Acta Cybernetica **17** (2006) 617–632.

# Word Order and Discontinuities in Dependency Grammar

C. Bartha*, T. Spiegelhauer†, R. Dormeyer†, and I. Fischer†

**Abstract**

Natural languages are always difficult to parse. Two phenomena that constantly pose problems for different formalisms are word order—what part of a sentence has to be placed where—and discontinuities—words that belong together but are not placed into the same phrase. Dependency grammar, a linguistic formalism based on binary relations between words, is very adequate for handling both problems. A parser for dependency grammar together with its grammar writing formalism is described in this paper. Word order and discontinuities in Hungarian are handled based on this formalism.

## 1 Introduction

When taking a look in the standard literature [9] on computational linguistics, long introductions in phrase structure grammars invented by Chomsky can be found. They have been in the focus for nearly fifty years now. Phrase structure grammars turned out to be a helpful method when modeling English; quite a lot of parsers can be found together with extensive grammars. But it also turned out that they are not useful when it comes to languages with free or semi-free word order. Discontinuous constituents and long distance dependencies pose difficulties, too. Several work arounds and extensions have been invented to overcome these problems. Some of these extensions and new developments, e.g. *Head-Driven Phrase Structure Grammar* [13], are similar to dependency grammar. Dependency grammar, invented by Lucien Tesnière [14], [15], is popular in Europe and Japan. In this paper, a parser for dependency grammar [4] is described. Currently grammars are written for several different languages. Grammar fragments for English, German and Latin have been written [4]. These languages differ in their word order. English has a fixed word order, e.g. the subject has to come first in a declarative sentence. In German, the word order is semi-free. For noun phrases, it is fixed; on the sentence level, the verb has to be in the second position in a declarative sentence. Other elements can

*Siemens PSE Hungary, Szeged, CSS IBS5, E-mail: csongor.barta@siemens.com

†Lehrstuhl für Informatik 2, Friedrich–Alexander Universität Erlangen–Nürnberg, Martensstr. 3., 91058 Erlangen, Germany, E-mail: tilly79@gmx.de, ricarda@dormeyer.de, Ingrid.Fischer@informatik.uni-erlangen.de

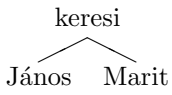


Figure 1: A dependency tree for *János keresi Marit.*

take the other positions, no restrictions are given here. In Latin there are even less word order restrictions. Words can be placed nearly everywhere.

But not only word order is of special interest. Another problem are words that belong together but are not placed next to each other in the sentence. This phenomenon can be found in all languages analyzed. An English example is fronting as in *Ann John told me he had seen.* In German and Hungarian verb prefixes can be separated from the verb and move to another position.

At the moment especially non-indo-European languages are researched. Currently grammars for Japanese [17] and Hungarian are developed. For Japanese, a lot of different dependency grammar implementations exist. This is not the case for Hungarian. Only one international publication could be found containing a dependency grammar for Hungarian [18]. The grammar described in [18] differs a lot from our approach. In [18] first all prefixes and suffixes are separated from word stems. The resulting string is the input for the dependency parser. In our approach this separation does not take place, a sentence is analyzed in its original writing.

In the sequel, our dependency parser and the Hungarian grammar developed up to now are described. In Section 2 the basics of dependency grammar are introduced. After this linguistic introduction, our dependency parser is specified in Section 3. Special features of our Hungarian grammar are given in Section 4. In Section 5 we describe the underlying algorithm. We end with a conclusion.

## 2 A Short Overview on Dependency Grammar

In dependency grammar binary relations between the words of a sentence are used as the basic construct. The most important part of a sentence is the verb, it opens several slots for other parts of the sentence. Taking the verb *keresi (to seek)*[1] it opens two slots, one for a noun in the nominative case, which is the subject, and one for a noun in the accusative case, the object. In the sentence *János keresi Marit (János seeks Mari)* these slots are occupied by *János*, the subject, and *Marit*, the object. Normally the relations are visualized with the help of trees. A tree for the running example is given in Figure 1.Please note, that every combination of the three words results in the same dependency tree: *János Marit keresi*, *Marit keresi János*, ....

[1] All Hungarian examples used throughout this paper including the English translations are cited from [11]. This example is taken from page 2, example (1).

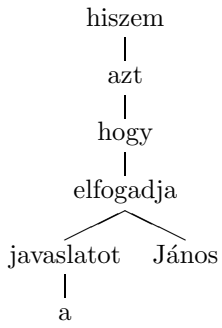


Figure 2: A dependency tree for *János azt hiszem hogy elfogadja a javaslatot. (János, I think, that (he) accepts the proposal.)*

The subject and object can also open new slots. A simple noun opens a slot for e.g. one determiner and any number including zero of adjectives. This means that several different kinds of slots are necessary. First slots are used that must be filled, with one element. If the element is missing, the corresponding sentence is grammatically not correct. In Figure 1 exactly one object is needed. In English each verb needs exactly one subject. Then there are slots that are optional, they can be filled but do not have to be filled. Time and place are optional for most verbs. They can be added, but they can also be left out. Another example for this is the subject in Hungarian. Finally there are slots that can be filled several times, e.g. a noun can take several adjectives. A word opening a slot is also called the head, the word filling the slot will be called the dependent in the rest of the paper.

Long distance dependencies or discontinuous constituents are another phenomenon that pose constant problems to formal grammars and the corresponding parsers. In Hungarian long distance dependencies have been described by different researchers with the earliest publication stemming from the beginning of the last century [11]. In dependency grammar a relation between a head and a dependent is considered discontinuous if not all words between this head and this dependent depend on one of the two. A Hungarian example with discontinuous constituents is given in the following table: [2]

| | | | | | | |
|---|---|---|---|---|---|---|
| *János* | *azt* | *hiszem* | *hogy* | *elfogadja* | *a* | *javaslatot* |
| *János* | *that-acc* | *I think* | *that* | *accept* | *the* | *proposal* |
| *János, I think that (he) accepts the proposal* | | | | | | |

*János* depends on the verb *elfogadja (accepts)*. Between the head *elfogadja* and the dependent *János*, the words *azt hiszem hogy* are found. *hiszem (I think)* as the main verb is the head of the sentence. All other words including *elfogadja (accepts)* and *János* depend somehow on *hiszem*. *elfogadja* depends of the head *hogy (that)* wich depends directly from *azt (that-acc)*. *azt (that-acc)* finally depends on *hiszem*

[2] This example including the English translation is taken from [11], page 258, example 77.

*(I think)*. Also *azt (that-acc)* and *hogy (that)* form a discontinuity as *hiszem (I think)* in between does not depend on either of the two but *azt (that-acc)* depends on *hiszem*.

The corresponding tree is given in Figure 2. The linear structure of the words in this sentence cannot be reconstructed from Figure 2. In this tree only the syntactic structure is given.

In phrase structure grammar linear and syntactic structure are combined in one tree leading to crossing edges in the phrase structure tree for this sentence. Trees with crossing edges cannot be constructed with context-free grammars.

# 3 A Parser for Dependency Grammar

Our parser [16] is based on three concepts. The parsing algorithm itself is similar to the well–known Cocke–Kasami–Younger algorithm for context-free phrase structure grammars [9]. The first dependency parser based on this idea was presented in [12]. Words are described by feature structures [9] enriched by a few symbols necessary for dependency grammars. Feature structures are combined with the help of graph unification. Our handling of discontinuous constituents and word order restrictions differs from [12].

## 3.1 Word Order

In Tesnière's original approach, word order was unimportant for syntactic description. Any order was allowed. This is not useful for parsers, too many wrong sentences would be accepted. The order between head and dependent must be considered as well as the order between the different dependents of one head. Also the number of elements following or preceding a word can be important. E.g. in German the verb in a declarative sentence must fill the second position. In our approach each word has a position list where positions of the word itself and other words are described. This includes as a minimum a position for the word itself. Then each dependent of a word can have a fixed position in this position list. Free positions within the position list are also possible, a free position can take every dependent that is not marked as fixed. The position list makes parsing easier: When a fixed position is following according to this list, the parser has to check for just one element. If the next element is free, only free elements have to be checked.

## 3.2 Discontinuous Constituents

Linguistically, a discontinuous dependency can be regarded as a ternary relation, i.e. a relation between a dependent, its *syntactic head* and its *linear head* [1]. The syntactic head is the word containing a slot for the discontinuous dependent. But because the syntactic head's constituent is discontinuous, the dependent is positioned in the position list of the linear head. In the example sentence, the syntactic head for *János)* is *elfogadja (accept)* and its linear head is *hiszem (I think)*. The

```
Word "Angéla" <"Name"> [
    lexeme: Angéla;
    gender: fem;
    case: nom;]

Word "olvas"   <"VerbPres"> [
    lexeme: olvas;
    mood: declarative;
    number: sing;
    person: 3;]

Template "Name" [
    category: noun;
    special: propername;
    number: sing;
    person: 3;]

Template "VerbPres" [
    category: verb;
    form: finite;
    tense: present;
    sentence: declarative;
    subj:  oslot [
        category: noun;
        cont: +;
        case: nom;];
    order: (%1 %2 i %3);]
%1 = slot  [cont: +;];
%2 = mslot [cont: +;];
%3 = mslot [cont: +;];
```

Figure 3: Simple grammar with templates for *Angéla olvas (Angéla reads)*

dependent fills a slot of its syntactic head, but occupies a position of the linear head's position list. Because of this, the processing of discontinuous dependencies has to work with linear and syntactic head. The parser must allow for all possible orders between syntactic head, linear head and dependent in a sentence.

### 3.3 Other Approaches

Over the years several other parsers for dependency grammar have been proposed. In [1] linear order and syntactic order are strictly separated, something we tried to avoid in our approach, because it makes grammar writing less intuitive and parsing less efficient. Fraser's parser [5] is based on backtracking and uses a parsing stack. Covington's approach [2] is cited very often. He invented a simple backtracking algorithm for free word order. But his approach is not well suited for semi-free word order phenomena. Finally there are several dependency parsers based on constraint resolution, a completely different approach [3].

## 4 Parsing Hungarian

In this section two Hungarian sentences are analyzed. With these examples our grammar description language is described.

### 4.1 Grammar Description Language and Free Word Order

The grammar description language is important, as it must be easy to learn and to handle for the linguist writing grammars for the parser. We will introduce it with the help of the easy example *Angéla olvas (Angéla reads.)*. In Figure 3, the corresponding grammar is given. Please note that due to space our example grammars are not complete.

To shorten the grammar, templates as introduced by [6] are possible. Templates encode parts of the feature structures that are used very often. Within the entries for words template names are used instead of complete feature structures. As a first step, the lexicon is transformed before parsing. Template names are removed, the feature structure parts these names described are unified with the rest of the feature structure.

Feature structures are started by "[" and ended with "]", features and values are separated by ":" and feature-value-pairs are separated by ";". In Figure 3 two word entries and two templates are given. The lexical entry for *Angéla* contains a template named `Name`, which is also given. Feature structures for `Name` and *Angéla* are unified leading to an entry where agreement features as `number, gender, case` (not all possible cases are given) and `person` are described. Also the `lexeme` and `category` are shown. The verb *olvas (reads)* is also composed with the template for verbs in present tense. This template is more interesting. It contains an optional slot for a subject indicating that in Hungarian, the subject can be left out. At the end the special feature `order` indicates possible positions. As this position list is used for every word in present tense, it is more complicated than necessary for our small example. The symbol `i` stands for the position of the word itself, in this case the current verb. Before this verb at least one element must be placed. %1 must be a slot, this slot must be filled. %2 is marked `mslot` (multiple slot). A multiple slot can be filled with an arbitrary number of elements but can also be empty. An arbitrary number of elements can also follow after the verb. It can also be described that the subject must go in the first position; in this case %1 has to be added to the subject slot. Please note that this is a lexicalised grammar, i.e. all information is stored in the lexicon, no extra grammar rules are needed.

## 4.2 An Example With a Discontinuity

Our treatment of free word order has already been introduced in the previous Section 4.1. Now a more complicated example introduced in Section 2 *János azt hiszem hogy elfogadja a javaslatot. (János, I think, that (he) accepts the proposal.)* is used to show how discontinuities are handled. The dependency tree has already been shown in Figure 2. In Figure 4 a short and not complete grammar for the example sentence is given. Templates are left out for simplicity. In the grammar, continuity and discontinuity are marked using special features. A feature `cont` is used to specify whether a dependency may be realized only continuously ("`+`"), only discontinuously ("`-`"), or both (not specified); a second feature `cont-const` is used to specify whether dependents may be extracted from the constituent headed by a certain word[3]. Both features can be applied to lexical entries of words, to slots, or to positions in a position list. Specification of dependents, slots or positions as continuous will stop this process from taking place.

To parse the sentence for example 4, the parser first encounters the words *János* and *azt (that-acc)*. *Azt (that-acc)* contains an open slot for a subjunction and

[3] No example for `cont-const` is given.

```
Word "János" [
  number: sing;
  person: 3;
  gender: masc;
  case: nom;
  lexeme: János;
  special: propername;
  category: noun;]

Word "azt" [
  conj: slot [
     category: subjunction;];
  lexeme: az;
  category: defpronoun;
  case: acc;
  order: (i);]

Word "hiszem" [
  category: verb;
  sentence: declarative;
  dir-obj: %2 slot [
      case: acc;
      category: defpronoun;];
  lexeme: hisz;
  numerus: sing;
  person: 1;
  order: (%1 %2 i %3);]
%1 = oslot [];
%3 = oslot [];
```

```
Word "hogy" [
    category: subjunction;
    lexeme: hogy;
    prop: %1 slot [
        category: verb;
        cont: +;];
    order: (i %1);]

Word "elfogadja" [
    category: verb;
    lexeme: elfogad;
    subj: oslot [category: noun;
                 case: nom;];
    dir-obj:  slot  [category: noun;
                     case: acc];
    order: (%1 i %2);]

Word "a" [
    category: determiner;
    lexeme: a;]

Word "javaslatot" [
    category: noun;
    case: acc;
    lexeme: javaslat;
    spec: %1 oslot [
        category: determiner;
        cont: +;];
    order: (%1 i);]
```

Figure 4: Simple grammar for *János azt hiszem hogy elfogadja a javaslatot. (János, I think, that (he) accepts the proposal.)*

therefore cannot act as head for *János*. The next word *hiszem (I think)* has an open slot for an definite pronoun in accusative. This slot can be filled with *azt (that-acc)*. This slot is marked with %2 and in the feature `order` of *hiszem (I think)* it is indicated, that it has to be positioned in front of the verb itself. *János* can fill position %1 in this list. %1 is not bound to any slot of the verb *hiszem (I think)* so the syntactic head of *János* is not available yet. The open slot of *azt (that-acc)* can not be filled with any word between *azt (that-acc)* and *hiszem (I think)*, so this slot is passed up to the syntactic head of *azt (that-acc)* which is *hiszem (I think)*. After *hiszem (I think)*, *hogy (that)* is found by the parser. *hogy (that)* can fill the slot of *azt (that-acc)* that has been passed up to *hiszem (I think)*. Additionally in the feature `order` of *hiszem (I think)*, *hogy (that)* can fill the position %3. *hogy (that)* has an open slot for an subordinate clause with a verb as head. `cont:+` indicates that this slot cannot be moved up. The next word word *elfogadja (accepts)* fills this slot and opens two new ones. One slot is for the subject, that is optional in Hungarian, he second slot is for an object. This object slot is filled by the final two words *a javaslatot (the proposal). javaslatot (proposal)* opens a slot for a determiner. This slot can not be moved up and the determiner must

be positioned in front of *javaslatot (proposal)*. *javaslatot (proposal)* then fills the object slot of *elfogadja (accepts)*. The subject slot of *elfogadja (accepts)* remains unfilled up to now. Because this slot can be discontinuous, it is now moved up the dependency tree until it reaches *hiszem (I think)*, where it can be filled with *János*. Similarly the sentences *Azt hiszem hogy János elfogadja a javaslatot*, *A javaslatot azt hiszem hogy János elfogadja.*, ... can be parsed with the help of this grammar. They have a similar meaning than the original sentence but stress different parts.

# 5 Algorithmic Description

The first subsection describes the simpler variant of the algorithm, which cannot deal with discontinuities [4], [16]. The subsequent subsections describe the extensions necessary to deal with discontinuities.

## 5.1 Parsing Sentences without Discontinuities

In the description of the algorithm, lexical entries without the complete feature structures will be used. The words' lexical entries are based on the part-of-speech $K$ of a word and three lists:

- First for each word an unsorted list of empty slots $\mathcal{L}$, that can be filled, is necessary.
- Additionally, each lexical entry contains a position list $P$, that is split in two parts:
  - $P_{\leftarrow}$ contains the positions to the left side of the word described and
  - $P_{\rightarrow}$ contains positions to its right side.

  Both lists start with the position that is closest to the current word, the following entries are sorted according to the distance from the word described. Positions are named according to their entry in $\mathcal{L}$ with their part of speech from the unsorted list of empty slots or $x$, if they have no corresponding empty slot. Those $x$ labelled entries can be filled by any word.

The basic data structure of the algorithm is a chart. When parsing a sentence $w_0 w_1 \dots w_{n-1}$ consisting of $n$ words, the chart is an $(n+1) \times (n+1)$ table. In this table, sets of chart entries are stored. Entries are never removed from the chart and are immutable after creation. A chart entry at the position $(i, j)$ contains information about the partial derivation of the part of the sentence ranging from the word $w_i$ to the word $w_{j-1}$. A chart entry is also referred to as chart edge. A chart edge is inactive if the lists $\mathcal{L}$, $P_{\leftarrow}$ and $P_{\rightarrow}$ are empty or contain only multiple and optional slots, written as $\langle\rangle$. Otherwise an entry is active, because it still has positions which have to be filled. Based on the chart, the parsing Algorithm 5.1 is used.

**Algorithm 5.1** *main loop (continuous)*

```
1: for j := 1 to n do
2:   for all lexical entries (K_j, L_j, P_←j, P_→j) of a word a_j do
3:     Insert (K_j, L_j, P_←j, P_→j) in m_{j-1,j}
4:     As long as changes are possible:
5:     for all i, k < j do
6:       if k_1 = (D, ⟨⟩, ⟨⟩, ⟨⟩) ∈ m_{i,k} ∧ k_2 = (R, L_R, P_←, P_→) ∈ m_{k,j} ∧ P_← ≠ ⟨⟩
         then
7:         extend(k_1, k_2, i, j)
8:       end if
9:       if k_1 = (R, L_R, ⟨⟩, P_→) ∈ m_{i,k} ∧ k_2 = (D, ⟨⟩, ⟨⟩, ⟨⟩) ∈ m_{k,j} ∧ P_→ ≠ ⟨⟩
         then
10:        extend(k_2, k_1, i, j)
11:      end if
12:    end for
13:  end for
14: end for
15: if m_{0,n} contains an inactive edge then
16:   return true
17: else
18:   return false
19: end if
```

$m_{i,j}$ is the set of chart entries at $(i, j)$. At the beginning all sets are empty. An active chart entry $(R, \mathcal{L}_R, P_{\leftarrow}, P_{\rightarrow})$, which serves as a head, can be extended with an inactive chart entry $(D, \langle\rangle, \langle\rangle, \langle\rangle)$, which serves as a dependent, if one entry is contained in $m_{i,k}$ and one entry is contained in $m_{k,j}$. The constituents described by these chart entries have to lie next to each other in the sentence which is parsed. The extension of chart edges is described in the Procedure 5.2 *extend*.

If the extension of a chart edge from $m_{i,k}$ and a chart edge $m_{k,j}$ was successful, the new edge is inserted in $m_{i,j}$ in the chart, as it contains a derivation for the words $w_i \ldots w_{j-1}$. It is only inserted into the chart if it is not contained in the chart yet. Therefore each entry is contained in the chart only once. If $P_{\leftarrow}$, the list containing the open slots left of the head is empty, i.e. all positions left of the head are filled, the algorithm tries to fill the first position on the right side of the head. A position can be filled if the parts of speech of the head and the position match.[4] For an example of a sentence with no discontinuities, which is parsed with this algorithm, see [16].

[4] And the corresponding feature structures can be unified in the parser.

**Procedure 5.2** *extend*$(k_1, k_2, i, j)$ *(continuous)*

**Require:** $k_1 = (D, \langle\rangle, \langle\rangle, \langle\rangle)$
**Require:** $k_2 = (R, \mathcal{L}_R, P_{\leftarrow}, P_{\rightarrow})$
1: **if** $P_{\leftarrow} = \langle\rangle$ **then**
2: $p := head(P_{\rightarrow})$
3: $P_{\rightarrow} := tail(P_{\rightarrow})$
4: **else**
5: $p := head(P_{\leftarrow})$
6: $P_{\leftarrow} := tail(P_{\leftarrow})$
7: **end if**
8: **if** $p = x$ **then**
9: **for all** $(d, X) \in \mathcal{L}_R$ with $X = D$ **do**
10: write $(R, \mathcal{L}_R - (d, X), P_{\leftarrow}, P_{\rightarrow})$ in $m_{i,j}$
11: **end for**
12: **else**
13: **for all** $(d, X) \in \mathcal{L}_R$ with $d = p \wedge X = D$ **do**
14: write $(R, \mathcal{L}_R - (d, X), P_{\leftarrow}, P_{\rightarrow})$ in $m_{i,j}$
15: **end for**
16: **end if**

## 5.2 Parsing Sentences with Discontinuities

The extension of Algorithm 5.1 to be able to handle discontinuities relies on the following observation. A discontinuous dependency is always part of a continuous constituent, as was described in Section 2. When a discontinuous dependency is established, it cannot be established a single step. It must be estalished in two steps, which might be separated by an arbitrary numbers of steps, because Algorithm 5.7 only tries to combine constituents which lie next to each other. The continuous constituent which contains both the dependent and head of the discontinuity and furthermore contains the position which the dependent fills was called head. A discontinuity is a relation between three words, a dependent, a syntactic head, which contains the slot for the dependent, and a linear head, which contains the position for the dependent. When encountering a discontinuity, two cases must be distinguished. Either the dependent or the syntactic head of the discontinuity appear first in the sentence. For an example where the dependent appears before the syntactic head take a look at the example sentence from Section 2, Figure 2. The dependent *János* is attached discontinuously to its syntactic head *elfogadja (accepts)* and *János* appears before *elfogadja (accepts)* in the sentence. The linear head of the discontinuity of *János* and *elfogadja (accepts)* is *hiszem (I think)*, because *hiszem (I think)* contains the position which the word *János* fills.

The following extensions were made to handle parsing with discontinuous constituents. A word is not a quadruple $(K, \mathcal{L}, P_{\leftarrow}, P_{\rightarrow})$ as in the continuous case, but becomes a quintuple $(K, \mathcal{L}, P_{\leftarrow}, P_{\rightarrow}, T)$, because words can be temporarily attached to other words. Temporarily attached words are stored in the list $T$ and

are a mechanism to enable the parsing of discontinuities. Temporarily attached words are used to handle a discontinuity of the type where the parser encounters a dependent before its syntactic head. As the discontinuous variant of the parsing algorithm only attempts to unify constituents which are adjacent in the sentence, a dependent cannot fill a slot of its syntactic head directly. Therefore the dependent is attached temporarily to the linear head, until the slot from the syntactic head, which the dependent is supposed to fill, is moved up to the linear head. Then the dependent fills the slot. The slot from the syntactic head might be moved up several times before it reaches the linear head. When the dependent is attached to its linear head temporarily, it fills a position. This position must be a free position, which means it cannot be connected to a slot. Were the position connected to a slot, the dependent would fill a position and a slot, and would therefore be attached continuously.

Another mechanism mentioned in the paragraph above dealing with discontinuities is moving up slots. This extension deals with a discontinuity of the type where the syntactic head appears in the sentence before the dependent, as well as with the type of discontinuity mentioned in the above paragraph. As the syntactic head and the dependent cannot be combined directly, the slot which the dependent eventually fills is moved up to the feature structure of the head of the syntactic head when the syntactic head acts as a dependent and fills the slot of another constituent. This slot may be moved up several times, until it finally reaches the linear head. Then the dependent can fill the slot which was moved up.

If a chart entry has an open slot that can be realized discontinuously, this entry can be used as a dependent. This differs from the continuous algorithm. When parsing with continuous constituents, all slots must be filled before a chart entry can be used as a dependent. As mentioned before, discontinuous open slots can be moved up to be filled later. Therefore the Procedure 5.3 *extend* must be changed. To deal with linear heads and the position $x$ (a position which is not connected to a slot) a new Procedure 5.4 $extend\_free$ is introduced. Words that fill a position from the position list of a linear head, are not unified with a slot $\mathcal{L}$ as in the continuous case. They are attached to a possibly linear head temporarily. Later the temporarily attached words will eventually have to fill a slot. What happens in that case is described in the Procedure 5.5 *reduce*. If a word or a constituent fills a position $p \neq x$ from $\mathcal{L}$, it is treated the same way as if it were attached to a word which cannot be a linear head.

### 5.2.1 Moving Up Slots and Attaching Words Temporarily

The Procedure 5.3 *extend* must be changed to move up slots and attach words temporarily. If this procedure is called, the possible dependent contains no temporarily attached words and both its position lists are empty. If the next position to be filled is connected to a slot, then the possible dependent is unified with the slot. A successful unification results in a new chart entry. If the next position to be filled is a free position, then the Procedure 5.4 *extend_free* deals with the dependent. The Procedure 5.4 *extend_free* checks whether the head can be a linear head. If that is

**Procedure 5.3** *extend*$(k_1, k_2, i, j)$ *(discontinuous)*

**Require:** $k_1 = (D, \mathcal{L}_D, \langle\rangle, \langle\rangle, \langle\rangle)$
**Require:** $k_2 = (R, \mathcal{L}_R, P_{\leftarrow}, P_{\rightarrow}, T_R)$
**if** $P_{\leftarrow} = \langle\rangle$ **then**
  $p := head(P_{\rightarrow})$
  $P_{\rightarrow} := tail(P_{\rightarrow})$
**else**
  $p := head(P_{\leftarrow})$
  $P_{\leftarrow} := tail(P_{\leftarrow})$
**end if**
**if** $p = x$ **then**
  *extend_free*$(D, \mathcal{L}_D, R, \mathcal{L}_R, P_{\leftarrow}, P_{\rightarrow}, T_R, i, j)$
**else**
  **for all** $(d, X) \in \mathcal{L}_R$ with $d = p \wedge X = D$ **do**
    **if not** ($\mathcal{L}_D \neq \langle\rangle \wedge (d, X)$ must be a continuous constituent) **then**
      insert $(R, \mathcal{L}_R - (d, X) + \mathcal{L}_D, P_{\leftarrow}, P_{\rightarrow}, T_R)$ in $m_{i,j}$
    **end if**
  **end for**
**end if**

the case the dependent is temporarily attached to the possibly linear head and a new chart entry is created. Otherwise the head and the dependent are treated the same as in Procedure 5.3 *extend*.

### 5.2.2 Filling Open Slots with Temporarily Attached Words

Another extension necessary to be able to handle discontinuities, is to deal with temporarily attached words. At some point during the parse those temporarily attached words eventually have to fill slots in the word $w$ they have been attached to. Because this is an expensive operation, it is delayed as long as possible. For this reason, filling the remaining slots with temporarily attached words is only done, if the word's positions lists are empty and the word itself could possibly be attached to another word as a dependent. For every open slot the algorithm

**Procedure 5.4** *extend_free* $(D, \mathcal{L}_D, R, \mathcal{L}_R, P_{\leftarrow}, P_{\rightarrow}, T_R, i, j)$ *(discontinuous)*

**if** head can be a linear head **then**
  insert $(R, \mathcal{L}_R, P_{\leftarrow}, P_{\rightarrow}, T_R + (D, \mathcal{L}_D))$ in $m_{i,j}$
**else**
  **for all** $(d, X) \in \mathcal{L}_R$ with $X = D$ **do**
    **if not**($\mathcal{L}_D \neq \langle\rangle \wedge (d, X)$ must be a continuous constituent) **then**
      insert $(R, \mathcal{L}_R - (d, X) + \mathcal{L}_D, P_{\leftarrow}, P_{\rightarrow}, T_R)$ in $m_{i,j}$
    **end if**
  **end for**
**end if**

**Procedure 5.5** *reduce*$(k, i, j)$ *(discontinuous)*

```
1: if T = ⟨⟩ then
2:   insert (R, L, ⟨⟩, ⟨⟩, ⟨⟩) in m_{i,j}
3: else
4:   if L ≠ ⟨⟩ then
5:     for all (d, X) ∈ L do
6:       for all (D, L_D) ∈ T with X = D do
7:         if not(L_D ≠ ⟨⟩ ∧ (d, X) must be a continuous constituent) then
8:           insert (R, L − (d, X) + L_D, ⟨⟩, ⟨⟩, T − (D, L_D)) in m_{i,j}
9:         end if
10:      end for
11:    end for
12:  end if
13: end if
```

attempts to unify every temporarily attached word with the open slot. For every successful unification, the result is stored in the chart. It should be noted that a temporarily attached word need not necessarily be attached discontinuously, that depends solely on the slot it fills. If the temporarily attached word fills a slot which was moved up, it is attached discontinuously. Otherwise it is attached continuously, see Procedure 5.5.

### 5.2.3 Checking for Possible Dependents

Another necessary extension is a method for determining whether a word is a possible dependent, the Procedure 5.6 *check_dep*. As before, all words which do not contain any slots can become a dependent. But in contrast to the continuous

**Procedure 5.6** *check_dep(k) (discontinuous)*

```
Require: k = (D, L_D, ⟨⟩, ⟨⟩, ⟨⟩)
1: if L_D = ⟨⟩ then
2:   return true
3: else
4:   if L_D has continuous empty slots then
5:     return false
6:   else
7:     if Lexical entry of k must be continuous then
8:       return false
9:     else
10:      return true
11:    end if
12:  end if
13: end if
```

case, a word with open slots can become a dependent, if the open slots can be filled discontinuously and the word does not have to be a continuous constituent.

### 5.2.4 Main Parsing Algorithm

The main loop (Procedure 5.7) incorporates all the extensions made so far. Four different cases are considered in the main loop. The first and the second case are similar to those in the continuous main loop. One difference is that more words may be possible dependents, and open slots in a dependent are moved up when the dependent is attached to its head. A dependent may also be attached to its head temporarily. In the first case the dependent is to the left of the head, in

**Algorithm 5.7** *main loop (discontinuous)*

1: **for** $j := 1$ to $n$ **do**
2: **for all** lexical entries $(K_j, \mathcal{L}_j, P_{\leftarrow j}, P_{\rightarrow j})$ of a word $a_j$ **do**
3: Insert $(K_j, \mathcal{L}_j, P_{\leftarrow j}, P_{\rightarrow j})$ in $m_{j-1,j}$
4: **for all** $i, k < j$ **do**
5: **if** $k_1 = (D, \mathcal{L}_D, \langle\rangle, \langle\rangle, \langle\rangle) \in m_{i,k} \wedge \mathit{check\_dep}(k_1) \wedge k_2 = (R, \mathcal{L}_R, P_{\leftarrow}, P_{\rightarrow}, T_R) \in m_{k,j} \wedge P_{\leftarrow} \neq \langle\rangle$ **then**
6: $\mathit{extend}(k_1, k_2, i, j)$
7: **end if**
8: **if** $k_1 = (R, \mathcal{L}_R, \langle\rangle, P_{\rightarrow}, T_R) \in m_{i,k} \wedge k_2 = (D, \mathcal{L}_D, \langle\rangle, \langle\rangle, \langle\rangle) \in m_{k,j} \wedge \mathit{check\_dep}(k_2) \wedge P_{\rightarrow} \neq \langle\rangle$ **then**
9: $\mathit{extend}(k_2, k_1, i, j)$
10: **end if**
11: **if** $k_1 = (D, \mathcal{L}_D, \langle\rangle, \langle\rangle, T_D) \in m_{i,k} \wedge T_D \neq \langle\rangle \wedge k_2 = (R, \mathcal{L}_R, P_{\leftarrow}, P_{\rightarrow}, T_R) \in m_{k,j} \wedge P_{\leftarrow} \neq \langle\rangle$ **then**
12: $\mathit{reduce}(k_1, i, k)$
13: **end if**
14: **if** $k_1 = (R, \mathcal{L}_R, \langle\rangle, P_{\rightarrow}, T_R) \in m_{i,k} \wedge k_2 = (D, \mathcal{L}_D, \langle\rangle, \langle\rangle, T_D) \in m_{k,j} \wedge T_D \neq \langle\rangle \wedge P_{\rightarrow} \neq \langle\rangle$ **then**
15: $\mathit{reduce}(k_2, k, j)$
16: **end if**
17: **end for**
18: **end for**
19: **end for**
20: **for all** $k \in m_{0,n}$ with empty position list and non-empty list $T$ **do**
21: $\mathit{reduce}(k, 0, n)$
22: **end for**
23: **if** $m_{0,n}$ contains an inactive edge **then**
24: **return** true
25: **else**
26: **return** false
27: **end if**

the second case to the right. If temporarily attached words are removed from the words they are attached to, the third and the fourth case come into play. Words are removed from the list $T$ of a word $w$ only if $w$ might be a dependent. Finally temporarily attached words must be removed from chart entries $m_{0,n}$, which span the whole sentence. Otherwise possible solutions may not be found.

# 6 Conclusion and Future Work

Word order phenomena and discontinuity in Hungarian were modelled for a dependency parser. It turned out that, as for the other languages tested, it was possible and easy to write down. Nevertheless there is still a lot of work to do. Up to now only special problems of Hungarian have been modeled as a proof of concept for the parser. Next a full-flexed Hungarian grammar should be developed.

It is most important to add a Hungarian morphology to the parser. Up to now, the parser works with a full form lexicon for Hungarian. This might be a solution for other languages, but it does definitely not work for Hungarian due to the high number of possible word endings.

## Acknowledgements

The authors thank Szilvia Svada and Gabriella Kókai for explaining all the tricky details of the Hungarian grammar.

# References

[1] Norbert Bröker, *Separating surface order and syntactic relations in a dependency grammar*, in Proceedings of the 36th Annual Meeting of the ACL and 17th International Conference on Computational Linguistics, Montreal, 1998.

[2] Michael A. Covington, *Parsing discontinuous constituents in dependency grammar*, Computational Linguistics, 16(4):234-236, 1990.

[3] Ralph Debusmann, Denys Duchier and Geert-Jan Kruijff *Extensible Dependency Grammar: A New Methodology*, Recent Advances in Dependency Grammar, COLING 2004.

[4] Ricarda Dormeyer, *Syntaxanalyse auf der Basis der Dependenzgrammatik*, PhD Thesis, Computer Science, Friedrich-Alexander University Erlangen-Nuremberg, 2004.

[5] Norman M. Fraser, *Parsing and dependency grammar*, UCL Working Papers in Linguistics, 1:296-319, 1989.

[6] Peter Hellwig, *Chart parsing according to the slot and filler principle*, in Proceedings of the 12th International Conference on Computational Linguistics, pages 242-244, Budapest, 1988.

[7] Richard Hudson, *Word Grammar*, Blackwell, Oxford, 1984.

[8] Richard Hudson, *Towards a computer-testable word grammar of English*, UCL Working Papers in Linguistics, 1:321-338, 1989.

[9] Daniel Jurafsky and James H. Martin, *Speech and Language Processing, An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition*, Prentice Hall, New Jersey, 2000.

[10] László Keresztes, *Hungaro Lingua: Praktische ungarische Grammatik*, Debreceni Nyári Egyetem, 1999.

[11] Katalin Kiss, *The Syntax of Hungarian*, Cambridge Syntax Guides, Cambridge University Press, 2002.

[12] Michael C. McCord, *Slot grammar. A system for simpler construction of practical natural language grammars*, in Rudi Studer, editor, Natural Language and Logic, pages 118-145. Springer, Berlin, Heidelberg, 1990.

[13] Ivan Sag, Thomas Wasow and Emily Bender: *Syntactic Theory. A Formal Introduction*, Second Edition, Stanford: Univ. of Chicago Press, 2000.

[14] Lucien Tesnière, Esquisse d'une syntaxe structurale, Klincksieck, Paris, 1953.

[15] Lucien Tesnière, Eléments de syntaxe structurale, Klincksieck, Paris, 1959.

[16] Thomas Tröger, *Ein Chartparser für natürliche Sprache auf der Grundlage der Dependenzgrammatik*, Master Thesis, Computer Science, Friedrich-Alexander University Erlangen-Nuremberg, 2003.

[17] Alexandra Pröll, *Eine Dependenzgrammatik für das Japanische*, Bachelor Thesis, Computer Science, Friedrich-Alexander University Erlangen-Nuremberg, 2004.

[18] Gábor Prószéky, Ilona Koutny and Balázs Wacha, *A dependency syntax of Hungarian*, in Dan Maxwell and Klaus Schubert, eds., Metataxis in Practice, pages 151-182. Foris Publications, Dordrecht, 1989.

[19] Markus Schulze, *Ein sprachunabhängiger Ansatz zur Entwicklung deklarativer, robuster LA-Grammatiken*, PhD Thesis, Computer Science, Friedrich-Alexander University Erlangen-Nuremberg, 2004.