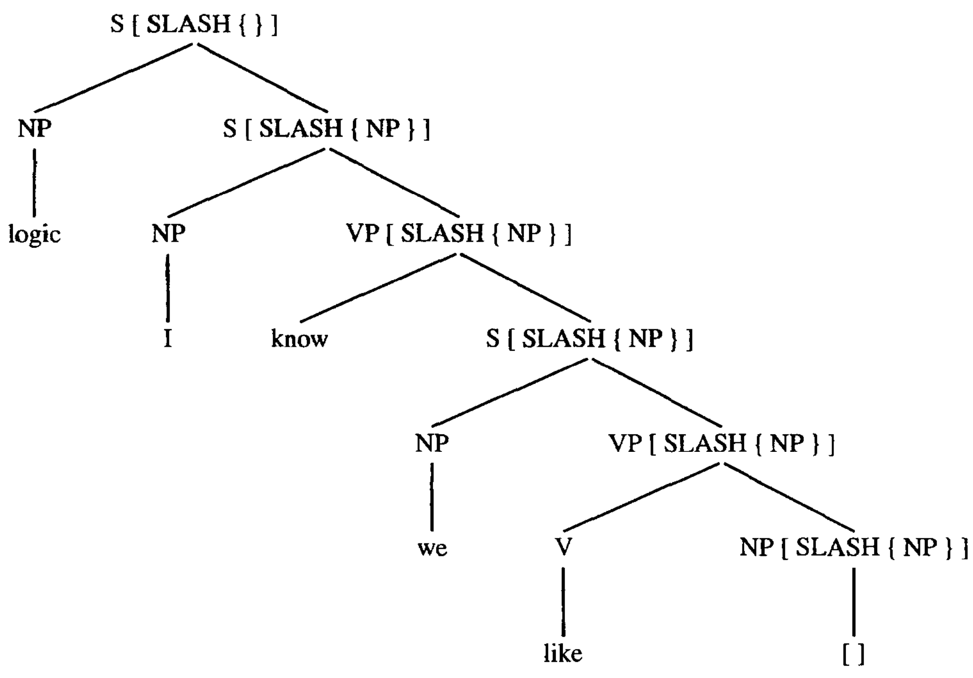


FIGURE 5. A sample HPSG analysis.

rules, which support a number of different, equivalent rule formats. Type information is used both to structure the grammatical information and for efficiency.

The *comprehensive unification formalism* (CUF) [41] is a logical language for the expression of linguistic information, based on equations over feature structures. It is used as an intermediate level between linguistic theories, such as HPSG, and their algorithmic interpretation. A Prolog III implementation of CUF has been described in [99], which allows functional notation and indexing of shared structures, as well as statement of negative or disjunctive information, thus making it possible to compress linguistic information (for instance, when the equivalent positive formulation is longer than the negative one).

Principle formulations are also close in CUF to their linguistic expressions. Taking the same example as before, the head feature principle can be encoded

```
head_feature_principle :=
        dtrs : head_dtr :  syn : loc : head : X
        &
        syn : loc : head : X.
```

CU-Prolog [111] is a CLP language for parsing, in which constraints are user-defined predicates which constrain symbolic objects. The rule format is

H: B1, B2, ..., Bn ; C1, C2, ..., Cm .

The H and the Bs are, respectively, the head and the body as in regular Prolog, while the

Cs are constraints on the variables occurring in the rest of the clause. Constraints must obey certain restrictions: all arguments must be variable; no variable can occur in two different places, etc.

CU-Prolog has been used to develop a Japanese grammar theory called JPSG [51], based on the linguistic HPSG theory .

Typed unification grammars (TUG) [46] represent an object-oriented approach to computational linguistics, in which a type-inheritance mechanism allows us to define classes and subclasses of objects, and a corresponding evaluation mechanism can compute relations between classes of objects.

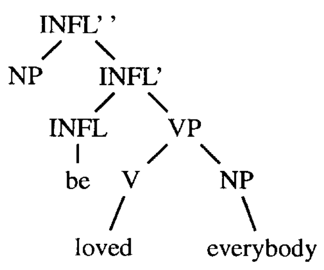
TUG are in the class of logic formalisms and can be used both for parsing and for generation. Here again, most HPSG principles are described fairly directly in TUG. Provision is made for more complex operations than the simple combination of information. For instance, the subcategorization principle is written in terms of "append."

5.3.2. Chomskian Approaches. Good introductions to Chomskian theories can be found in [116, 27]. Simplifying somewhat the presentation of van Riemsdijk and Williams [116], we can view the principal components of a GB grammar [22] as consisting of three rule systems and a set of modules which define well-formedness conditions on each of four levels of representation.

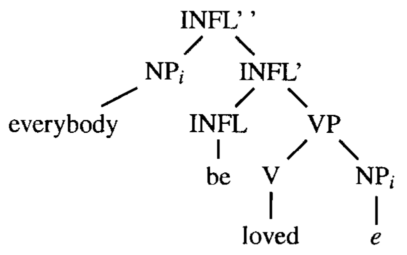
The levels of representation are given in the following list.

- D-structure (DS), which reflects the thematic structure of the utterance (basically, "who did what to whom").
- An intermediate level, called S-structure (SS), representing its surface grammatical structure and related to D-structure by the displacement of NPs from their D-structure positions.
- Logical form, (LF), in which meaning is most explicitly represented and related to S-structure by the displacement of certain phrases.
- Phonological form, (PF), where phonological properties are defined and related to S-structure by a phonological mapping.

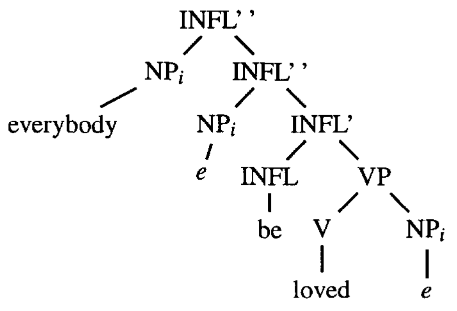
For instance, following [61], the PF representation "Everybody is loved" together with the D-structure, S-structure, and LF representations shown below might constitute a well-formed representation quadruple for English:



D-structure



S-structure



LF

Of the three rule systems that relate these levels, the one that will most occupy us is the movement rule called "move-α," which is used to obtain SS from DS and to obtain LF from SS.

The modules act as conditions on rule application, or as well-formedness conditions on representations or on rules. Some important modules are listed:

- X-bar theory, which places constraints on phrase structure at DS.
- θ-theory, which restricts the roles that arguments can play.
- Bounding theory, which restricts movements.
- Binding theory, which rules binding relations between noun phrases (overt and empty).
- Case theory, which specifies the environment where overt noun phrases may appear.
- The empty category principle, which specifies the environments where traces may appear.

Figure 6 shows the form levels or representation as nodes, the main rules generating them as labels, and the modules constraining them in boxes.

The principles of grammar, it is hypothesized, are innate and, therefore, true of any language. Language-specific values for a set of parameters, together with a specific lexicon, specialize the set of structures admitted into those of the corresponding specific language.

We shall exemplify this principle-and-parameters notion for X-bar theory. This theory generalizes head categories (such as "noun," "verb," and "adjective") into an abstract category called X, by means of which it collapses all rules describing phrases with a specific head (i.e., all noun phrases, adjective phrases, verb phrases, etc.) into basically just two rules describing X-phrases in general: the X-bar rule, which combines a head with a complement, and the XP (or X-double-bar) rule, which combines a specifier with an X-bar. This can be depicted with the following ID (i.e., where order of constituents is not specified) rules:

```
xp(Category) --> specifier(Category), x_bar(Category).
x_bar(Category) --> x(Category), complement(Category).
```

The category x refers to phrasal heads (e.g., nouns, verbs, adjectives); x_bar describes

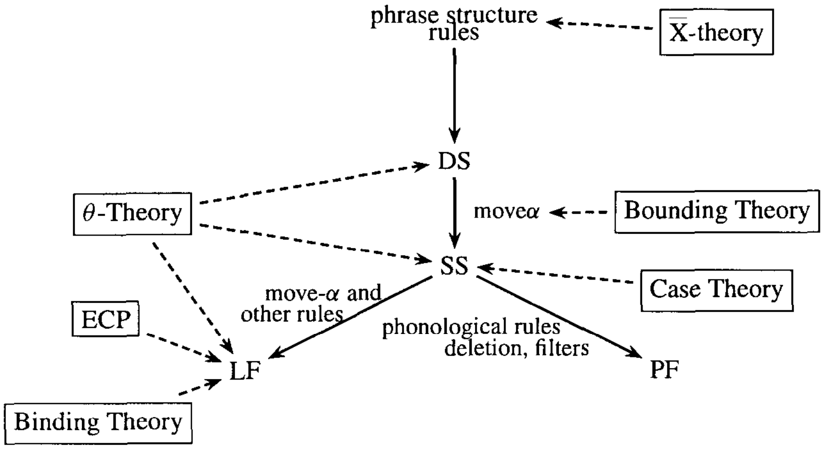


FIGURE 6. Simplified model of GB theory.

the level above the lexical one in a parse tree, and xp corresponds to the phrasal level. Nontraditional heads, such as inflection, are also represented; their phrases are covered by the same two rules. Sample specifiers are articles in the case of noun phrases, adverbs for adjective phrases, and negation for verbs.

In order to specialize these language-independent rules into English, all we have to do is set the value of the parameters "head" and "spec" to "first." For other languages, alternative settings can describe heads appearing before specifiers or after complements.

We have included this summary of one of the models of GB to provide some context for our discussion, but we shall not attempt to cover all its aspects here. We shall instead presents some insights on this theory's approach to the problem of overgeneration, which relates to the problem of long-distance dependencies which has been a leitmotif throughout this paper.

Chomskian solutions to overgeneration. In Section 4.1.1, we got a taste of how grammar additions to prevent overgeneration may induce bad side effects, which in turn require more ad hoc additions, and so on. More expressive formalisms than DCGs, such as MGs or XGs, do not solve this problem. For instance, the rules that exemplify XGs in Section 4.1.2,

```
(1) rel_marker(I), skip(X), trace(I) --> rel_pronoun(I), skip(X).

(2) relative --> rel_marker, s.

(3) np --> trace(_). ,
```

allow for the relative pronoun to be coindexed with a trace outside the relative clause, as in the incorrect sentence

(4) * the house [that$_i$ Jack built the house] costs [t$_i$].

This type of overgeneration can be circumvented, as suggested by Pereira [85], by adding extraneous symbols to "bracket" the relative clause, so that elements outside the relative clause cannot be coindexed with elements inside it. However, this bracketing technique cannot prevent other types of undesirable movement, as in:

(5) * Who$_i$ do you wonder why John likes [t$_i$]? ,

In these cases, as noted in [104], if we used bracketing to disallow movements that cross verb phrase boundaries, we would be also disallowing acceptable sentences such as

What$_i$ did they learn [t$_i$]? .

Chomsky's GB theory [22] elegantly captures all cases above, by forbidding movement exactly in those cases in which it would result in incorrect sentences. This is done through bounding theory, which restricts movements, and binding theory, which rules coreference relations between noun phrases.

For instance, a simplified version of bounding theory [104]: A moved constituent must c-command its trace, where a node α c-commands a node β if and only if α does not dominate β, but the first branching node that dominates α dominates β.

This rules out sentences such as (4) above, since in any usual syntactic analysis of it, the first branching node that dominates "that" does not dominate anything after the relative clause.

Similarly, the binding rule known as *subjacency*—no rule can relate a constituent X to constituents Y or Z in a structure of the form

...Y...[α ...[β ...X...]...]...Z...,

where α and β are "bounding nodes" (this is a language-dependent concept; in English, the bounding nodes are taken to be s and np)—rules out sentences such as (5) above,

* Who [$_S$ do you wonder why [$_S$ John likes [t$_i$]]]? .

since "Who" does c-command the trace, but it does so across two bounding nodes.

5.3.3. Logic Programming Incarnations of Chomskyan Solutions to Overgeneration. The problem of overgeneration in implementing Chomskian theories has been attacked in logic programming through a variety of resources, some of which we survey in this section, all aiming at dynamically choosing the earliest possible moment in which the principles that rule out incorrect sentences can detect their incorrectness. The earliest approaches were those of [122, 102].

There is a growing concensus toward formulating constraints at the level of S-structure itself (cf. the levels of representation described in Section 6.2) rather than explicitly computing them via derivations from D-structure to S-structure to LF. This declarative approach seems to be more computationally tractable [12]. The D-structure information, although factored into the parsers, is not actually computed. Efficiency can be further improved by interleaving structure building and constraint-and-principle application.

Thus, for instance, Johnson describes formulations in which the knowledge about a level of representation is used without necessarily constructing an explicit representation at that level, and coroutines between the principles of grammar, which ensure that all existing nodes are well-formed before constructing any new nodes [61]. Crocker [28] uses multiple metainterpreters, coroutined using the goal-freezing mechanisms of constraint logic programming. Fong [47] provides a static and dynamic ordering of principles, and interleaves them with phrase structure construction (this approach involves logic grammars as well: rules are represented as DCG rules with language parameters, while principles are represented as Prolog commands that are close to linguistic formulations, plus Prolog definitions). Stabler [105] proposes interleaving even semantic and syntactic interpretation by using a control regime that gives precedence to those statements dealing
with semantic interpretation, so that partially known structures can be interpreted early. A PARLOG implementation of GB theory was proposed in [72].

Among the approaches that use logic grammars, there are two basic approaches to incorporating principles such as c-command and subjacency:

(i) Modifying logic grammars so that these principles are automatically enforced in a standard way.
(ii) Endowing logic grammars with constraining primitives that a user can tailor to different linguistic proposals.

The first approach is exemplified by restricted logic grammars [104], which are an extension of left extraposition grammars, which automates c-command and subjacency into the grammar's specification of movement, by restricting the access to the extraposition list so the parser will allow traces only in the positions allowed by Chomsky's constraints. The equivalent RLG rule for the XG rules (1) and (2) above is

```
relative «<  trace --> rel_pronoun, s.,
```

where the skipped substring is indicated by "«<", and a similar treatment of right extraposition (in which a trace is left behind at the left after a constituent is moved to the right) can be obtained using the functor: "«<", instead. This allows for sentences such as

[The woman [t_i]] is here [that Jill saw]$_i$.

A related proposal is that of government binding logic grammars (GBLG) [18], which were also designed for a symmetric treatment of both left and right extraposition within GB theory. Their rules are DCG-like rules with single-argument grammar symbols, some of which are reserved for special GB treatment (e.g. "trace"). The direction of movement is, as in RLGs, indicated by the operators "«<" and "«< ." GBLGs are implemented through a bottom-up parser.

The second approach is exemplified by static discontinuity grammars (SDGs) [32, 33, 35, 37, 38, 5], which provide primitives to constrain movement according to different types of GB theories, and which express movement and coreference through bunches of DCG-like rules related by common substitutions and which must all apply together, albeit to discontinuous constituents (this allows for tree-like representations of derivations, while retaining the expressive power of type-0 rules).

SDG rules allow a concise implementation of the move-α principle, which basically states that any constituent is allowed to move into any empty position, modulo constraints such as expressed by binding and bounding theories.

For instance, the following SDG rule (an instance of move-α) allows relating an NP trace with an empty np position at the time that it coindexes them (trace is now noted as a feature of an np; +wh and empty are other possible features):

```
<np(J, empty) --> np(J,+wh).
np(J,+wh) --> np(J,trace).> .
```

SDG grammars also automatically build up syntactic structure and can dynamically consult the structure built so far in order to enforce linguistic constraints that are statically expressed in terms of node domination within a parse tree status.They have been used for applications implementing the linguistic theories of government binding [22] and barriers [20].

Constraining mechanisms are described in terms of calls to a few primitives, such as "dominates(N1,N2)" (with the meaning "N1 dominates N2 in the parse tree so far"), or of definitions of the form

constraint(Path, Node, Root):- body. ,

where body is any Prolog goal relevant to describing the constraint, Path describes a path in the derivation, Node describes a node in that path under which no element can move out of a given zone, and Root is the root of that zone.

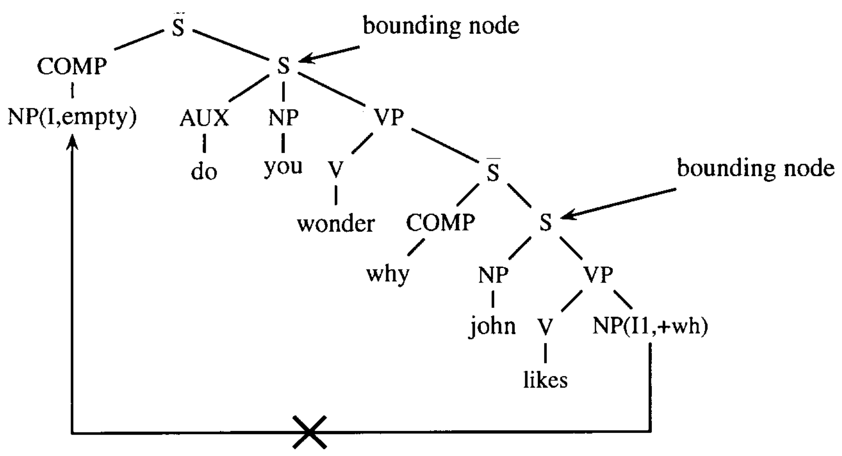


FIGURE 7. Move-α is blocked by subjacency.

Subjacency, for instance, can be described as

constraint([A...[B...]...],B,A):- bounding(A), bounding(B).

Figure 7 exemplifies: the path on which the constituent NP(I1,+wh) appears contains two bounding nodes (A=B=S), so move-α is blocked: NP(I1,+wh) is not allowed to move to (relate to) the position NP(I,empty). Thus the incorrect sentence "Who do you wonder why John likes?" is rejected.

This approach allows for manipulation of both structure and control at the user's level, while nevertheless automating most procedural concerns.

SDGs have been used in two language processing applications: an automatic generator of machine error messages and a grammar of Spanish with clitic treatment. Material related to these applications is covered in [16, 15, 32]. SDG variants for generation using GB theory have also been examined [97]. An SDG formulation of barriers has been proposed, in which D-structure and S-structure are merged [38] into a single level of representation in which all principles and constraints are applied. This results in less duplication of information, easier interaction betweeen levels (since all constraints are on the same level), and the possibility of reconstructing from it the original levels of representation.

Yet another approach is that of Stabler [107], which axiomatizes many aspects of GB theory[8]in first-order logic by using proof and transformation techniques that result in executable while elegant and transparent formulations. This transparency also facilitates the initial assessment of correctness and provides flexibility in the face of change, thus increasing the linguist's confidence in the correspondence between intentions and results. Further, Stabler shows how these techniques can result in some provably correct and complete applications.

The disadvantages of first-order logic are also discussed: it does not allow one to state that some (unspecified) relation is symmetric, or that there are only finitely many strings;

[8]Stabler's linguistic focus is actually on the more recent Chomskian theory barriers, whose definitions differ from GB ones, but which also have many points in common. A full description can be found in [20].

some facts are awkward to represent (e.g., that there are exactly 100 things); some can be represented with great computational problems (e.g., equality).[9]However, it is shown that despite these drawbacks, we can get transparent representations of GB principles with a relatively tractable equality theory.

6. APPLICATION

Some of the systems developed around specific applications have as their main goal to advance the state of knowledge, whereas others aim at carrying out practical natural language processing tasks. For the former, elegance and theoretical basis are paramount, whereas the latter are mostly concerned with coverage and efficiency—although both types of systems can exhibit, of course, both kinds of concerns to a certain extent.

The theoretically oriented systems include those which make extensive use of linguistic theory, such as Fong's [47], which correctly accounts for hundreds of different constructions from an introductory linguistics textbook [47], and those which develop new representational devices, such as Candide [86], a knowledge acquisition system for the incremental interpretation of utterances in context, which aims at accounting for the influences of context on the combinatorial aspects of interpretation, while preserving compositionality as much as possible.

Practically oriented systems include database interfaces such as [31], CHAT-80 [120], message-understanders such as PUNDIT [29, 30], and sizeable systems such as LMT, a lexicalist machine translation system with one of the widest coverage English analyzers in existence, in which transformations can be described in terms of an input tree, an output tree, and conditions [77, 10], the core language engine [4], or systems being used everyday, such as TAUM-METEO for weather forecast, which uses q-systems [23].

Many other specific applications have been explored. For instance, logic grammar aided learning of lexicons [93]; detecting grammatical mistakes of a student learning French, through a Prolog-based analyzer with mixed (bottom-up, top-down) strategy [8]; assisted sentence composition applications to language interfaces [96]; applications for communicating with handicapped persons [50]; machine translation for agricultural reports [56]; and reversible language processors (those which can be easily adapted both for analysis and for synthesis), e.g., [56, 109, 94, 37, 43]).

Let us also mention that some of the formalisms developed with computational linguistics in mind have found applications outside of it. For instance, DCSGs [110], a DCG-like formalism for free word order languages in which grammar rules are viewed as definitions for set conversions, has applications to general problem solving as well; and DCTGs [1] are being used in software specification problems.[10]

7. CONCLUDING REMARKS

We find ourselves at the exciting historical point at which the advances of logic programming make it possible to address the needs of growingly ambitious applications in natural

[9]These weaknesses in expression, which are also true of Horn clause logic, on which most logic grammar formalisms are based, are not as limiting in logic programming as they are in the full first-order logic formulation of Stabler, since the former allows us to identify a class of minimal models.

[10]James Hanlon, personal communication.

language processing with hopes of reasonable efficiency (see also [87]), and in which theoretical linguistics results are coincidentally developing in directions that are more and more compatible with computational linguistics needs.

Advances in logic programming can be applied to the use of one linguistic theory or another. For instance, the problem of coreference of long-distance constituents, as seen in Section 5.2.1, has been attacked using higher-order logics both in GPSG-like frameworks (intuitionistic, linear logics) and in Chomskian frameworks (relevance logic). From the discussion of that section and of Section 5.3.2, it should be clear that a use of these techniques which is merely *inspired* in linguistic theory is not enough: problems of over- and undergeneration immediately appear when we want to extend the linguistic coverage. Minimizing these problems requires all the interdependent threads of our chosen theory to be delicately woven.

However, linguistic theory, while having developed remarkable insights on some very complex linguistic phenomena, does not have as its goal or method to provide immediate comprehensive descriptions of actual natural language. A natural language processing system that must process actual text (say, spontaneous speech or news reports) with a minimum of coverage and accuracy must solve many problems for which linguistic theory does not have, even in principle, solutions. Moreover, when building actual language processing systems, many instances are found in which the analyses of linguistic theory are contradicted by data.

The general analyses from linguistic theory provide, as we have seen, important insights for natural language systems. While no single linguistic theory has all the answers, all of them have contributed to natural language processing, in general, and to logic programming questions, in particular. Adapting these general analyses and insights from linguistic theory into actual language processing systems with the help of recent logic programming developments is a challenging task, which is already yielding useful feedback to linguistic theory (e.g., [106], p. 259) and which is likely to continue motivating, in turn, further developments in logic programming itself.

I am indebted to Jo Calder, Andrew Fall, Eli Hagen, Chris Mellish, Paul Tarau, and the anonymous referees for their most helpful comments on this article's first draft. This research was supported by NSERC Operating Grant 31-611024 and by NSERC CSS and SFU PRG Infrastructure and Equipment grants given to the Logic and Functional Programming Laboratory at SFU, in whose facilities this work was developed. I am also grateful to the Centre for Systems Science, LCCR, and the School of Computing Sciences at Simon Fraser University for the use of their facilities.

REFERENCES

1. Abramson, H., Definite Clause Translation Grammars, in: *Proceedings of the IEEE Logic Programming Symposium*, Atlantic City, NJ, February 1984.
2. Abramson, H., and Dahl, V., *Logic Grammars*, Symbolic Computation AI Series, Springer, New York, 1989.
3. Ait-Kaci, H., and Nasr, R., Login: A Logical Programming Language with Built-In Inheritance, *J. Logic Program.* 3:187–215 (1986).
4. Alshawi, H., *The Core Language Engine*, MIT Press, Cambridge, MA, 1992.
5. Andrews, J., Dahl, V., and Popowich, F., A Relevance Logic Characterization of Static Discontinuity Grammars, in: *Workshop on Linear Logic and Logic Programming*, 1992.
6. Balari, S., and Varile, G. B., CLG(n): Constraint Logic Grammars, in: *Proceedings of the International Conference on Computational Linguistics*, 1990, pp. 1–6.

7. Bancilhon, F., Maier, D., Sagiv, Y., and Ullman, J., Magic-Sets and Other Strange Ways to Implement Logic Programs, in: *Proceedings of the Fifth ACM Symposium on Principles of Database Systems*, 1986.
8. Barchan, J., Woodmansee, B., and Yazdani, M., *A Prolog-Based Tool for French Grammar Analysis*, Instructional Science 15, 1985.
9. Barwise, J., and Perry, J., *Situations and Attitudes*, MIT Press, Cambridge, MA, 1983.
10. Bernth, A., and McCord, M., LMT for Danish-English Machine Translation, in: C. Brown and G. Koch, (eds.), *Natural Language Understanding and Logic Programming, III*, North-Holland, Amsterdam, 1991, pp. 179–194.
11. Berwick, R. C., Principle-Based Parsing. in: *The Processing of Linguistic Structure*, MIT Press, Cambridge, MA, 1987.
12. Berwick, R. C., Abney, S. P., and Tenny, C., (eds.), *Principle-Based Parsing: Computation and Psycholinguistics*, Kluwer Academic Publishers, Dordrecht, 1991.
13. Billot, S., and Lang, B., The Structure of Shared Forests in Ambiguous Parsing, in: *27th Annual Meeting of the Association for Computational Linguistics*, 1989, pp. 143–151.
14. Bresnan, J., *The Mental Representation of Grammatical Relations*, MIT Press, Cambridge, MA, 1982.
15. Brown, C., *Generating Spanish Clitics Using Static Discontinuity Grammar*, Ph.D. Thesis, School of Computing Sciences, Simon Fraser University, 1987.
16. Brown, C., Dahl, V., Massam, D., Boyer, M., and Pattabhiraman, T., Tailoring GB Theory for a Useful Subset of Error Messages, Technical Report LCCR 86-14, Simon Fraser University, Burnaby. B.C., 1986.
17. Carpenter, B., *The Logic of Typed Feature Structures with Applications to Unification Grammars, Logic Programs and Constraint Resolution*, Cambridge University Press, 1992.
18. Chen, H.-H., Lin, I.-P., and Wu, C.-P., A New Design of Prolog-Based Bottom-Up Parsing System with Government-Binding Theory, in: *International Conference on Computational Linguistics 88*, 1988, pp. 112–116.
19. Chen, W., and Warren, D. S., C-Logic of Complex Objects, Technical Report, Department of Computer Science, State University of New York at Stony Brook, 1989.
20. Chomsky, N., *Barriers*, MIT Press, Cambridge, MA, 1986.
21. Chomsky, N., *Syntactic Structures*, Mouton, The Hague, 1957.
22. Chomsky, N., *Lectures on Government and Binding*, Foris Publications, Dordrecht, Holland, 1981.
23. Colmerauer, A., Les Systemes-q ou un Formalisme pour Analyser et Synthétiser des Phrases sur Ordinateur, Technical Report 43, Département d'Informatique, Université de Montreal, Québec, 1970.
24. Colmerauer, A., Metamorphosis Grammars, in: L. Bolc (ed.), *Natural Language Communication with Computers*, Springer, New York, 1978, pp. 133–187.
25. Colmerauer, A., Kanoui, H., Pasero, R., and Roussel, P. Un Système de Communication Homme-Machine en Français, Technical Report, Groupe d'Intelligence Artificielle, Université d'Aix-Marseille II, Marseille, 1973.
26. Covington, M., *Natural Language Processing for Prolog Programmers*, Prentice-Hall, Englewood Cliffs, NJ, 1994.
27. Cowper, E. A., *A Concise Introduction to Syntactic Theory*, University of Chicago Press, 1992.
28. Crocker, M., A Principle-Based System for Syntactic Analysis, *Canadian J. Linguistics* 36(1):1–26 (1991).
29. Dahl, D., Palmer, M. S., and Passonneau, R. J., Focussing and Reference Resolution in Pundit, in: *Proceedings of AAAI-86*, 1986, pp. 1083–1088.
30. Dahl, D., Palmer, M. S., and Passonneau, R. J., Nominalizations in Pundit, in: *Proceedings of 25th Annual Meeting of the Association for Computational Linguistics*, 1987, pp. 131–139.

31. Dahl, V., Translating Spanish into Logic through Logic, *Computational Linguistics* 7(3):149–164 (1981).
32. Dahl, V., Representing Linguistic Knowledge through Logic Programming, in: *Proceedings of the Fifth International Conference/Symposium on Logic Programming*, Seattle, August 1988.
33. Dahl, V., Discontinuous Grammars, *Computational Intelligence* 5(4):161–179 (1989).
34. Dahl, V., Incomplete Types for Logic Databases, *Appl. Math. Lett.*, 4(3):25–28 (1991).
35. Dahl, V., and Massicotte, P., Processing Techniques for Discontinuous Grammars, in: H. E. A. Abramson (ed.), *Meta-Programming for Logic Programming*, MIT Press, Cambridge, MA, 1988, pp. 141–156.
36. Dahl, V., and McCord, M. C., Treating Coordination in Logic Grammars, *Amer. J. Computational Linguistics* 7(3):149–164 (1983).
37. Dahl, V., and Popowich, F., Parsing and Generation with Static Discontinuity Grammars, *New Generation Computing* 8(3):245–274 (1990).
38. Dahl, V., Popowich, F., and Rochemont, M., A Principled Characterization of Dislocated Phrases: Capturing Barriers with Static Discontinuity Grammars, *Linguistics and Philosophy* 16:331–352 (1993).
39. Damas, L., and Varile, G. B., CLG: A Grammar Formalism Based on Constraint Resolution, in: *Proceedings of the EPIA 1989*,E. M. Morgado and J. P. Martins (eds.), *Lecture Notes in Artificial Intelligence 390*, Springer, New York, 1989.
40. De Groot, D., Restricted And-Parallelism, in: *Proceedings of the International Conference on Fifth Generation Computer Systems*, 1984.
41. Dörre, J., and Eisele, A., A Comprehensive Unification Based Grammar Formalism, Dyana Report R3.1.B, Centre for Cognitive Science, Edinburgh University, 1991.
42. Dowty, D. R., Wall, R. E., and Peters, S., *Introduction to Montague Semantics*, D. Reidel Publishing Company, Dordrecht, 1981.
43. Dymetman, M., Transformation de Grammaires Logiques et Reversibilité en Traduction Automatique, Technical Report, Université Joseph Fourier (Grenoble 1) and CNRS, 1992 (Thèse d'état).
44. Earley, J., An Efficient Context-Free Parsing Algorithm, *Commun. ACM* 13(2):94–102 (1970).
45. Eisele, A., and Dörre, J., Unification of Disjunctive Feature Descriptions, in: *Proceedings of the 1988 Conference of the Association for Computational Linguistics*, 1988, pp. 286–294.
46. Emele, M., and Zajac, R., Typed Unification Grammars, in: *Proceedings of the 13th International Conference on Computational Linguistics*, 1990, vol. 3, pp. 293–298.
47. Fong, S., Computational Properties of Principle-Based Grammatical Theories, Ph.D. Thesis, MIT AI Lab, 1991.
48. Gazdar, G., Phrase Structure Grammar, in: P. Jacobson and G. K. Pullum (eds.), The Nature of Syntactic Representation, D. Reidel, Dordrecht, Holland, 1982.
49. Gazdar, G., Klein, E., Pullum, G., and Sag, I., *Generalized Phrase Structure Grammar*, Harvard University Press, Cambridge, MA, 1985.
50. Guenthner, F., Pasero, R., and Sabatier, P., Communicating with My Handicapped Friends, Technical Report, Groupe d'Intelligence Artificielle, Aix-Marseille II University, 1993.
51. Gunji, T., *Japanese Phrase Structure Grammar*, D. Reidel Publishing Co., Dordrecht, 1986.
52. Hermenegildo, M., and Greene, K., Prolog and Its Performance: Exploiting Independent And Parallelism, in: *Proceedings of the Seventh International Conference on Logic Programming*, 1990, pp. 253–268.
53. Hodas, J., Specifying Filler-Gap Dependency Parsers in a Linear Logic Programming Language, in: K. Apt (ed.), *Proceedings of the 1992 Joint International Conference and Symposium on Logic Programming*, MIT Press, Cambridge, MA, 1992, pp. 622–636.

54. Hodas, J., and Miller, D., Logic Programming in a Fragment of Intuitionistic Linear Logic, *J. Inform. Computation*, 1992.
55. Hopcroft, J., and Ullman, J., *Introduction to Automata Theory, Languages and Computation*, Addison-Wesley, Reading, MA, 1979.
56. Isabelle, P. M. D., Critter: A Translation System for Agricultural Market Reports, in: *International Conference on Computational Linguistics*, Budapest, 1988, vol. 1, pp. 261–266.
57. Ishikawa, A., and Akama, S., A Semantic Interface for Logic Grammars and Its Application to DRT, in: C. Brown and G. Koch (ed.), *Natural Language Understanding and Logic Programming, III*, North-Holland, Amsterdam, 1991, pp. 281–292.
58. Jaffar, J., and Lassez, J.-L., Constraint Logic Programming, in: *Proceedings of the 14th ACM Symposium on Principles of Programming Languages*, Munich, 1987, pp. 111–119.
59. Johnson, M., Memoization in Constraint Logic Programming, Technical Report, Brown University, 1983.
60. Johnson, M., Attribute-Value Logic and the Theory of Grammar, CSLI Lecture Notes 14, Center for the Study of Language and Information, Stanford, 1988.
61. Johnson, M., Deductive Parsing: The Use of Knowledge of Language, in: R. C. Berwick, S. P. Abney, and C. Tenny (eds.), *Principle-Based Parsing: Computation and Psycholinguistics*, Kluwer Academic Publishers, Dordrecht, 1991, pp. 39–64.
62. Johnson, M., Constraints and the Chart, unpublished manuscript.
63. Kamp, H., A Theory of Truth and Semantic Representation, in: J. Groenendijk, T. M. V. Janssen, and J. Stokhof (eds.), *Formal Methods in the Study of Language*, Mathematical Centre Tracts, Amsterdam, 1981, pp. 277–322.
64. Kaplan, R., and Bresnan, J., Lexical Functional Grammar: A Formal System for Grammatical Representation, in: J. Bresnan (ed.), *The Mental Representation of Grammatical Relations*, MIT Press, Cambridge, MA 1982.
65. Kaplan, R. M., and Maxwell, J. T., An Algorithm for Functional Uncertainty, in: *Proceedings of the 1988 Conference of the Association for Computational Linguistics*, 1988, pp. 297–302.
66. Karttunen, L., Features and Values, in: *XX Annual Meeting of the Association for Computational Linguistics and Xth International Conference on Computational Linguistics*, California, 1984.
67. Kasper, R. T., Conditional Descriptions in Functional Unification Grammar, in: *Proceedings of the 25th Annual Meeting of the Association for Computational Linguistics*, 1987, pp. 233–240.
68. Kasper, R. T., A Unification Method for Disjunctive Feature Descriptions, in: *Proceedings of the 26th Annual Meeting of the Association for Computational Linguistics*, 1988, pp. 235–242.
69. Kay, M., Experiments with a Powerful Parser, in: *Deuxieme Conference International sur le Traitement Automatique des Langages*, Grenoble, France, 1967.
70. Kay, M., *Parsing in Functional Unification Grammar*, Cambridge University Press, 1985, pp. 251–278.
71. Kay, M., *Algorithm Schemata and Data Structures in Syntactic Processing. Readings in Natural Language Processing*, Morgan Kaufmann, Los Altos, CA, 1986.
72. Khuns, R. J., A PARLOG Implementation of Government-Binding Theory, in: *Proceedings of the International Conference on Computational Linguistics-90*, 1990, pp. 394–399.
73. Lambek, J., The Mathematics of Sentence Structure, *Amer. Math. Monthly* 65:154–169 (1958).
74. Lambek, J., Multicategories Revisited, *Categories Comput. Sci. Contemporary Math.* 92:217–239 (1987).
75. Lang, B., Complete Evaluation of Horn Clauses: An Automata Theoretic Approach. Rapport de Recherche 913, INRIA, Rocquencourt, France, 1988.
76. McCord, M. C., Focalizers, the Scoping Problem, and Semantic Interpretation Rules in

Logic Grammars, in: D. H. D. Warren and M. van Caneghem (eds.), *Logic Programming and its Applications*, Academic Press, New York, 1981.

77. McCord, M. C., Design of LMT: A Prolog-Based Machine Translation System, *Computational Linguistics* 15:33–52 (1989).
78. Mellish, C. S., The Description Identification Problem, *Artificial Intelligence* 52(2):151–168 (1991).
79. Mellish, C. S., Implementing Systemic Classification via Unification, *Computational Linguistics* 14(1):40–51 (1988).
80. Miller, D., and Nadathur, G., Some Uses of Higher-Order Logic in Computational Linguistics, in: *Proceedings of the 24th Annual Meeting of the Association for Computational Linguistics*, 1986, pp. 247–255.
81. Moens, M., Calder, J., Klein, E., Reape, M., and Zeevat, H., Expressing Generalizations in Unification-Based Grammar Formalisms, in: *European Association for Computational Linguistics 89*, 1989, pp. 174–181.
82. Moortgat, M., Categorial Investigations: Logical and Linguistic Aspects of the Lambek Calculus, Ph.D. Thesis, University of Amsterdam, Amsterdam, The Netherlands, October 1988.
83. Oehrle, R. T., Bach, E., and Wheeler, D., (eds.), *Categorial Grammars and Natural Language Structure*, Reidel, Dordrecht, The Netherlands, 1988.
84. Pareschi, R., and Miller, D., Extending Definite Clause Grammars with Scoping Constructs, in: D. H. D. Warren and P. Szeredi (eds.), *International Conference in Logic Programming*, MIT Press, Cambridge, MA, 1990, pp. 373–389.
85. Pereira, F. C. N., Extraposition Grammars, *Amer. J. Computational Linguistics* 7:243–256 (1981).
86. Pereira, F. C. N., and Pollack, M., Incremental Interpretation, *Artificial Intelligence* 50(1):37–82 (1991).
87. Pereira, F. C. N., Prolog and Natural Language Analysis: Into the Third Decade, in: S. Debray and M. Hermenegildo (eds.), *Proceedings of the 1990 North American Conference on Logic Programming*, 1990, pp. 813–832.
88. Pereira, F. C. N., Semantic Interpretation as Higher-Order Deduction, *Proceedings of the Second European Workshop on Logics and AI*, Springer, New York, 1990.
89. Pereira, F. C. N., and Shieber, S. M., Prolog and Natural Language Analysis, CSLI Lecture Notes 10, Centre for the Study of Language and Information, Stanford, 1987.
90. Pereira, F. C. N., and Warren, D. H. D., Parsing as Deduction, in: *Proceedings of the 21st Annual Meeting of the Association for Computational Linguistics*, 1983.
91. Pereira, F. C. N., and Warren, D. H. D., Definite Clause Grammars for Language Analysis—A Survey of the Formalism and a Comparison with Augmented Transition Networks, *Artificial Intelligence* 13:231–278 (1980).
92. Pollard, C., and Sag, I., Information-Based Syntax and Semantics, Technical Report, CSLI Notes 13, Center for the Study of Language and Information, Stanford, 1987.
93. Rayner, M., Hugosson, A., and Hagert, G., Using a Logic Grammar to Learn a Lexicon, in: *Proceedings of the Conference on Computational Linguistics*, Budapest, 1988, vol. 1, pp. 124–129.
94. Rincel, R., and Sabatier, P., Using the Same System for Analysing and for Synthesizing Sentences, in: *Proceedings of the 13th International Conference on Computational Linguistics*, 1990, pp. 440–442.
95. Rosenschein, S. J., and Shieber, S. M., Translating English into Logical Form, in: *Proceedings of the 20th Annual Meeting for Computational Linguistics*, University of Toronto, Toronto, Ontario, 1982, pp. 1–8.
96. Sabatier, P., Interfaces en Langage Naturel: Du Traitement du non Attendu a la Composition de Phrases Assistee, *Ann. Telecommunications* 44(1-2):77–84 (1989).
97. Saint-Dizier, P., Contextual Discontinuous Grammars, in: *Natural Language Understanding and Logic Programming II*, North-Holland, Amsterdam, 1988.

98. Saussure, F. D., *Course in General Linguistics*, McGraw-Hill, New York, 1959. Originally published in French in 1919.
99. Schatz, U., and Lehner, C., Implementation of the Comprehensive Unification Formalism, Technical Report 90-31, University of Munich, 1990.
100. Seki, H., On the Power of Alexander Templates, in: *Proceedings of the Eighth ACM Symposium on Principles of Database Systems*, 1989.
101. Sells, P., Lectures on Contemporary Syntactic Theories, CSLI Lecture Notes 3, Centre for the Study of Language and Information, Stanford, 1985.
102. Sharp, R., A Model of Grammar Based on Principles of Government and Binding, Master's Thesis, University of British Columbia, 1985.
103. Shieber, S., *Constraint-Based Grammar Formalisms—Parsing and Type Inference for Natural and Computer Languages*, A Bradford Book: MIT Press, Cambridge, MA, 1992.
104. Stabler, E. J., Restricting Logic Grammars with Government-Binding Theory, *J. Computational Linguistics* 13(1–2), pp. 1–10 (1987).
105. Stabler, E. J., Avoiding the Pedestrian's Paradox, in: R. Berwick, S. P. Abney, and C. Tenny (eds.), *Principle-Based Parsing: Computation and Psycholinguistics*, Kluwer Academic Publishers, Dordrecht, 1991, pp. 199–238.
106. Stabler, E. J., Implementing Government-Binding Theories, in: R. Levine (ed.), *Formal Grammar: Theory and Implementation*, Vancouver Studies in Cognitive Science, Oxford University Press, 1992, pp. 243–275.
107. Stabler, E. J., *The Logical Approach to Syntax*, ACL-MIT Press Series in Natural Language Processing, MIT Press, Cambridge, MA, 1992.
108. Steedman, M., Dependency and Coordination in the Grammar of Dutch and English, *Language* 61:523–568 (1985).
109. Strzalkowski, T., Automated Inversion of Logic Grammars for Generation, in: *Proceedings of the 28th Meeting of the ACL*, Pittsburgh, PA, 1990.
110. Tanaka, T., Definite-Clause Set Grammars: A Formalism for Problem Solving, *J. Logic Program.* 10(1):1–18 (1991).
111. Tuda, H., Hasida, K., and Sirai, H., JPSG Parser on Constraint Logic Programming, in: *Proceedings of the European Chapter of the Association for Computational Linguistics*, 1989, pp. 95–102.
112. Uehara, K., Ochitani, R., Kashusho, O., and Toyoda, J., A Bottom-Up Parser Based on Predicate Logic: A Survey of the Formalism and its Implementation Technique, in: *Proceedings of the 1984 International Symposium on Logic Programming*, 1984.
113. Uszkoreit, H., Categorial Unification Grammars, in: *Proceedings of the International Conference on Computational Linguistics 86*, 1986, pp. 187–194.
114. van Benthem, J., The Lambek Calculus, in: *Categorial Grammars and Natural Language Structure*, Reidel, Dordrecht, The Netherlands, 1988.
115. van Benthem, J., *Language in Action—Categories, Lambdas and Dynamic Logic, Studies in Logic and the Foundations of Mathematics* 130, North-Holland, Amsterdam, The Netherlands, 1991.
116. van Riemsdijk, H., and Williams, E., *Introduction to the Theory of Grammar*, MIT Press, Cambridge, MA, 1986.
117. Villemonte de la Clergerie, E., Layer Sharing: An Improved Structure-Sharing Framework, in: *ACM Symposium on the Principles of Programming Languages*, Association for Computing Machinery, New York, 1993, pp. 345–358.
118. Warren, D. H. D., Earley Deduction, unpublished note, 1975.
119. Warren, D. H. D., The SRI Model for OR-Parallel Execution of Prolog, Abstract Design and Implementation Issues, in: *Proceedings of the Fourth Symposium on Logic Programming*, San Francisco, 1987, pp. 46–53.
120. Warren, D. H. D., and Pereira, F. C. N., An Efficient Easily Adaptable System for Interpreting Natural Language Queries, Technical Report, Dept of Artificial Intelligence, University of Edinburgh, 1982.

121. Warren, D. S., Memoing for Logic Programs, *Commun. ACM*, 35(3):94–111 (1992).
122. Wherli, E., A Modular Parser for French, in: *Proceedings of the 8th International Joint Conference on Artificial Intelligence*, Karlsruhe, 1983, pp. 686–689.
123. Woods, W., Transition Network Grammars for Natural Language Analysis, *Commun. ACM* 13:591–596 (1970).
124. Woods, W., An Experimental Parsing System for Transition Network Grammars, in: R. Rustin (ed.), *Natural Language Processing*, Algorithmic Press, New York, 1973, pp. 145–149.
125. Matsumoto, Y., Tanaka, H., Hirakawa, H., Miyoshi, H., and Yasukawa, H., BUP— A Bottom-Up Parser Embedded in Prolog, *New Generation Computing* 1:145–158 (1983).
126. Zeevat, H., Klein, E., and Calder, J., Unification Categorial Grammar, *Lingua e Stile* 26(4):499–527 (1991).