An Environment for Experimentation with Parsing Strategies*

Gregor Erbach
Universitat des Saarlandes
FR 8.7 Allgemeine Linguistik - Computerlinguistik
W-6600 Saarbrucken, FRG
e-mail: erbach@coli.uni-sb.de

Abstract

An environment for the experimentation with parsing strategies is presented which consists of a parser which can process arbitrary parsing strategies, a functional language for the definition of strategies, and a statistical component which helps the user assess the effects of the different strategies.

1. Introduction

1.1. Motivation

The past decade has witnessed the emergence and widespread acceptance of declarative grammar formalisms, some well-known exemplars being Definite Clause Grammar [Pereira and Warren 1980], Generalized Phrase Structure Grammar fGazdar et al. 1985], and Head-driven Phrase Structure Grammar (Pollard and Sag 1987],

In contradistinction to their more procedural predecessors like Transformational Grammar [Chomsky 1965] and Augmented Transition Networks [Woods 1970], today's declarative grammar formalisms do not prescribe an order in which the possible operations on the grammar are to be carried out[1].

What both today's declarative grammars and the procedural grammars have in common is that they *give* all the sentences generated by the grammar equal status, and do not account for degrees of acceptability.

Consequently, many current natural language systems have parsers that enumerate all analyses of a given string, but have no way of preferring one analysis of ambiguous sentences. The choice of a path in the search space is generally accidental, and it is not possible to formulate a parsing strategy explicitely.

This situation is not very satisfactory both from the psycholinguistic and from the engineering point of view.

From the psycholinguistic perspective, one property which needs to be modeled and explained is that humans in general consider only one reading of an ambiguous sentence. Another property is the robustness of human sentence processing when presented with ill-formed input.

From the engineering perspective, it would rather be desirable to integrate syntactic preferences and semantic processing into the parsing process for early disambiguation in order to avoid the cost of exploring the complete search space, and the knowledge processing needed for selecting one reading of an ambiguous sentence.

1.2. What is a Parsing Strategy

A parsing strategy determines the behaviour the parser in case of non-determinsm [Kay 1980]. Such non-determinism may arise by the choice of a rule to apply, and the choice of linguistic objects to which the rule is applied[2]. The application of a rule to linguistic objects is a parsing task.

In the literature, the term "parsing strategy" is used in three different senses:

a) Avoiding useless parsing tasks

A parsing strategy is a rule-selection strategy that avoids any structure building which does not contribute to the final parse result. Such strategies would be top-down parsing, which makes sure that the only rules are chosen which may produce a parse with category S, bottom-up parsing which ensures that only rules are chosen which are licensed by the words in the input string, the directed parsing methods [Kay 1980, Wir6n 1987] which combine the merits of top-down and bottom-up parsing. Head-driven parsing also falls into this category.

This work was supported by IBM Germany's LILOG Project.

[1] Definite Clause Grammars are an exception only if they are interpreted by the standard Prolog proof procedure. For alternative processing regimes see [Pereira and Shieber 1987]

[2] This is the case for rule-based grammars. For principle-based grammars like Government-Binding Theory [Chomsky 1981] or HPSG [Pollard and Sag 1987], a parsing strategy might specify which principle to apply first and to which linguistic object.

b) Best-fim parsing (heurjsticaiiy guided search)

A parsing strategy is a heuristically guided search strategy, [Kay 1980]. With such a strategy, more promising parsing tasks are preferred over less promising ones. A best-first parsing strategy prefers one of several parsing tasks, but gives the results of the successful parsing tasks equal status.

c) Ambiguity resolution (and degrees of acceptability)

A parsing strategy provides a preference for one analysis for ambiguous (sub)strings. It may well be that this preference follows from the execution of a best-first strategy defined without reference to preferences for alternative analyses. On the other hand, a best-first parsing strategy may rely on these preferences for its choice of the best parsing task. We return to this issue in sections 3.2 and 3.3.

A parsing strategy is defined by a function which assigns a priority to each parsing task. While the output of the function is a numerical priority value, little is known about the input arguments to the function.

Haugeneder and Gehrke [1988] propose a model where the user can assign different weights to eight factors (initial priority of the rule, initial priority for different readings of the lexemes, complexity of the structure, scoring of word hypotheses for spoken input, priority of an active edge, span of active edge, span of inactive edge, number of words left for processing). We are not that committed to the input of the heuristic function, and allow considerable more flexibility. In our view, finding appropriate input arguments to the function is one central objective of reasearch on parsing strategies.

1.3. Development of parsing strategies as an experimental process

Like [Haugeneder and Gehrke 1988], we view the discovery of parsing strategies as a largely experimental process of incremental optimization. Each cycle in the development consists of the following steps:

a) definition (or modification) of a parsing strategy
b) parsing of example sentences
c) analysis of the parser's behaviour

The third step should not only indicate whether the desired behaviour has been achieved, but also help to locate sources of inefficiency. These three steps must be supported by the following features of the parser:

a) A language for the definition of parsing strategies
b) The ability to process different strategies
c) Statistics and diagnostic tools for evaluation of its behaviour with respect to a particular parsing strategy

2. Implementation of Parsing Strategies

The system presented here allows the user to define parsing strategies for declarative grammars in a declarative fashion by writing an priority assignment function which gives a priority to a given parsing task.

2.1. The Parser[3]

The parser is a bottom-up[4] active chart parser. The essential data structure of the parser is the agenda [Kay 1980], a list of pairs of parsing tasks and associated priorities.

The parser can process grammars encoded in the unification grammar formalism STUF [Bouma et al. 1988]. Because unification of feature structures is computationally expensive, parsing tasks correspond to unifications, namely the unification of an item with a rule to produce an active item (or a passive item in the case of unary rules), or the combination of an active and a passive item:

- apply rule (Rule-Name,Item)
- a and p(Active-Itern,Passive-Item)

The parser can also be run without producing active items; in this case, a rule is applied to as many items as the right-hand side of the rule has elements, corresponding to the parsing task apply_rule (Rule-Name, Item}, . . ., $Item_n$).

Since both rules and linguistic objects (chart items) are involved in a parsing task, the two kinds of non-determinism mentioned in section 1.2 arc present in the choice of a parsing task[5]: the choice of a rule, and the choice of linguistic objects to apply the rule to.

The top level of the parsing algorithm is very simple:

```
while agenda not empty, and not success do
1. remove the task with the highest priority
   from the agenda
2. execute that parsing task
3. generate new parsing tasks
4. assign each new parsing task a priority and
   add it to the agenda
end while
```

A parsing strategy is defined by writing a priority-assignment function which is used in step 4 of the parsing algorithm.

Without any particular priority assignment function, the above is an algorithm schema [Kay 1980], because the choice of a parsing task from the agenda is undetermined.

2.2. Specification of Parsing Strategies

[3]The parser is implemented in Quintus Prolog and integrated in LILOG's linguistic development environment LEU/2.

[4]It is a bottom-up parser because some of the grammars have no context-free skeleton to guide top-down analysis.

[5]There is another kind of non-determinism not accounted for here. It arises when the linguistic objects which are manipulated contain disjunctions. From a theoretical point of view, all disjunctions can be brought into disjunctive normal form, and each of the disjuncts can be treated as a linguistic object.

"If there are several tasks with the same priority, the one that was most recently generated will be used.

2.2.1. A Functional Language for the Specification of Parsing Strategies

We provide a restricted language, in which parsing strategies arc specified, containing the following primitive functions:

rule(TASK) :	the grammar rule of a parsing task
item{N TASK[7]) :	the n-th item of a parsing task
*is (I*TEM) :	the feature structure of an item
path (PATH FS) :	value of a path in a feature structure
coreferent(PATHI PATH2 FS):	returns 1 if PATHI and PATH2 in the feature structure FS are coreferent, 0 otherwise
resulting-item(TASK[8]):	the resulting item of a parsing task
j h s (RULE):	the left-hand side of a rule
r h s (RULE):	the right-hand side of a rule (a list)
initial-priority(RULE):	the initial priority of a rule
start. inq-vertex (ITEM) :	the starting vertex of an item
ending-vertex (ITEM) :	the ending vertex of an item
remainder(ITEM):	a list of feature structures, if the item is active, the empty list if the item is passive
daughters (ITEM) :	a list of daughters, or 'lex' if the item is lexical
acceptability(ITEM):	the acceptability assigned to an item (see section 3.2)
cpu-1imo () :	a constantly increasing value

In addition, the usual functions for the arithmetic operations and comparisons, list manipulation (first, rest, eons, length), and truth functions (and, or, if, if-then-else, not, equal) are provided.

The user can define more complex functions from these primitive functions. Some examples are given below (variables are designated by upper-case letters):

```
span(ITEM) =
   ending-vertex(ITEM) - starting-vertex(ITEM)

category(ITEM)    path([syn cat] fs(ITEM))

min(X Y) = if( <(X Y) X Y)

complexity(ITEM) -
   if( equal (daughters(ITEM) lex )
       ]
       complex(daughters (ITEM)) + 1 )

complex(ITEMLIST) =
   if( equal( ITEMLIST () )
       0
       f (complexity(first (ITEMLIST) )
          complex(rest(ITEMLIST))))
```

In particular, the function priority (STRATEGY TASK) can be defined, which will then be used as the priority assignment function for the parsing strategy given as the first argument.

Note that the function acceptability (ITEM) introduces almost unlimited power because of the syntactic, semantic and pragmatic factors enter into the determination of the degree of acceptability of a linguistic object

2.2.2. Some examples

Some priority assignment functions are given in the following. These simple parsing strategies merely serve to illustrate the flexibility of the mechanism.

depth-first (depth): Every new parsing task gets higher priority than the other parsing tasks still on the agenda, i. e. the agenda behaves as a stack. In practice, this can be done by using some constandy increasing value as the priority of the task.

```
priority(depth TASK) = cpu-time()
```

right-to-leftfrightleft). Use the starting vertex of the resulting item as the priority of the parsing task.

```
priority (rightleft TASK) =
     starting-vertex(resultinq-item(TASK))
```

prefer long items over short ones (longitem); Use the span of the resulting item as the priority.

```
priority(1ongitem TASK) =
     span(resulting-item(TASK))
```

prefer long rules (longmie)

```
priority (longrule TASK) =
    length(rhs(rule(TASK) ) )
```

combine active and passive items before applying rules (ap)

```
priority(ap TASK) = equal(TASK a_and_p(A P))
```

The priority is 1 if the task is the combination of an active and a passive item, and 0 otherwise.

sort new to front (new-to-front); This general strategy [Haugeneder and Gehrke 88] ensures that all new tasks are added to the front of the agenda, i. e. that they have higher priority than any other tasks already on the agenda. However, an order may be imposed upon the new tasks. In order to achieve this behaviour one ensures that the priorities for the

[7] 'An item is a object consisting of starting vertex, ending vertex, feature structure, local tree (rule name and list of daughter items), remainder of an active item, acceptability rating, and a unique identifier with which the item is referred to in the list of daughters and in the parsing task. If the identifier is given as input to a function, the function is applied to the corresponding item.

[8] For resulting items, only information about starting and ending vertex, and the local tree is available.

new tasks lie in the interval between the current and the next item number. Since item numbers are incremented by 1, the priority must be in the interval [itemcount itemcount+1]. The heuristic function, which is defined using the primitive functions listed above, has a range between 0 and 0.99.

```
priority(new-to-front TASK) =
      itemcount( ) + heuristic-function(TASK)
```

2.3. Reduction of the Search Space

In the parsing process, many parsing tasks are generated. Whenever a new passive item P is added, the following parsing tasks are generated:

- for every rule R a task apply rule (R,P) and
- for every active item A whose ending vertex is the starting vertex of Pa task a__and P(A,P).

Whenever an active item A is added:

- for every passive item P whose starting vertex is the ending vertex of the A a task a_and_p (A, P).

However, not all parsing tasks do succeed - in fact, most of them fail. But, assigning a priority to a parsing task which is going to fail anyway is a waste of effort.

We have implemented a computationally inexpensive filter which eliminates most of the useless parsing tasks. The filter uses only a subset of the information present in the feature structures of the rules, and encodes this as a Prolog term. If the unification of the Prolog terms involved in the parsing task fails, the parsing task cannot succeed. The parsing tasks which pass the filter are assigned a priority and added to the agenda.

The use of a filter resulted in a reduction of the total parse time of 60 to 70 percent.

2.4. Statistical Information

We assume that the definition of parsing strategies is an experimental process of incremental optimization. In order to obtain information about the behavior of a parsing strategy, the following statistics are collected during the parsing process.

A1: cpu-time until first result is found

A2: cpu-time after complete exploration of the search space

B: number of possible parsing tasks (i. e. the total search space)

C: number of parsing tasks on the agenda (i. e. the reduced search space)

DI: number of successful parsing tasks (i. e. number of chart items) after finding the first parse

D2: number of successful parsing tasks (i. e. number of chart items) after complete exploration of the search space

EI: number of parsing tasks which contribute to the first result (i. e. number of nodes in the result tree(s) including active items)

E2: number of parsing tasks which contribte to all results (i. c. number of distinct nodes in all result trees including active items)

E3: number of parsing tasks which contribute to the correct reading (i. e. number of nodes in the chosen result tree). There must be feedback from further processing steps about which reading for an ambiguous sentence was the correct one.

The values obtained by counting parsing tasks (B, C, D and E) are also available for each rule of the grammar. B, C, D2 and E2, which involve exhaustive search, are independent of a particular parsing strategy. They exhibit global properties of the grammar, and are useful in the determination of parsing strategies.

These figures are available after one parse, and may also be summed up over a number of parses. The ratio between these figures which is the basis for the development of parsing strategies.

Time efficiency: The ratio A1/A2 (CPU-time after first parse / CPU-time after exhaustive search) indicates the time efficiency of the chosen parsing strategy. A value of 0.6 would indicate that finding the first parse with that parsing strategy takes 60% of the time which is needed for finding all parses. A value that is greater than 1 indicates that the time needed for assigning priorities to parsing tasks is greater than the time saved by using the parsing strategy.

Successful possible tasks: The ratio D2/B indicates which proportion of all parsing tasks is successful.

Successful tasks on agenda: The ratio D2/C indicates which proportion of the parsing tasks on the agenda are ultimately successful. Ideally, this ratio should be equal to 1. If this value is very low for a particular rule, means that it is a rule which passes the filter (cf. section 2.3), but is frequently unsuccessful. In this case, the filter should be improved. If this is not possible with reasonable effort, the rule should be assigned low priority.

Space efficiency (for a strategy): The ratio D1/D2 (chart items after first parse / chart items after exhaustive search) gives an indication of how much space for storing chart items is saved by the parsing strategy. This can be an very important issue if large structures are built for each item, as is the case with feature-value grammars.

Useless items (for a Strategy); The ratio E1/DI (used items / built items) indicates how many of the successful tasks are used in the first final result with a particular parsing strategy. Ideally, the value should be equal to 1.

Useless items (with exhaustive searctri:The ratio E2/D2 (used items / built items after exhaustive search) indicates which proportion of successful tasks are used in any of the final results. If available, the ratio E3/D2 should be used. If this value is low for some rule, the rule is frequently successful, but rarely contributes to the final parse result(s).

Such rules are particularly disastrous, because their successful execution creates new items, which in turn lead to

the generation of new parsing tasks. Such rules should be assigned low priority. An example is the apposition rule, which combines any two adjacent NPs, as in the following examples:

Noam Chomsky, the well-known linguist, ...
Our teacher, a notorious drug addict,

2.5. Towards a Self-optimizing Parser

The results of the above statistics can be used to find a reasonable parsing strategy automatically. In this case, the priority assignment function would take as its only input the rule involved in the parsing task.

A promising parsing strategy would delay rules that

- arc frequently successful, but rarely contribute to the final result (ratio E3/D2 or E2/D2), or
- are not filtered out, but frequently fail (ratio D2/C).

One formulation of such an automatically defined parsing strategy might given below:

```
priority(auto TASK) =
     min( D2(rule(TASK))/C(rule(TASK) )
          E3(rule(TASK))/D2(rule(TASK) )    )
```

where C, D2 and E3 are the statistical figures for a particular rule summed over a representative number of parses.

3. Criteria for Priority Assignment

3.1. Surface Properties

Our experiments have shown that parsing strategies based on surface properties of chart items, such as length, starting position, syntactic category do not have any significant advantages over depth-first search.

Strategies based on the statistics described in section 2.4 were more successful, and reduced the time for finding the first parse of a sentence by about 40 percent. Our experiments only adressed the issue of finding one parse result quickly (which is appropriate for unambiguous strings), but did not at all address the issue of finding the preferred reading first.

3.2. Degrees of Acceptability

Psycholinguistic research strongly suggests that some analyses of an ambiguous sentence are more acceptable than others. While it is not clear how degrees of acceptability are determined, it is quite obvious how they can be used in the definition of parsing strategies: the priority of a parsing task should be high if the items involved in it have a high degree of acceptability.

In addition, rules should be given an initial priority, which may be determined by the function given in section 2.5. The same is true for different readings of lexical items, which can be given an initial probability.

The priority of a parsing task is then a function of

- the degree of acceptability of the constituents involved (or the initial probability of readings of lexical items)
- the initial priority of the rule involved

One possible such function for a task involving n items would be:

```
priority(average-acceptability TASK) =
  average( initial-priority(rule(TASK)),
           average( acceptability(item(1 TASK))
                    ...
                    acceptability(item(N TASK))))
```

It would be mathematically more tempting to view the acceptability values for items and initial priorities for rules as probabilities, because the theory of probability is well understood. The priority assignment function could then be defined by multiplying all the probabilities:

```
priority(probability TASK) =
     initial-priority(rule(TASK)) *
     acceptability(item(1 TASK))  *
     ...                          *
     acceptability(item(N TASK))
```

Two objections may be raised against such an approach. First, multiplication will always make the probability of a new node equal to or less then the probabilities of its daughters. For this reason, the probability decreases as trees get larger. This effect must somehow be compensated. The second objection against probabiltics is that serious calculation with probabilities requires reliable statistical data, which may not be available.

In the following we discuss some factors which play a role in determining the acceptability of a constituent

3.2.1. LP-rules

In German, violation of a linear precedence rule does not make a string completely unaccaptable, but reduces its adaptability [Uszkorcit 1986]. Each LP-rule may have a different weight, and violation of the LP-rule will decrease *the* acceptability of the constituent in which the rule is violated according to its weight.

3.2.2. Attachment Preferences

Attachment preferences have been extensively studied in psycholinguistics (e. g. [Fodor and Frazier 1970]). One main principle is Right Association, which means that a modifier is "attached into the phrase marker as a right sister to existing constituents and as low in the tree as possible".

This strategy can be modelled by a strategy, which prefers parsing tasks in which the modifier is the rightmost constituent, and which have a short span. Semantic factors can override syntactic attachment preferences.

3.2.3. Semantic Processing

Since the purpose of language is to convey meaning, the acceptability of substrings cannot be determined solely on

syntactic grounds, but must also take into account the semantics. Early semantic processing is useful for checking selectional restrictions, and in the resolution of ambiguities. With the approach presented here, the results of early semantic processing can be integrated directly into the parsing process, and help to choose the most promising paths in the search space.

3.3. Ill-formed Input

Parsing strategies are very useful for dealing with ill-formed input. In one scheme for processing ill-formed input (Erbach 1987, Mellish 1989], hypotheses are added to the chart in order to correct ill-formedness. For example, missing words are added, superfluous words removed. These hypothetical items are then used to construct items which would be constituents if the hypothesis were true.

In order to prevent the unrestricted introduction of hypotheses, a cost is associated with each hypothsis. The cost associated with a constituent is the sum of the costs of its daughters. A parsing strategy will try to minimize the cost, i. e. the higher the cost of the resulting item, the lower its priority.

In this way, the analysis which contains the fewest and the less costly hypotheses is preferred.

4. Conclusion

We have presented an environment for the experimentation with parsing strategies consisting of:

- a parser which can process various parsing strategies
- a functional language for the definition of strategies
- a statistical component which helps the user assess the effects of the different strategies.

There are two general approaches which seem promising for discovering a good parsing strategy.

The first approach is to use the statistics compiled about individual rules in order to assign a priority to each rule, which is then used as the priority of the parsing task involving the rule. This method can be applied automatically.

The second, and more ambitious, approach, involves the notion of degree of acceptability of a constituent. Such an approach makes the priority of a parsing task depend on the degrees of acceptability of the constituents involved, and on the initial priority of the rule.

The most important research issues are

- how to determine degrees of acceptability,
- how to calculate the degree of acceptability of a new constituent from the degrees of acceptability of its daughters, and
- how to formulate a parsing strategy that makes use of degrees of acceptability and other properties of the parsing task, in order to obtain the desired reading with a minimun of search.

References

[Bouma el al. 1988] G. Bouma, E. Konig, and H. Uszkoreit. A Flexible Graph-Unification Formalism and its Application to Natural-language Processing. IBM Journal of Research and Development 32(2), 170-184.

[Chomsky 1965] Noam Chomsky. Aspects of the Theory of Syntax. Cambridge, MA: MIT Press.

[Chomsky 1981] Noam Chomsky. Lectures on Government and Binding. Dordrecht: Foris.

[Erbach 1987] K. G. Erbach. Parsing Ill-formed Input with an Augmented Chart Parser. Appendix to: K. G. Erbach. An efficient chart parser using different strategies. Department of Artificial Intelligence Discussion Paper Number 52. University of Edinburgh.

[Fodor and Frazier 1970] J. Fodor and L. Frazier. The Sausage Machine: A New Two-Stage Parsing Model. In Cognition 6.

[Gazdar et al. 1985] G. Gazdar, E. Klein, E. Pullum, I. Sag. Generalized Phrase Structure Grammar. Oxford: Basil Blackwell.

[Haugeneder and Gehrke 1988] Hans Haugeneder and Manfred Gehrkc. Improving Search Strategies, An Experiment in Best-First Parsing. COLING 1988, Budapest, 237-241.

[Kay 19801 Martin Kay. Algorithm Schemata and Data Structures in Syntactic Processing. Report CSL-80-12, Palo Alto, CA: XEROX PARC.

[Mellish 1989] Chris Mellish. Some Chart-Based Techniques for Parsing Ill-formed Input. 27th ACL Proceedings, Vancouver, 102-109.

[Pereira and Shieber 1987] F. Pereira and S. M. Shieber. Prolog and Natural Language Analysis. CSLI Lecture Notes No. 10, Stanford, CA.

[Pereira and Warren 1980] F. Pcreira and D. H. D. Warren. Definite Clause Grammars for Natural Language Analysis. A Survey of the Formalism and a Comparison with Augmented Transition Networks. In Artificial Intelligence 13,231 -278.

[Pollard and Sag 1987] Carl Pollard and Ivan Sag. Information-based Syntax and Semantics. Volume 1: Fundamentals. CSLI Lecture Notes No. 13, Stanford, CA, 1987.

[Uszkoreit 1986] Hans Uszkoreit. Constraints on Order. Report No. CSLI-86-46. Center for the Study of Language and Information. Stanford, CA.

[Wir6n 1987] Mars Wiren. A Comparison of Rule-Invocation Strategies in Context-Free Chart Parsing. 3rd European ACL Conference Proceedings. Copenhagen, 226 - 235.

[Woods 1970] W. A. Woods. Transition Network Grammars for Natural Language Analysis. Communications of the ACM 13,591 -606.