Robust and Efficient Semantic Parsing of Free Word Order Languages in Spoken Dialogue Systems

Ralf Engel

DFKI GmbH, Stuhlsatzenhausweg 3, 66123 Saarbrücken, Germany
ralf.engel@dfki.de

Abstract
This paper presents SPIN, a semantic parser for spoken dialogue systems which is especially designed to analyze efficiently free word order languages.

Characteristic for semantic parsers is that the output structure is built up more or less directly from word level without a deep syntactic analysis. Typically, semantic parsers provide fast and robust processing and simplify the creation of the knowledge bases. The complexity of the utterances which can be processed is somewhat limited, but this is acceptable for most spoken dialogue systems.

Most semantic parsers use as underlying formalisms context free grammars (CFGs), e.g., [1] or finite state transducers (FSTs), e.g., [2] or variants of them, e.g., [3, 4]. These systems are well suited for fixed word order languages, like English or French but less for free word order languages, like German, Turkish, Japanese, Russian or Hindi.

To provide a better support for free word order languages, we developed a new approach which supports order-independent matching, i.e., the order of elements in the input stream is not important when a rule is applied. Order-independent matching is already used for the syntactic analysis of free word order languages, usually in combination with ordering constraints, e.g., [5, 6]. A positive side effect of order-independent matching is that the robustness against speech recognition errors and syntactically incorrect user input is increased.

But order-independent matching makes efficient parsing much more difficult, parsing of arbitrary grammars is NP-complete [7]. We developed a new parsing approach which provides efficient parsing on rule bases that are, according to our experience, typical for spoken dialogue systems. Fast processing is important as many speech recognizers produce word lattices, so faster processing means that more paths in the word lattice can be analyzed and a better overall performance of the dialogue system can be achieved. Key elements of the parsing algorithm are a fixed application order of the rules and pruning of irrelevant results.

The SPIN parser implements a rewriting system that works on typed feature structures, i.e., a rule matches some feature structures and replaces them with a newly created typed feature structure. Such an approach is also used in constraint based parsing approaches for syntactic analysis. The recognized words are represented as typed feature structures enriched with additional syntactical information like stem or part of speech looked up in a lexicon. The usage of typed feature structures as basic representation formalism allows the processing of complex utterances, like nested database queries. Such queries cannot be handled by simple slot filling techniques, e.g., as used in VoiceXML.

The parser is successfully used in different dialogue systems, including SmartKom [8], MIAMM [9] and SmartWeb [10].

The paper is structured in the following way: In section 2, order-independent matching is defined and the advantages are described using German as example language. Section 3 describes the newly developed parsing algorithm and section 4 presents a preliminary evaluation of the parsing approach.

2. Order-independent matching in action
2.1. Definition of order-independent matching
Order-independent matching means that the elements in the input stream do not need to be in the same order as the conditions and the matched elements do not need to be contiguous. For example the rule

\[X' \rightarrow C' A'\]

matches the input \( A B C D \) and generates \( X B D \).

Order-dependent matching, i.e., the location of the input elements is regarded, can be embedded in rules using so-called order sequence (expressed with square brackets), e.g.,

\[X' \rightarrow D' [A' B' ]\]

As the matched input elements do not need to be continuous, the order of elements in the stream cannot be used anymore to check if two elements are adjacent after a new element has been inserted. Instead, the built-in features \( \text{from} \) and \( \text{to} \) store the positions. For newly inserted elements the features are filled with the positions.
with the minimum value of all from features of the matched elements and the maximum value of all to features.

2.2. Word order phenomena in German

Constituents can be reordered relatively freely in German. E.g., in main clauses only the location of the verb complex is fix. The other constituents, like subject, objects, prepositional phrases and adverbs can be placed at the first position (so-called Vorfeld) or after the first part of the verb complex (so-called Mittefeld). The optimal location for each constituent depends on various circumstances, e.g., if a constituent is emphasized or newly introduced, but also the length of a constituent is relevant. The function of a constituent is either defined by its case or, if the case is not obvious, by its semantic meaning or by the preferred reading.

The utterances (1) and (2) taken from the SmartKom project [8] illustrate the reordering of constituents:

(1) Ich will heute mit dem Auto nach Hannover fahren. (I want to today by car to Hannover.)
(2) Heute will ich nach Hannover mit dem Auto fahren. (Today want I to Hannover by car.)

2.3. Processing of utterances

A great advantage of free-order matching is that a single rule is sufficient to cover the different reordering variants. E.g., the above utterances are analyzed by the following rule: ²

\[
\text{ich will} \stackrel{\text{ComputeRoute}}{=} \begin{cases} \text{end:City(name:Hannover),} \\ \text{means:Car(),time:Today()} \end{cases} \]

As the matched input elements do not have to be continuous, it is possible that general constituents, like temporal expressions, certain adverbs, e.g., bitte (please) and other expressions, like ich will (I want) are handled in separate rules. Examples for rules of this type are:

\[
\begin{align*}
\text{A(mode:polite)} & \rightarrow \text{A=Action()} \text{ bitte} \\
\text{A(time:$T)} & \rightarrow \text{A=TimedAction()} \text{ $T=Time()} \\
\text{A} & \rightarrow \text{ich will} \text{ A=Action()} \\
& (\text{ bitte = please, ich will = I want to})
\end{align*}
\]

If the input elements had to be continuous, the above rules would have to be integrated in all rules processing verb phrases.

Order-independent matching increases also the robustness against speech recognition errors, as the parts of the utterance that are recognized incorrectly can be skipped. This is a mechanism that is also used in other approaches, e.g., [11]. An example is if utterance (1) is recognized incorrectly as

(3) Ich will *Leute mit *Demo nach Hannover fahren.
(I want to *people by *demo to Hannover.)

at least the action together with the target location could be extracted. Unprocessed words are still part of the final result and can be filtered out. The portion of unprocessed words can be used in a scoring function for the results.

Additionally, the robustness against errors produced by human speakers is increased. Many spoken utterances contain inflection errors and wrongly located words and constituents. The first problem can be tackled by matching the stem of word instead of the orthography and the second one is not problematic when order-independent matching is used.

Another useful feature of order-independent matching is that a single rule covers questions for different constituents if the questioned constituents are marked with a special feature, e.g., focus. Example for rules are:

\[
\begin{align*}
\text{Time(focus:whQuery)} & \rightarrow \text{Word(orth:wann)} \\
\text{$T(focus:whQuery)} & \rightarrow \left\{ \text{Word(stem:welch) $T=Thing()} \right\} \\
& (\text{wann = when, welch = which})
\end{align*}
\]

If these rules are used, the rule

\[
\begin{align*}
\text{EPG(avMedium:$M,channel:$C)} & \rightarrow \text{Word(stem:kommen) $M=AvMedium()} \\
& \% \text{ auf $C=Channel()} \text{ $T=Time()} \\
& (\text{kommen = are broadcasted, auf = on})
\end{align*}
\]

is able to process different questions like

\[
\begin{align*}
\text{Wann kommen Nachrichten?} & \rightarrow \text{When are the news broadcasted?} \\
\text{(Welche Filme kommen auf PRO7?)} & \rightarrow \text{(Which movies are broadcasted on PRO7?)}
\end{align*}
\]

2.4. Additional remarks

In this approach the function of a constituent (e.g., subject, object) is assigned on a semantic basis. Our experience shows that this is sufficient in almost all cases and increases the robustness against incorrectly recognized inflections. But the parser includes also extensions that provide access to the case stored in the lexicon, so it is also possible to determine the function of a constituent on a syntactic basis.

Usually, rules using order-independent matching can be applied successfully to syntactically incorrect utterances without recognizing them as syntactically incorrect. To some degree, this is a disadvantage as utterances with speech recognition errors leading to syntactically incorrect utterances cannot be detected.

3. Parsing algorithm

3.1. Parsing challenge

Typically, parsing algorithms are optimized to avoid the generation of multiple identical (intermediate) results and intermediate results that cannot be further processed. The first issue can be addressed using a chart, the second one using top-down predictions.

Parsing of order-independent rules adds another issue: Many results are generated which can be regarded as irrelevant for further processing; a phenomenon which is unknown when parsing pure order-dependent rules, like CFG-rules.

Irrelevant results are produced when rules are not applied although they could be applied in principle. An example should illustrate this: The utterance

\[
\text{Nimm die Sendung auf!} \\
\text{(Record the broadcast!)}
\]

²In the following examples we use a PROLOG-style syntax to represent the typed feature structures. The conditions within the rules are partially instantiated typed feature structures. They have to subsume the input elements to match them. Variables start with $, optional conditions with %. word is used as an abbreviation for Word(orth:word).
has the two results

Record(object:Broadcast(det:def))

and

Record(object:Broadcast()) Word(orth:die)

The second result was generated without the application of the rule responsible for processing determiners. If order-dependent matching is used in the above case, the second result is not generated as the rule responsible for producing the Record structure would fail at the unprocessed determiner die. In the case of order-independent matching the unprocessed die is simply ignored.

This is a severe problem as this can happen at several locations in the utterance and with several rules. The number of different results is the product of the number of alternatives at each location.

3.2. The parsing algorithm

The parsing algorithm tries to prune results that can be regarded as irrelevant. Therefore, two methods are combined: The rules are applied in a fixed application order and the rules are either marked as destructive or non-destructive. We first describe both methods and then we explain how the combination of them manages to prune irrelevant results. Pruning of results means that the parsing algorithm is not complete.

3.2.1. Fixed application order

We use a chart based button-up parser with a fixed application order of rules, i.e., each rule has an application number and the rules are applied in that order. Rules with the same application number are applied repeatedly in a loop until none of these rules can be applied anymore.

The application number for each rule is determined automatically offline in a preprocessing step. Therefore, a dependency graph between the rules is constructed which is linearized afterwards. A relation between rule A and rule B exists if at least one instance exists that can be created by rule A and deleted by rule B.

Before the dependency graph is linearized, the graph is checked for cycles. A cycle means that a rule A can process the result generated by a rule B, but rule B can also process the result generated by rule A. If a cycle is detected, all rules of that cycle get the same application number avoiding that relevant solutions are not generated.

Besides the pruning effect, the fixed application order reduces the number of necessary rule tests and improves further the parsing performance.

3.2.2. Destructive and non-destructive rules

Additionally to the application number, a rule is marked as either destructive or non-destructive. If the rule is marked as destructive the new result replaces the matched elements in the chart and the intermediate result before the rule application gets lost. If the rule is marked as non-destructive the new result is added as another solution to the chart.

If a rule is destructive or non-destructive is also determined automatically in an offline process. A rule A is marked as non-destructive if at least one other rule B exists that is able to match partially the same input and can be applied after rule A, i.e., the application number of rule B is equal or greater than of rule A. If rule A was not marked as non-destructive, it might happen that rule B cannot be applied as part of the necessary input is already deleted by rule A. The algorithm to detect if two rules can match partially the same input is somewhat complex (as order sequences have to be regarded) and is not explained in detail here.

Destructive rules have also the positive side effect that the size of the chart is reduced as elements are removed from the chart.

3.2.3. Pruning of irrelevant results

The fixed application order combined with destructive rules prunes some of the irrelevant results caused by unapplied rules as described in section 3.1. The reason why these rules are pruned is that most of the modifying rules are marked as destructive and that they are ordered before the rules consuming the modified elements.

Figure 1 illustrates the application order of rules using a small set of example rules.
3.2.4. Multiple solutions for one rule application

The application of one rule can have more than one solution as the conditions can match different elements in the input stream and different paths in the chart can be selected. The parser computes first one solution and stores all necessary information to compute the next solutions in an agenda. Some heuristics are used to compute the assumed best solution first, e.g., input elements located closer to already matched input elements are tested first. The information stored in the agenda includes a copy of the chart as destructive rules can destructively modify the chart later on. (Parsing of CFGs requires only one chart as the entries are only added.)

If the next solutions are directly computed before the next rule is applied, all results can be inserted into the same chart. If the next solutions are computed later after other rules have been applied, the copy of the chart, which does not contain the changes of the later applied rules, has to be used. Therefore, some intermediate results may be generated more than once during further processing. This is a disadvantage of the approach but otherwise pruning of solutions would not be possible. It is a current research topic how at least part of the already computed intermediate results can be reused.

By modifying how many solutions are computed at once, the parser can be tuned to work either more depth-first (providing a better anytime property) or more breadth-first (reducing the time needed to compute all solutions).

3.3. Restrictions

The described pruning mechanism does not work well with arbitrary rule sets. In the worst case, if all rules depend in a cyclic way on each other, all rules get the application number 1 and are marked as non-destructive. This would lead to an unoptimized bottom-up parsing strategy. Not only such extreme cases degrade the performance, but also alleviated cases, e.g., lengthy loops or a high portion of non-destructive rules. But the experience we have made is that this is not the case even if the dialogue system is more complex and comprises several applications.

4. Evaluation

The parser is used in different dialogue systems. The system with the largest functionality and the largest number of rules is the SmartKom system [8] which is used for the following evaluation. SmartKom covers different applications including EPG (electronic programming guide), reservations for movie theaters and route planning. In this project 435 rules are used. The average size of 8.0 rules, the largest loop contains 39 rules. 158 rules are marked as non-destructive. This would lead to an unoptimized rule set. In the worst case, if all rules depend in a cyclic way on each other, all rules get the application number 1 and are marked as non-destructive.

The described pruning mechanism does not work well with arbitrary rule sets. In the worst case, if all rules depend in a cyclic way on each other, all rules get the application number 1 and are marked as non-destructive. This would lead to an unoptimized bottom-up parsing strategy. Not only such extreme cases degrade the performance, but also alleviated cases, e.g., lengthy loops or a high portion of non-destructive rules. But the experience we have made is that this is not the case even if the dialogue system is more complex and comprises several applications.

We tested the performance on a Pentium IV 3.2GHz computer with a test corpus of 268 utterances with an average length of 5.1 words and a maximal length of 12 words. The average processing time was 6.8 ms, the largest one 62 ms (including time for garbage collection). The short parsing times allow to parse several paths of a word lattice until a timeout is reached.

Currently, we work on a more elaborated evaluation, which checks the exact effects of the various optimizations.

5. Conclusions and outlook

In this paper we presented an efficient and robust semantic parser for free word order languages. We demonstrated the advantages of order-independent matching with several example utterances and rules. A new parsing approach was introduced enabling fast processing with rule bases that are, according to our experience, typical for dialogue systems. A preliminary evaluation showed that the parser provides fast processing in a real-world dialogue dialogue system.

A current research topic is to examine how order-independent matching can be used to process multiple input streams, e.g., for multimodal dialogue systems including gesture input or for processing ellipses by including the last utterance as additional stream.

6. Acknowledgments

This research was funded by the German Federal Ministry for Education and Research under grant number 01IMD01A. The views expressed are the responsibility of the authors. Points of view or opinions do not, therefore, necessarily represent official Ministry for Education and Research position or policy.

7. References