String Generating Hypergraph Grammars with Word Order Restrictions

Martin Riedl[1], Sebastian Seifert[1], and Ingrid Fischer[⋆2]

[1] Computer Science Institute, University of Erlangen–Nuremberg, Germany,
Martin.Riedl@informatik.stud.uni-erlangen.de
Sebastian.T.Seifert@stud.informatik.uni-erlangen.de
[2] ALTANA Chair for Bioinformatics and Information Mining, University of Konstanz, Germany, Ingrid.Fischer@inf.uni-konstanz.de

Abstract. Discontinuous constituents and free word order pose constant problems in natural language parsing. String generating hypergraph grammars have been proven useful for handling discontinuous constituents. In this paper we describe a new notation for hypergraph productions that allows on-the-fly interconnection of graph parts with regard to user-defined constraints. These constraints handle the order of nodes within the string hypergraph. The HyperEarley parser for string generating hypergraph grammars [1] is adapted to the new formalism. A German example is used for the explanation of the new notation and algorithms.

1 Free Word Order in Natural Languages

A prominent difference between natural languages lies the order of words or constituents (groups of words that belong together on a syntactic level) in the various types of sentences. E.g. declarative sentences in English have a fixed word order with the subject first, followed by the verb, the objects and finally prepositional phrases. There are also languages with nearly no order restrictions at all: in Hungarian the excessive use of word endings ensures that sentences can be understood despite the completely free word order. In German, there is a mixture. Declarative sentences have the finite verb in the second position and the infinite verb in the last but one position. Other sentence parts, especially between the finite and infinite part of the verb, can take variable positions. This is a combination of fixed and free word order.

A German example sentence is given in (1). The first line is in German, the second a word-by-word translation into English and the third an idiomatically correct translation:

(1) Die Nachricht wurde durch einen Boten von Marathon nach Athen gebracht der dann starb
The news was by a messenger from Marathon to Athens brought who then died
The news was brought from Marathon to Athens by a messenger, who then died.

⋆ This research was done while Ingrid Fischer was employed at the Chair of Computer Science 2, University of Erlangen-Nuremberg.

Note that the finite verb “wurde” (“was”) is in the second position and the infinite verb “gebracht” (“brought”) in the last but one position. In between these positions, the source “von Marathon” (“from Marathon”), the target “nach Athen” (“to Athens”) and the agent “durch einen Boten” (“by a messenger”) of the action are listed. Their positions are not fixed as shown in (2). All of these variations have the same meaning.

(2)

Die	Nachricht	**wurde**	durch	einen	Boten	von	Marathon	nach	Athen	**gebracht**	der	dann	starb
The	news	**was**	by	a	messenger	from	Marathon	to	Athens	**brought**	who	then	died

Die	Nachricht	**wurde**	von	Marathon	durch	einen	Boten	nach	Athen	**gebracht**	der	dann	starb
The	news	**was**	from	Marathon	by	a	messenger	to	Athens	**brought**	who	then	died

Die	Nachricht	**wurde**	von	Marathon	nach	Athen	durch	einen	Boten	**gebracht**	der	dann	starb
The	news	**was**	from	Marathon	to	Athens	by	a	messenger	**brought**	who	then	died

Several other combinations of agent, source and target are invalid. The source “von Marathon” (“from Marathon”) must appear before the target “nach Athen” (“to Athens”). Otherwise, the meaning of the sentence changes: the messenger seems to be born in Marathon and it is not clear that the message’s transport starts there (3).

(3)

Die	Nachricht	wurde	nach	Athen	durch	einen	Boten	von	Marathon	gebracht	der	dann	starb
The	news	was	to	Athens	by	a	messenger	from	Marathon	brought	who	then	died.

In addition to word order, the running example (1) demonstrates another problem of German syntax: “Bote” (“messenger”) is described more closely through the relative clause “der dann starb” (“who then died”). This relative clause does not follow the noun “Bote” (“messenger”) directly, but is moved to the last position of the sentence (after the infinite verb). Together, the noun and the relative clause form a discontinuous constituent. Discontinuous constituents are separated by one or more other constituents but still belong together on a semantic or syntactic level. This connection between the two parts cannot be expressed with general context–free Chomsky grammars [2].

The desired phrase structure tree for example 1 is shown in Fig. 1.[3] To shorten the representation, triangles are used to indicate that several (terminal) words are generated from one nonterminal symbol. The most important part of the tree is the derivation of the verb phrase (*VP*) into an auxiliary verb (*Aux*), the prepositional phrases denoting the agent (*PP*), the source (*PPS*) and the target (*PPT*) of the action and the infinite verb part, the second participle (*Part2*). The prepositional phrase (*PP*) for the agent contains the discontinuous constituent and is transformed into a preposition (*Prep*), a noun phrase (*NP*) and the relative clause (*RelCl*) in the last position of the example sentence. The root of the tree *S* starts a sentence that is split into a noun phrase (*NP*) and a verb phrase (*VP*).

[3] It is possible to construct a weakly equivalent context–free Chomsky grammar to parse such a sentence, using some workaround for the discontinuous constituent like attaching one of its parts in another production than the other.

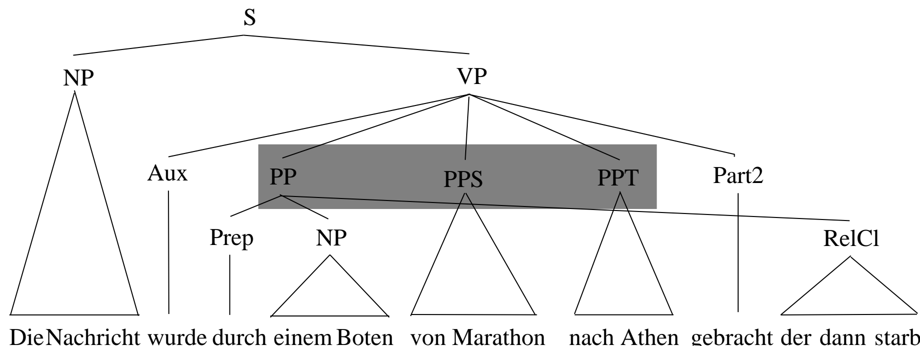

Fig. 1. The phrase structure tree for "Die Nachricht wurde durch einen Boten von Marathon nach Athen gebracht der dann starb." ("The news was brought from Marathon to Athens by a messenger who then died.")

In [3] string generating hypergraph grammars were used to construct such trees based on the the context–free substitution of hyperedges with hypergraphs. To parse sentences with these grammars, an Earley based algorithm, called HyperEarley, was presented in [4]. The Earley algorithm [5] is a well–known $\mathcal{O}(n^3)$ parsing algorithm for context–free grammars that is particularly suited for natural language processing.

Free word order is possible in example (1) between the three prepositional phrases denoting agent, source and target as shown in example 2. These phrases are shaded in Fig. 1. Every variation between the order of *PP*, *PPS*, *PPT* is possible as long as the source *PPS* is mentioned before the target *PPT*.

In this paper, string generating hypergraph grammars and their HyperEarley parser are extended to allow on-the-fly interconnection of graph parts with constraints for free word order and word order variations. In the next section a short introduction into string generating hypergraph grammars is given. Based on the example in (1), word order constraints and their notation within the grammar are introduced in Section 3. The extension of the HyperEarley algorithm is described in Section 4. The paper ends with a conclusion and an outlook.

2 String Generating Hypergraph Grammars

Hyperedge replacement grammars have been studied extensively in the last decades. Introductions and applications can be found in [6, 7]. A subset of hypergraph grammars, context–free string generating hypergraph grammars, are described in detail in [6, 8, 9]. The definitions used in this paper are briefly summarized:

A labeled **hypergraph** (E, V, s, t, l, b, f) consists of finite sets of **hyperedges** E and of nodes V, a source function s and a target function $t : E \to V^*$ which assign a sequence of source respectively target nodes to each hyperedge,

a labeling function $l : E \to \Sigma$ where Σ is a finite alphabet, and sequences of external source nodes b and external target nodes f. A hypergraph's **type** is the pair $(|b|, |f|)$. A hyperedge's type is likewise defined as $(|s(e)|, |t(e)|)$ where $e \in E$. A **string** (hyper-)graph consists solely of hyperedges of type $(1, 1)$ connected via nodes being source to one edge and target to one edge. Only the single external source node and the single external target node re connected to one and not two hyperedges. A string hypergraph represents a linear sequence $\langle s_1 s_2 \ldots s_n \rangle \in \Sigma^n$ of edge labels.

A context–free hyperedge replacement grammar $G = (N, T, P, S)$ consists of the finite sets T of **terminal** edge labels, N of **non-terminal** edge labels, P of **productions** and a start graph S called the **axiom**. A context–free production or rule $p = (L, R)$, commonly written $L \to R$, is a pair of hypergraphs with left-hand-side (L) and right-hand-side graph (R). L is a **singleton** hypergraph, i.e. a graph consisting of a single hyperedge e with external nodes b and f such that $s(e) = b$ and $t(e) = f$. The types of both L and R need to be the same. Each production describes a replacement of a single hyperedge as described by L with the graph R. R's external nodes are merged with L's nodes, respecting order, so that are R is added disjointly (except for the external nodes) to the host graph the production is applied to. A context–free hyperedge replacement grammar is a **string generating hypergraph grammar (SGHG)** if the language generated by the grammar consists only of string hypergraphs. For natural language processing, the hyperedges of a string hypergraph are labeled with the words of the sentence they represent. Σ is the union of words (terminals T) and names of syntactic units (nonterminals N). A string graph's hyperedge labels, read in order from the external source to the external target node, form the underlying sentence.

Please note that for the rest of this paper we assume that the string generating hypergraph grammars are reduced, cycle–free and ϵ–free [2]. If a hyperedge replacement grammar generates a string language, the start graph must be of type $(1, 1)$. A prominent property of such SGHG is that each node is source for at most one hyperedge and target for at most one hyperedge; otherwise no string language will be generated [4].

3 Word Order Constraints in SGHGs

With the help of SGHG, phenomena of discontinuity in natural language can be easily modeled by a context–free grammar. However, modeling the syntactic structure of sentences in which some parts may be reordered freely becomes a tedious task, since the number of productions representing reorderings of n free parts grows with $n!$. Though the ID/LP approach [10] developed for alleviating this problem in classical (flat) context–free grammars cannot easily be transferred to hypergraph based linguistic modeling, it has inspired a somewhat similar notation. The main idea of ID/LP is to distinguish immediate dominance (ID) constraints from linear precedence (LP) constraints. The left hand side of a phrase structure rule (i.e. context–free Chomsky rule) dominates the symbols on

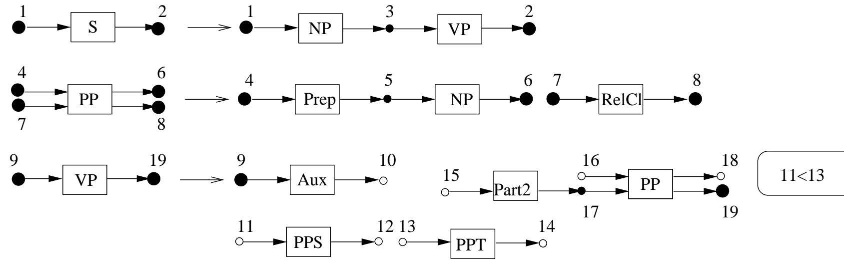

Fig. 2. String generating hyperedge replacement rules with word order constraints for example (1).

its right hand side. The order of the symbols of the right hand side is the linear precedence of the rule. Traditional phrase structure rules incorporate immediate dominance and precedence into a single rule. In contrast ID/LP maintains separate rule sets. In our approach, we also separate immediate dominance and linear precedence, but only within one rule.

This method is described with the help of example (1). The main rules to construct this sentence and its variations are shown in Fig. 2.[4] The first rule replaces S (sentence) with NP (noun phrase) and VP (verb phrase). This rule is identical to the well known $S \rightarrow NP\ \ VP$ in phrase structure grammars. For this rule, no hyperedges or word order constraints are necessary. Numerical node labels indicate the sequence of source and target nodes of the left and right hand side and map external nodes onto each other. External nodes are drawn larger than internal nodes.

The second rule handles the discontinuous constituent "durch einen Boten ... der dann starb" ("by a messenger who then died"). The prepositional phrase (*PP*) has type $(2, 2)$. The rule's right hand side contains the preposition (*Prep*) and the noun phrase (*NP*) leading to "durch einen Boten" ("by a messenger"). These parts have to follow each other in this order.[5] Since this is not the case for the relative clause generated from the symbol *RelCl*, there is no connection between the relative clause and the preposition with the noun phrase on the right hand side. Between both parts other constituents can be inserted.

The third rule has to deal with varying word order. The verb phrase (*VP*) is split into the auxiliary (Aux), the second participle ($Part2$) and three prepositional phrases. The prepositional phrases denoting the grammatical source and target of the verb (PPS, PPT) as well as Aux and $Part2$ have one source and one target node. The prepositional phrase PP for the agent of the verb has two source and two target nodes. This rule is not an SGHG rule if applied literally,

[4] In this paper, hyperedges are drawn as rectangles with their label inside. Nodes are drawn as circles, the connection between nodes and edges is marked with arrows.

[5] Of course in a real world grammar, a rule must be included that splits "einen Boten" ("a messenger") into a determiner and a noun. This rule is omitted here, since it does not offer any new insights.

since there are internal nodes that are either not a source or not a target. These nodes are drawn in white. Such an "illegal" rule represents a set of legal rules with different orderings of their parts based on the white nodes. The additional information "$11 < 13$" indicates a **constraint** on linear precedence of nodes in the final string graph. To write down constraints on freely attachable nodes, the node numbers are reused. For the sake of simplicity in this paper, all nodes have unique numbers even between different rules.

The white nodes are called **open nodes**, the black nodes are **closed nodes**. It is shown in [4] that in SGHG each internal node is source for one hyperedge and target for one hyperedge and each external node is either source to one hyperedge or target for one hyperedge. Otherwise no string language is generated. These nodes are the so–called closed nodes in the remainder of the paper.

For open nodes on the right hand side of productions, these requirements are not fulfilled immediately but inbound or outbound hyperedges are chosen during the derivation. **Open internal nodes** are either source or target node to exactly one hyperedge. This means that open internal nodes lack an outbound or an inbound hyperedge. In Fig. 2 in the third rule $10, 11, 12, 13, 14, 15, 16, 18$ are open internal nodes. $10, 12, 14, 18$ have only an inbound hyperedge whereas $11, 13, 15, 16$ have only outbound hyperedges. **Open external nodes** are not connected to any hyperedge. External nodes define the type of a graph (the right hand side and the left hand side of a rule must have the same type). There are no open external nodes in Fig. 2. All external nodes are closed.

The idea of an open node is to have the possibility to choose between different hyperedges of the right hand side that might connect to that node by another open node. When combining open nodes during the derivation, the various string graph fragments are combined to form a single string graph, i.e. no open nodes are left and no cycles produced. Different word orders can thus be produced. For our running example, this means that, if we regard the finally derived string as a "path" through the intermediate graphs, that the right hand side of the *VP*–production is entered through node 9 and the *Aux* hyperedge. The outbound node 10 is an open node, so we can choose freely from any open node without an inbound hyperedge as a possible successor. Here, we can continue with the nodes $11, 13, 15, 16$. This means *PPS, PPT, PP* or *Part2* might follow *Aux*.

Not all possible word order variations are desirable. In our example, we must make sure that the source of the action is mentioned before the target of the action. This is achieved through constraints over node labels. The constraint $11 < 13$ in Fig. 2 states that the open node 11 must be entered before the open node 13. The constraint $12 < 13$ would have had the same effect. In the generative model, constraints are accumulated during derivation and restrain free combination of string graphs at the end.

In general C is a set of **order relations** (constraints): Let O be the set of all open nodes. An order relation between two open nodes $o_1 < o_2$ with $o_1, o_2 \in O$ means that o_1 must be before o_2 in the final string sequence. The set of order relations applied to an open string graph can be used to create a directed acyclic graph (DAG). The nodes of the DAG are the open nodes. If

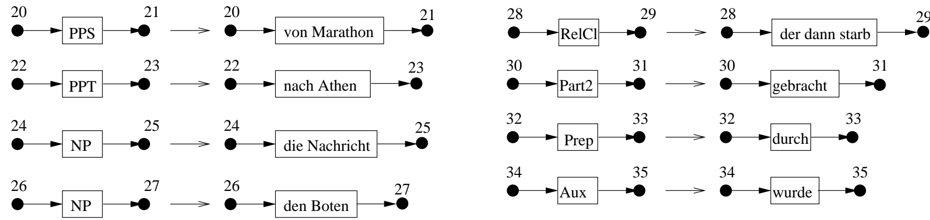

Fig. 3. Missing rules for the analysis of example (1) leading to the derivation tree in Fig. 1.

the set of order relations does not fulfill the criteria of a DAG (e.g. there are cycles inside the graph), the constraints are not well defined. The DAG can be checked for implicit cyclic dependencies that occur through statically connected hyperedges by merging nodes that are statically connected in the open graph [11].

A **hypergraph grammar rule with restriction in word–order** consists of a triple (L, R, C), L and R being the actual rule consisting of a left hand side and a right hand side that may have open nodes, and C being a set of constraints which restrict reordering of the free parts.

For the real–world application of the rules in Fig. 2 to example (1), several productions are missing. Fig. 3 contains the complementing rules to build the example sentence (1). The combination of rules from Fig. 2 and Fig. 3 produces the derivation tree given in Fig. 4 and together they form the example grammar. All sentences given in example (2) are also produced by the grammar. [6]

This section concludes with the definition of the example graph grammar (N, T, P, S) consisting of nonterminal symbols N, in our example all linguistic acronyms, terminal symbols T, here the words of example (1), the productions P given in Fig. 2 and Fig. 3 and the start symbol S, the whole *sentence* in our application context.

4 Parsing SGHG with restriction in word order

In [4] an Earley based parser [5] called **HyperEarley** for string generating hypergraph grammars was introduced. Its main data structure is called a **chart**. For the chart, positions at the beginning of the string, between the words and at the end of the string of words to be parsed are numbered. This numbering scheme is easily transferred onto string hypergraphs. The nodes in the hypergraph are

[6] Please note that it is not possible to generate the following sentence without the discontinuous constituent. Different rules are necessary.

Die	Nachricht	**wurde**	von	Marathon	nach	Athen	durch	einen	Boten	der	dann	starb	**gebracht**
The	news	**was**	from	Marathon	to	Athens	by	a	messanger	who	then	died	**brought**

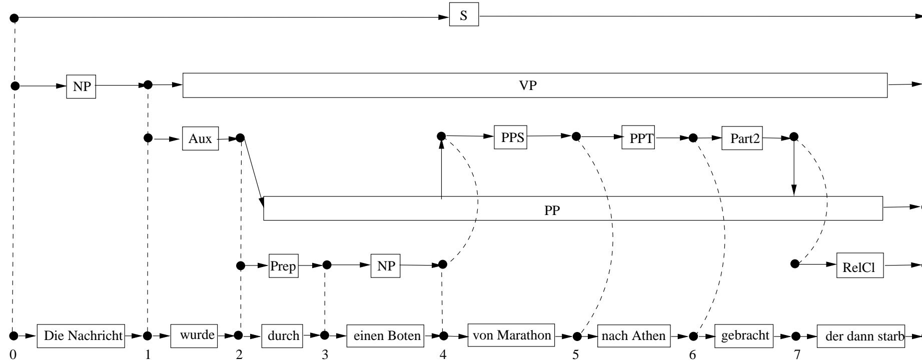

Fig. 4. The derivation tree for the application of the rules given in Fig. 2 to example (1).

numbered from 0 to 8 as done at the bottom of Fig. 4.[7] When parsing a sentence $s_0 s_1 \ldots s_{n-1}$ consisting of n words, the chart is a $(n+1) \times (n+1)$ table. In our running example we have a 9×9 table. In this table, sets of **chart entries** are stored. A chart entry at position (i, j) contains information about the partial derivation trees constructed for the substring $s_i \ldots s_{j-1}$.

Entries are never removed from the chart and are immutable after creation. They consist of information about the currently used grammar production and the progress made in completing the subtree given by this production. This is visualized by so–called **dotted rules**, where the dot marks the parsing progress. The dot is one special node on the right hand side of the rule that marks which parts of the right hand side of the rule have already been found. It is also called **current node**. If the dot is at an external target node of the rule's right hand side, the chart entry is **inactive**. Otherwise it is **active**, i.e. ready to accept a terminal or an inactive chart entry. Note that (different to the classical Earley algorithm) an inactive chart entry must not necessarily be finished; there might still be hyperedges to be processed via another external source node.

The HyperEarley algorithm, like the Earley algorithm, consists of three steps that alternate until the possibilities to apply one of them are exhausted. These steps are **shift** handling terminal hyperedges, **predict** inserting new active chart entries and **complete** combining active with inactive chart entries to generate new entries. When applied to a chart entry e and a hyperedge h that is part of the rule's right hand side e, the method **parts(e,h)** returns either a previously created chart entry describing a derivation of h or null.

The parsing of grammars with open nodes and word order constraints is inspired by [12–14] handling ID/LP grammars with Earley–like algorithms.[8]

---

[7] Please do not confuse the numbers in Fig. 4 (the chart positions) with the numbers in Fig. 2 (the node labels for constraints).

[8] While it is possible to translate SGHG productions with open nodes and word order constraints into productions without open nodes and constraints, this leads to an

Algorithm 4.1 `predict`

Parameters:
e: active chart entry
grammar: the parser's current grammar

1: entry–list← {}
2: node–list ← list of nodes that can be connected to the current node **or** external source node of start symbol's rules.
3: **for all** n in node–list **do**
4: h = hyperedge following n, h labeled with nonterminal
5: **if** parts(e,h) is defined **then**
6: add continuation(parts(e,h), e) to entry–list
7: **else**
8: **for all** rules r where label(lefthandside(r))=label(h) **do**
9: entry–list ← generate–prediction(r,n)
10: **end for**
11: **end if**
12: **end for**
13: **for all** chart entries c in entry–list **do**
14: **if** c is not in chart entry (to(e), to(e)) **then**
15: add c to chart entry (to(e), to(e))
16: predict(c)
17: **end if**
18: **end for**

The main idea is that possible connections of the right hand side's parts are delayed as long as possible. If the connection of an open node cannot be delayed further, all possible connections are handled in parallel.

4.1 Insertion of new chart entries: `predict`

`predict` (see Alg. 4.1) is called with an active chart entry e and is applied whenever new active entries are inserted into the chart.

A parse starts with the prediction of the start symbol S. In this case e is `null`. `node-list` is filled with the external source nodes of the start symbol's rules as there is no current node yet (line 2). In line 4, h equals the S-hyperedge. `parts(`e`,`h`)` is not defined, so for all productions with left hand side labeled S, new predictions are generated in line 9. These predictions are inserted in the chart as new entries in lines 13–15. `to(`e`)` is defined as 0 as e is `null`. Finally in line 16 `predict` is called recursively with the newly generated chart entry. Recursion stops when terminal rules are reached. In our case, rules for NP have to be predicted.

This is an example for the second case for `predict`'s application. A new, active chart entry is inserted if another active chart entry expects a nonterminal symbol. In this case, the first inserted chart entry for S has its current node set

explosion of rules within the grammar and of chart entries, slowing down the parsing process considerably [12–14].

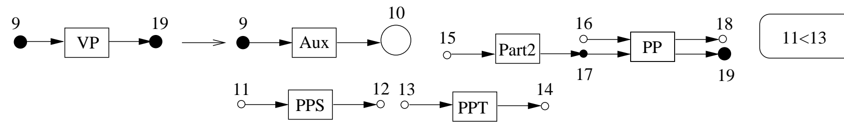

Fig. 5. The large node is the current node in this rule during the parsing process. It remains open for which hyperedge it will be predicted next.

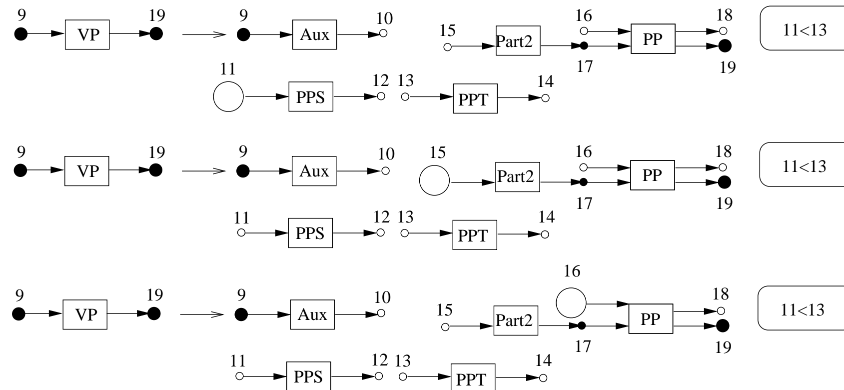

Fig. 6. The large node is the current node in this rule during the parsing process. These chart entries are predicted after Fig. 5.

to node 1 so that chart entries for the hyperedge labeled with the non–terminal *NP* must be predicted. In Fig. 3 two rules for *NP* are given. In line 2 `node-list` is set to $24, 26$. Two new entries are inserted into the chart starting at $(0, 0)$ with current nodes 24 and 26, see Fig 3.

The second case is especially interesting when free word order is possible within a rule. In Fig. 2 this is the case for the third rule. Let's assume the situation given in Fig. 5. Here the current node is at node 10; the auxiliary verb has already been processed. The question is, which new entry is predicted next. As 10 is an open target node, it may connect to every other open source node on the right hand side. Open source nodes are $11, 13, 15, 16$. But the open node 13 cannot be used, because otherwise the constraint stating $11 < 13$ is not fulfilled. In Fig. 6 the three rules for the new chart entries are shown. In Alg. 4.1 in line 2, the possible new current nodes are calculated. In our example, `node-list` contains the nodes $11, 15, 16$. These new chart entries are inserted in the lines 13–15.

In the third case, `predict` inserts **continuation entries**, active entries that restart the parsing of an inactive entry through another external source node. In this case `parts(`e`,`h`)` is not `null` (line 5) but returns another chart entry, signifying that the current node has already been moved over this hyperedge

Algorithm 4.2 `shift`

Parameters:
t, ith-hyperedge in input string graph labeled with terminal

```
 1: entry–list ← {}
 2: for all active entries e where to(e)= i − 1) do
 3:    node–list ← list of source nodes that can be identified with the current node.
 4:    for all n in node–list, n not external target node do
 5:       h = hyperedge following n, h labeled with terminal
 6:       if label(h) = label of hyperedge t then
 7:          entry-list ← generate–new–chart–entries(e,t,n)
 8:       end if
 9:    end for
10:    for all chart entries c in entry–list do
11:       insert c into chart[from(e),to(t)]
12:       if c is inactive then
13:          complete(e)
14:       else
15:          predict(e)
16:       end if
17:    end for
18: end for
```

once. In our running example, the second and third rule of Fig. 2 lead to this situation. The second rule for *PP* can be predicted twice depending on the current node in the third rule. For the first prediction it is at node 16, for the second prediction at node 17.

4.2 Handling edges labeled with terminal symbols: `shift`

The task of `shift` as shown in Alg. 4.2 is to process the ith hyperedge of the input string graph. In line 2 the main loop over all active chart entries e ending at position $i-1$ starts. In `node-list` (line 3), all source nodes of the rule's right hand side in chart entry e that can be identified with the current node are collected. If the current node is a closed node, it is only the current node itself. For an open node, it might be several different (open) nodes. As for `predict`, these nodes are calculated based on the DAG generated from open nodes and word order constraints. The loop over the collected nodes (line 4) determines for each node n the hyperedge h following n. If h matches the terminal t in label and type, e can be extended using t. In line 7 the new chart entry is generated. Finally, all newly generated chart entries collected in `entry-list` must be inserted in the chart (lines 10–17). A chart entry is inserted at the beginning of the active entry `from(`e`)` to the end of the terminal entry `to(`t`)`. If the newly generated entry is inactive `complete` is called, otherwise `predict`.

In Fig. 7 shifting over "von Marathon" ("from Marathon") is shown.[9]

[9] Please note that in our rules given in Figs. 2, 3 terminal and nonterminal symbols are not mixed on the right hand side of a rule. There is only one terminal symbol on

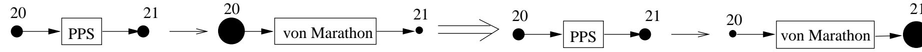

Fig. 7. Shifting in the rule handling "von Marathon" ("from Marathon").

Algorithm 4.3 `complete`

Parameters:
ia, an inactive chart entry

1: entry–list ← {}
2: **for all** active entries e where to(e)=from(ia) **do**
3: node–list ← list of nodes in e that can connect to the current node
4: **for all** n in node–list, n not external target node **do**
5: h = hyperedge following n
6: **if** expects(e, ia) **then**
7: entry–list ← generate–new–chart–entries(e, ia, n)
8: **end if**
9: **end for**
10: **end for**
11: **for all** chart entries e in entry–list **do**
12: insert e into chart[from(ia), to(ia)]
13: **if** e is inactive **then**
14: complete(e)
15: **else**
16: predict(e)
17: **end if**
18: **end for**

4.3 Combination of active and inactive chart entries: `complete`

`complete` handles inactive chart entries ia. For an inactive chart entry, the dot is at an external target node. The inactive entry can be used by an active chart entry to advance its own current node. In line 2 the main loop over all active chart entries e with `to(`e`) = from(`ia`)` is given. If the current node of the active entry is an open node, then (line 3) the nodes that are not external target nodes and can be connected to the current node must be calculated. For all these nodes the following hyperedge h is determined in line 5. The function `expects(`e`,`ia`)` in line 6 is extended compared to the original Earley algorithm. `expects(`e`,`ia`)` determines if a given inactive edge ia is accepted for completion of e. Please note that parsing of an inactive edge is not necessarily finished; an edge is inactive if the current node, the dot, has reached a target node of the rule. There might be several external target nodes. If the label or type of the left hand side of the inactive chart entry's rule differs from e's expected nonterminal edge label or type, `expects(`e`,`ia`)` is false. If the node used to

the right hand side in Fig. 3. This is often the case in natural language applications but not necessarily so.

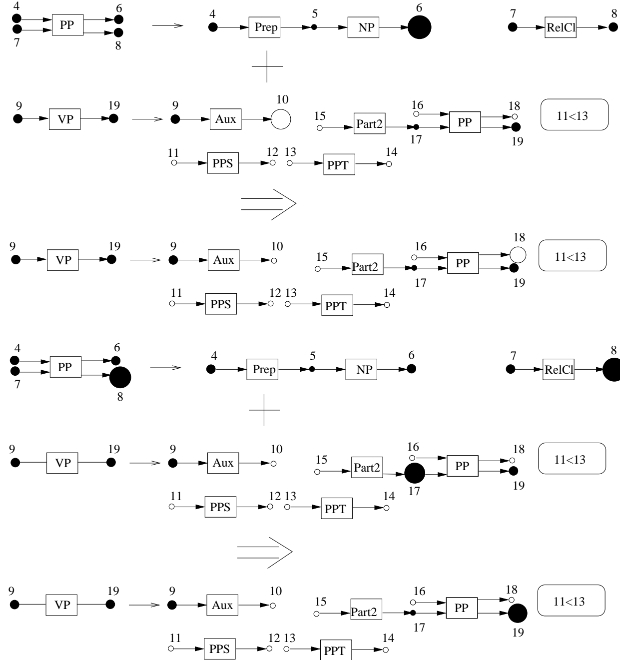

Fig. 8. Completion steps in the running example.

enter *ia* does not correspond to the current node in e, `expects(e,ia)` is false. And if *ia* is a continuation chart entry, but the inactive entry that has been continued does not match `parts(`e`,`h`)`, *ia* represents a different derivation of the hyperedge than the one assumed the last time it was traversed; therefore, `expects(`e`,`ia`)` is false. It is true otherwise. If `expects` returns true, a new chart entry is generated (line 7). As usual in lines 10–16, the newly generated entries are inserted into the chart. If the new entry is inactive, `complete` is called, otherwise `predict`. Deciding whether a chart entry is active or inactive is more tricky than for the usual Earley algorithm. If, after completion, an open internal target node is current, it might be possible to merge it with either an internal source node or with an external target node (depending on fulfillment of constraints and similar consistency considerations). In the first case, the entry would have to be counted as active, in the latter as inactive. Instances of both variants are therefore generated, if necessary. Furthermore, care must be taken not to generate a finished entry, i.e. an inactive entry with no unused external nodes remaining, as long as there are open parts left.

In Fig. 8 two applications of `complete` are shown. In both situations the *PP*–rule is inactive and combined with the active *VP*–rule. They differ in the external target nodes used. For Fig. 8, top, it must be first determined which of the open nodes can be used after node 10. Node 16 is possible. `expects` then

checks whether the label of the inactive chart entry's left hand side and the label of the hyperedge following the current node match. Both are labeled *PP*. Additionally it checks whether the rank of both hyperedges match and whether the same entry node (external source node) is used. In our case it is the first entry node. After `complete` as in Fig. 8, top, is used, the *PP-rule* must be repredicted. Then the completion as in Fig. 8, bottom, can take place after several steps.

Finally a main procedure `parse` is needed taking sentences to be parsed as input, and returning all derivation trees. This procedure is similar to the `parse` procedure in [1]. It starts with `predict` to insert chart entries for the start symbol S. `shift` is called each time `predict` and `complete` do not lead to new entries.

5 Conclusion and Future Work

Extending string generating hypergraph grammars with free nodes and word order constraints has been proven useful to model free word order languages as German or Hungarian. For phrases with free word order consisting of n words, $n!$ phrase structure rules are necessary compared to one rule in the new formalism. Depending on the amount of free word order within a natural language this can lead to huge savings in grammar size.

The extension of the Earley parsing algorithm was based on the the ID/LP Earley parser of S. Shieber [12]. Parsing is in the worst case exponentially in grammar size. The argument in [13, 14] for Shieber's ID/LP parser can easily be transferred to HyperEarley. Nevertheless using formalisms like ID/LP or HyperEarley is useful. In [15] approaches for Head Driven Phrase Structure Grammar based on discontinuous constituents and free word order versus continuous constituents are compared. It is shown, that the former generates significantly less chart entries than the latter approaches. The analysis is based on two large grammars for German.

Several extensions to the parser described are possible. Especially for German it is necessary to address positions in a sentence directly. In a German declarative clause the finite verb is always in the second position and the infinite verb in the last or last but one position. The second position cannot be addressed yet in opposite to the last position. All other parts of the declarative sentence can be moved freely around these two fixed positions. Also there might be one or no element in the last position after the infinite verb. To express this kind of constraints order lists as used in dependency grammars are necessary [16]. Also the ideas of *partially ordered multiset context–free grammars* [17] can be transferred onto SGHG to allow for more descriptive power.

References

1. Seifert, S., Fischer, I.: Parsing String Generating Hypergraph Grammars. In Ehrig, H., Engels, G., Parisi-Presicce, F., Rozenberg, G., eds.: 2nd International Conference on Graph Transformations (ICGT04). Number 3256 in Lecture Notes On Computer Science, Rome, Italy, Springer-Verlag (2004) 352 – 267

2. Jurafsky, D., Martin, J.H.: Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition. Prentice Hall (2000)
3. Fischer, I.: Modelling Discontinuous Constituents with Hypergraph Grammars. In Pfaltz, J.L., Nagl, M., Böhlen, B., eds.: 2nd International Workshop on Applications of Graph Transformation with Industrial Relevance (AGTIVE'03). Number 3062 in Lecture Notes in Computer Science, Charlottesville, USA, Springer Verlag (2005)
4. Sebastian Seifert: Ein Earley-Parser für Zeichenketten generierende Hypergraphgrammatiken. Studienarbeit, Lehrstuhl für Informatik 2, Universität Erlangen-Nürnberg (2004)
5. Earley, J.: An Efficient Context–Free Parsing Algorithm. Communications of the ACM **13** (1970) 94–102
6. Habel, A.: Hyperedge Replacement: Grammars and Languages. Number 643 in Lecture Notes in Computer Science. Springer-Verlag, Berlin (1992)
7. Drewes, F., Habel, A., Kreowski, H.J.: Hyperedge Replacement Graph Grammars. In Rozenberg, G., ed.: Handbook of Graph Grammars and Computing by Graph Transformation. Vol. I: Foundations. World Scientific (1997) 95–162
8. Engelfriet, J., Heyker, L.: The string generating power of context-free hypergraph grammars. Journal of Computer and System Sciences **43** (1991) 328–360
9. Engelfriet, J., Heyker, L.: Context–free hypergraph grammars have the same term-generating power as attribute grammars. Acta Informatica **29** (1992) 161–210
10. Gazdar, G., Klein, E., Pullum, G., Sag, I.: Generalized Phrase Structure Grammar. Harvard University Press (1985)
11. Martin Riedl: Wortstellungsrestriktionen für Zeichenketten generierende Hypergraphgrammatiken. Studienarbeit, Lehrstuhl für Informatik 2, Universität Erlangen-Nürnberg (2005)
12. Shieber, S.M.: Direct Parsing of ID/LP Grammars. Linguistics and Philosophy **7** (1984) 135–154
13. Barton, G.E.: On the Complexity of ID/LP Parsing. Computational Linguistics **11** (1985) 205–218
14. Barton, G.E.: The Computational Difficulty of ID/LP Parsing. In: Proceedings of the 23rd conference on Association for Computational Linguistics July 08-12. (1985) 76–81
15. Müller, S.: Continuous or discontinuous constituents? a comparison between syntactic analyses for constituent order and their processing systems. Research on Language and Computation, Special Issue on Linguistic Theory and Grammar Implementation **2** (2004) 209–257 http://www.cl.uni-bremen.de/ stefan/Pub/discont.html.
16. Barta, C., Dormeyer, R., Spiegelhauer, T., Fischer, I.: Word Order and Discontinuities in a Dependency Grammar for Hungarian. In Zoltan, A., Csendes, D., eds.: Proceedings of the 2nd Conf. on Hungarian Computational Linguistics (MSZNY), Szeged Hungary, Juhasz Nyomda (2004) 19–27
17. Mark-Jan Nederhof, G.S., Shieber, S.: Partially Ordered Multiset Context-Free Grammars And Free-Word-Order Parsing. In: 8th International Workshop of Parsing Technologies (IWPT), Nancy, France (2003)