Aligning mildly context-sensitive formalisms for data-driven parsing

ILYA KASHKAREV

Master Thesis
Universität des Saarlandes

25th September 2012

Declaration

Eidesstattliche Erklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

Declaration

I hereby confirm that the thesis presented here is my own work, with all assistance acknowledged.

Saarbrücken, 25th September 2012

Signature: Ilya Kashkarev

Acknowledgements

I am grateful to my supervisor in Saarbrücken Dr. Yi Zhang for constant help and consultation. He helped a lot in directing my thoughts in a way that led to fruitful results.

I would like to thank my supervisor in Nancy Dr. Sylvain Pogodalla for inspiring my interest in scientific work and very intensive and deep discussions.

Last but not least, I am indebted to my relatives for support and to my dear friend Olga Nikitina for all the other good things that happened to me during the work on this thesis.

Contents

1 Introduction

In the current research we investigating the relations between formalisms with different levels of deepness of linguistic knowledge integrated in them. On the far end of this scale we may put context-free grammars (CFG), as linguistically rather trivial formalism. In its simplest form a CFG assigns categories, or tags, to certain continuous substrings of a sentence called constituents. The analisys of a sentence with a CFG is a local tree, whose nodes correspond to constituents, and the mother node is a concatenation of the substrings of the daughters. Locality practically means that the daughters of every constituent are, so that their projection on the string are a sequence of adjacent substring with the same order.

CFG is a very nice and simple formalism. A number of fast and practical algorithms were developed for CFG parsing, which allow to parse in polinomial time with moderate constant parameter [64, 19]. The underrepresentation of linguistic knowledge, with only one parameter, the category of a node, allows for coarse parsing with good quality. As a more general term, context-freeness means that the derivation of a current node, or the application of the current rule, does not depend on the context, i.e. the history of the derivation, as well as other neighbouring rules. This provides very concise representation of partial analises.

A probabilistic context-free grammar (PCFG) is a CFG grammar with probabilities assigned to each rule of the grammar. A grammar can easily be read from a treebank and the probabilities computed on the basis of the frequencies of the rules in the treebank. There are a number of big treebanks that one can use (Penn treebank [42], TIGER corpus [3]).

The formalism, though, has a few disadvantages. First, long-distance dependencies may not be directly, if at all, analized by CFG. The notion of locality is very restricted here. In a single grammar rule one may only represent direct daughters of a node and no other constituent is permited to inteleave. The dependancies may still be encoded into treebanks, but they cannot be modelled on the regular basis by a grammar. Not handeling long-distance dependencies is of course one of the features that makes CFGs so effecient to parse with. More complex grammars need good mechanisms to filter possible dependencies.

Discontinuous constituents is a notion that makes it possible for a constituent to consist of more than one continuous segments. A local tree cannot have discontinuities. It has recently been argued that the phrase stucture component should be loosened to be just ordered trees, rather than local tree. This point of view is strongly supported by empirical evidence.

Indeed, even such a strict word order language as English exposed significant amount of discontinuous phrases, not to speak of other more flexible languages.

Although it is difficult to prove whether natural languages are not context-free, evidences from linguistic studies have indicated that a richer formalism would better capture the mechanism of natural languages. The class of languages which is considered a proper class for modelling natural languages is mildly context-sensitive languages. A good formalism to analyze the language must generate context-sensitive languages. In particular, it must be capable of generating context-free languages, parsing must be possible in polinomial time and it must be able to analyze so-called cross-serial dependencies. These are so to say word order variation, proven to be non-context-free, which appear in natural languages.

Linear context-free rewriting systems (LCFRS [62]) and abstract categorial grammars (ACG [12]) are examples of this sort of formalisms. LCFRS is a rather straightforward generalization of CFG. One also allows category labels to contain additional information to assist parsing. The grammars are linear in the sense that every rule may only pass the information up in the derivation in any order or combination, nothing can be duplicated or deleted. Thus, the formalism actually unites a class of formalisms with different interpretations of operations allowed on the information. We can for instance use the parse trees as features of nodes and join these trees by the grammar rules.

A particularily important interpretation of LCFRS is when non-terninals (nodes) are assigned a list of variables which stand in one-to-one correspondence with continuous segments spanned by this non-terminal. Therefore, we have an integrated direct representation of discontinuities. The application of a rule may, then, concatenate these variables or list them separately, depending on whether the corresponding segments are adjacent. The simplisity of the generalisation made leaves the possibility to use only slightly modified versions of many parsing algorithms.

Having these generalizations makes it possible to directly read grammars from treebanks with crossing branches. And this type of representation has an advantage of clearer subcatigorasation frames' representation as well as tighter syntax-semantics connections. We obtain more informative rules and they in turn are better motivated. Having an overt representation of discontinuities in a grammar allowes us to have direct control over the structures and phenomena which have always deteriorated parsing efficiency as well as accuracy.

The new but still simple machinery allows LCFRS to go behind context-

freeness but the parsing complexity remains polinimial. The current research shows that the parsing speed is comparabale to that of CFG. It lies within the same range, but the systems are not easily scalable as opposed to CFG. The parsing times grow because discontinuous nodes if unrestricted may span over any possible subset of words. The more variables a non-terminal has, the larger the number of all possible subsets that it can span over becomes. There are theoretical results of how to minimize these numbers but they are hardly linguistucally motivated.

Concerning the accuracy of LCFRS parsing, it was also shown to be very close to that of CFG [29]. Several existing implementation were trained on large corpora, they manage to parse sentences of more than average length. Although the main aim of the generalization made is to handle discontinuities, we have not seen any detailed investigation of how LCFRS manages to process them. One of the first intentions we put into this work is to give a comparative analysis of the accuracy results on continuous and discontinuous data, on the sentence level and also node-wise. The analises should better reveal the impact of the introduction of discontinuities. We can actually observe whether they are captured and how many of them are missing. These observations help understand the specificity of the formalism and to integrate appropriate modifications.

Most of the heuristics to improve LCFRS come from already existing CFG methods. We are rather interested here in how the formalism can benefit from other formalisms with more complex linguistical foundations. To this end, we implement a LCFRS parser and seek ways to introduce more linguistically motivated features, not just probability smoothing or other usual CFG techniques to improve parsing results. At the same time the formalism itself offers certain places for improvement. We make an attempt to approach these inefficiencies. However, before that, we want to have theoretical support. It would be critical to observe to which extend LCFRS allows for the integration of deeper linguistic knowledge.

LCFRS is only one of the class of formalisms generating mildly context-sensitive languages, like, for example, the tree-adjoining grammars. A very flexible and general linguistic platform to encode other formalism into is abstract catgorial grammars (ACG). ACG is a type of categorial grammar, therefore, it is more logic-oriented. Words and constituents are understood as sort of logic formulae. They are operated on by logical operations to obtain an analisys of a sentence. Although the two formalisms initially have so diverse bases, a subclass of ACGs was proved [15] to generate the same string languages as LCFRS, as opposed to tree languages. Knowing this we conduct our theoretical research on ACGs, because they include LCFRSs,

are capable of encoding more complex machinery, and at the same time retain the linearity property which is similar to the linearity of LCFRS.

ACG is a relatively new formalism. It was designed as a platform to encode other formalisms and investigate their properties. And we use it as the platform for our attempt to try on more complex linguistic material. The class of all ACGs is divided into subclassed according to the complexities of the formulas used. The first non-trivial subclass consisits of second-order ACGs and they as we already mentioned correspond to LCFRS. There is not much information known about parsing with ACGs but it was shown that with second order ACGs one can parse in polinomial time. Other results suggest that with higher order ACGs polonomial parsing is impossible. Parse trees here are also local trees and they drive the analisys, however, the operation of λ-abstraction helps arrange the words, or the formulae, in any order. Wrong types of formulae restrict possible combinations. LCFRS, though, have all the possible applications listed in the form of rules. Typing of formulae makes ACG more appealing as the target formalism of a formal conversion of a type-based formalisms, like HPSG or LFG, which we are going to discuss soon.

The ACG grammars are carefully hand-written, thus, their linguistic precision should be better. But we have not seen any large scale experiments with reported results. The grammars are not yet generalized to probabilistic parsing as well. However, as other categorial grammars they seem to have better potential in handling word order phenomena. New research shows that coordination, different kinds of movements, cross serial dependencies are processed with relative easiness.

Although the formalisms we mentioned are already much more powerfull, than simple context-free grammars, there are other, more complex models of natural language. Head-driven phrase structure grammar (HPSG [51]) is a prominent linguistic theory capable of presenting deep linguistic knowledge. It proves to model a large variety of linguistic phenomena and has a very sophisticated machinery. It has very precise formal theory resulting from its strict commitment to formal representation of linguistic knowledge.

The basic structures used in HPSG are feature structures, which are collections of attribute-value pairs. They are designed to describe linguistic types, generalization of linguistic entities. Feature structure attributes, or features, correspond to properties of linguistic entities or their substructures, e.g. a sentence has a feature of subject. The values of the attributes are other feature structures. The basic empirical assumption is that a natural language is a system of linguistic types. This types should be modelled by a grammar and the proper representation is feature structures. Being a

constraint-based theory, HPSG is driven by a set of principles which validate certain feature structures and rule out all the others.

HPSG operates on feature structures, notions of subsumption and unification of those, but the phrase structure skeleton in HPSG is given a special role. It drives the derivation and usually is required to be a local tree. Therefore, from its origin the formalism neglected discontinuous phrases. For many fixed word order languages HPSG has achieved reasonable results without loosening this presidence constraint [8]. But for other languages, it becomes very impractical to operate on precise linguistic terms of artificially created phrases.

There are modifications to the formalism or grammars that allow to have dicontinuities. However, HPSG is still relatively slow if compared to other, simpler formalisms e.g. context-free grammars, especially if allowing discontinuity in phrases. The source of this complexity lies in too powerfull feature structure mechanisms which permits to copy structures unbounded in size, and poor interface between them and the phrase structure mechanism. The latter, though is explicated in lexical functional grammars. But in the current research we concentrate on a particular case of feature structure based formalism, HPSG, because it has very formal mathematical foundations, which is very helpful to conduct our theoretical investigation.

HPSG was aimed at good accuracy. Most of the grammars are hand written, the phenomena covered are usually revealed to the maximal thoroughness, writing a good coverage HPSG grammar takes years. The amount of data processed by an HPSG parser is huge and it is not obvious that all these data are really necessary. The machinery may be too complex to characterise the ususal human usage of the language. This point of view is supported by the results stating that in general case HPSG feature structure recognition may be undecidable, i.e. we are not able to say whether a given sentence type is in the language [32].

So, we observe that the more complex the linguistic material we can process, the worse the preformance usually goes. Here we would like to explicate the relationship between the more complex models of the language and less comlex models on the examples of HPSG, LCFRS, and ACG. There can be potential benefits from both sides. The complex HPSG structures may theoretically be simplified to borrow some effeciency from simpler ones. At the same time, the formalisms like LCFRS seem to suffer from the lack of more linguistically approved information and can be enriched to support the processing of new information.

On the theoretical level we construct a partial convertion of HPSG into ACG. In fact we use hyperedge replacement grammars (HRG) as the target

of our convertion, but some very tight correspondence of HRG and ACG was already proved before. Moreover, the complexity results suggest that if we try to build such a conversion in a more general fashion, then the second-order ACG would not be sufficient. But higher-order ACGs are not yet well-studied. It seems also clear that there may not in principle exist a fair conversion of $HPSG$ into HRG, basically because head-driven phrase structure grammars do not have context-free derivations. Thus, in this work, we try to establish conditions, possibly not too restrictive, under which a grammar in HPSG formalism may be modeled as an HRG.

Certain extensions to the base ACG formalism were proposed before [14]. These extensions can represent feature structures with a few extra primitives. However, we are rather interested in the expressive power of the unextended formalism, because the extentins are new and too powerful. Moreover, we cannot directly relate them to the LCFRS formalism.

The practical side of our investigation concerns LCFRS as there is ongoing research on LCFRS applications. We enrich the formalism with additional external information to improve parsing. As the target language we chose German. There are big availiable German corpora which one may use as the training data. German has variable word order patterns and it is very difficult to obtain good results within formalisms restricted to local trees. Frequently occuring discontinuities present lots of material to work on.

We implement an LCFRS parser for German, train and test it on one of the largest corpora. The parser, used with a context-free grammar does the usual probilistic context-free parsing. It provides us with a baseline to compare our results to. We also compare the results to other current state-of-art parsers for German. Our theoretical results and error analysis guide us to introduce modification to the parser.

2 Background

2.1 German and relatively free word order

The German language has relatively free order. This makes processing of German language data a challenge for many linguistic formalisms. Finite verbs in a German sentence take either second position — in the main clause (1) — or final position in a relative clause (2).

(1) *Karl kauft das Haus in Hamburg*
Karl buys the house in Hamburg

(2) *dass Karl das Haus in Hamburg kauft*
that Karl the house in Hamburg buys

In yes-no questions or in imperative sentences, though, German finite verbs hold to the first position. Separable particles and finite verbs are exclusively placed in the final position (see for instance Crysmann [9]).

Traditionally, German clauses are structured into topological fields [26]. These are relative positions in a sentence, which may even be incoherent segments of the sentence. A clause in the German language may be split al least into Vorfeld (pre-field), Linke Klammer (left bracket), Mittelfeld (middle field), verbal complex field, and Nachfeld (post-field). German word order principles are based on these positions in a sentence. They identify the head of the clause and all its components. The Vorfeld is usually the topic of the sentence of the verb-second type of sentences. The left bracket is the position of the finite verb in verb-initial and verb-second sentcence types. Mittelfeld is located between the left bracket and the verbal complex and it is reserved for all kinds of verb complements and modifiers. All non-finite verbs and the finite verb in the verb-last sentences go to the verbal complex which is then followed by the Nachfeld, which contains some extraposed relative clauses or postponed prepositional phrases. Topological field chunking, i.e. segmenting sentences into chunks corresponding to topological fields, turns out to be fruitfully used as a preprocessing step for parsing. Results have been reported with the quality approaching the results of part of speech taggers. Several approaches to chunking were compared for instance in [61].

Another instance of uncertainty in the German word order is scrambling, which manifests itself in the Mittelfeld. The field is typically reserved for complements and modifiers and they may often be shuffled in various combinations as well as they can actually be subcategorized by different verbs and interleave with modifiers of these verbs. The reasons for this uncer-

tainty may not strictly belong to syntax, rather phonology or pragmatics. Consider the following typical examples of scrambling:

(3) a. *dass der Mann der Frau das Buch gab*
b. *dass der Mann das Buch der Frau gab*
c. *dass das Buch der Mann der Frau gab*
d. *dass das Buch der Frau der Mann gab*
e. *dass der Frau der Mann das Buch gab*
f. *dass der Frau das Buch der Mann gab*

All six variants in (3) of the German sentense for 'that the man gave the book to the woman' are acceptable. Thus, we either do not order the subcatigorization list of the verb 'gab' or have to introduce discontinuous constituents.

Extraposed relative clauses occur frequently in German and many other languages. Their location is rather flexible. They seem to be triggered by the speaker to avoid prosodically heavy center embeddings. Moreover, the range of possible antecedents is not easily recognized [10]. Thus, they also form discontinuous constituents and are causes of various parsing errors.

2.2 Treebanks

Development of extensive corpora with good quality favours research in linguistics as well as machine learning methods in natural language processing. One of the biggest availiable German treebank is the TIGER corpus [3], approximately 55k sentences, with its predecessor the NEGRA corpus, 20k sentences. The treebanks contain sentences from the newpaper *Frankfurter Rundshau*. They are provided with part-of-speech and morphological annotation, phrase structure, as well as grammatical functions and secondary relations which are only used to mark coordination phenomena. The corpora use Stuttgart Tubinger POS tag set. The corpus annotation allows discontinuous phrases. Discontinuities are very frequent. One can observe that in treebanks like NEGRA or TIGER about 25-30 percent of sentences are discontinuous [38]. An example of a tree from the TIGER corpus is given in Figure 1.

Tubinger Treebank of Written German (TüBa-D/Z) is another important corpus of German, which contains about 60k sentences from the newspaper *Die Tageszeitung*. However, unlike TIGER, it does not contain crossing branches and uses topological field annotation in combination with functional labels to encode long-distance dependancies. Thus, its trees are

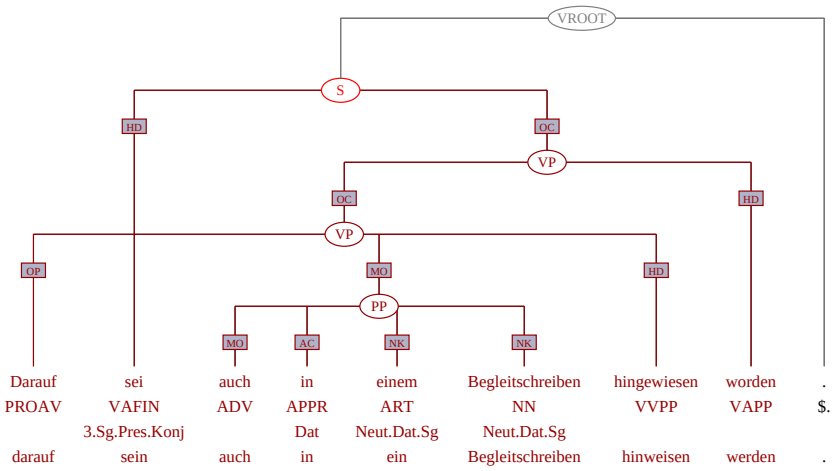

Figure 1: A tree from the TIGER corpus

deaper than the rather flat TIGER trees. The TIGER trees contain on average at least twice as many nonterminals as TüBa-D/Z trees [38].

2.3 Parsing of German

The treebanks are rather big and previous research shows that they are sufficient to create grammars that achieve a fair level of accuracy. Corpus-based large scale parsing of German was conducted in vatious formalisms.

2.3.1 Head-driven Phrase Structure Grammars

In the VERBMOBIL project [45] an attemnpt to model German the a large scale linguistic grammar in the framework of HPSG was made. The grammar was complemented with data specific to Verbmobil domain of scheduling dialogs and particular attention was paid to spontaneous speech analysis. The grammar has not been changed for a while. Exessive number of rules for many complex constructs make the grammar difficult to maintain. The grammar is able to parse about 82% of sentences of the 25k sentences from several VERBMOBIL CDs. Among the phenomena which account for most errors there are coordination and extraposition.

However, the grammar is not currently actively maintained. Due to complicated German word order priciples the grammar contains a number of dubious constructions, it became hard to conceive and maintain. What we can observe is that usual continuous phrase-structure parsing becomes hard with the growth of grammars. Neither it show good results on new unseen data of the general domain texts. [66] shows that the coverage of the grammar on sentence of the *Frankfurter Rundschau* newpaper is only about 30%.

The Babel-System [43] is a natural language understanding system for German written in Prolog. It is availiable online, processes morphological, syntactical and semantical information in the framework of HPSG. The system can parse 74% of sentences in a part of VERBMOBIL. The grammar is limited to verbs, nouns, adjectives and determiners and contains 10,500 stems, however covers a wide range of phenomena. Morphological processing is done offline.

An approach directly using the concept of discontinuous constituents is taken while implementing the Babel-System. The parser is designed in the form of a bottom-up chart parser. A word is taked from a given string and a coresponding lexical item is retrieved, then all the daughters of each rule are tried to unify with it. If the unifications succeeds, then the other daughter is searched for. Finally, the result is added to the chart. Exploitation of binary vectors denoting the span of the current edge in the chart provides correctness and limits the search space.

The Babel project shows that it is possible to parse directly discontinuities in HPSG, but it is far less efficient than for instance the PET parser, which assumes continuity. It is a state of art parser for HPSG grammars, which uses the same grammar formalism implementation as the LKB system. But unforturately, only continuous constituents are permited.

Discontinuous consituent potentially can span over every subset of words in a sentence, which amounts to an exponentioal number of possibilities relative to the length of the sentence. And indeed if parsing discontinuities without restriction on where they can arise makes the chart explode. In [44] a few restriction were given of the phrasal elements which may allow discontinuities in terms of their subcategorization lists. It was shown that, in the Babel system, the chart size can be reduced at least by half and the parsing time decreases by factor of three. The principle used roughly prohibits non-extraposed saturated phrasal signs to be discontinuous and otherwise allows no more gaps than the length of the subcat list plus one. Usage of this principe has a very strong impact on the performance, because there are no other essential control over gaps in HPSG.

As every grammatical theory which is able to apply deep linguistic knowledge, in practice HPSG is much slower than for instance simple context-free formalisms. To provide a trade-off between practicality and richness it may be important to establish connections between languages generated by HPSG and those of context-free grammars. An attempt to approximate HPSG with CFG was done, for example, in [33], where the authors do an approximation of HPSG for Japanese by looking at what HPSG rule-types take as input and what their output is.

Another successful PCFG approximatiom of HPSG, this time for English, was done in [65]. The authors extact the grammar from an atomatically parsed corpus. There are several corpora automatically parsed with the English Resource Grammar (ERG), from whose derivation trees the target PSCFG grammar is extracted. The F-score of up to 84,13% is achived, which ever exceeds the ERG parsing quality.

2.3.2 CFG

Context-free grammar is one of the most well-studied that is suitable for parsing natural language sentences. Thus, it is not surprising that many attempts were made to approximate different formalisms by context-free grammars, with or without additional pre- and post-processing.

Several papers have reported on their parsing results on the NEGRA Corpus. [38] achieves PARSEVAL F-measure of 68.94% whith raised to 76.18% after modification with topological fields information. They also discussed how various annotation schemes influence results. In general their left-corner parser performed significantly better on TüBa-D/Z with the F-score over 80%. The comparison suggests that more structured phrases are helpful, and at the clause level the usage of flat structures incorpotating topological fields is advantgeous.

A hierarchically split PCFG with Berkley parser [48] achieved the F-score of 80.7 on the sentences of length ≤ 40. They use a coarse-to-fine technique, when the categories are approximated by more rough ones and a 'filter' of rougher context-free grammar applies to get $n-$best parses to be then re-parsed by finer grammars. This method significantly improves performance without any loss in quality.

Rafferty and Manning [53] reported 79.67% F-score with the Stanford parser (On the first 20k sentences of the TIGER corpus). They reported the influence of markovization, i.e. adding of vertical and horizontal context to nodes of a tree. Adding up to two levels of vertical context positively influences the performance, adding extra 2% both for the TIGER corpus and

TüBa-D/Z. Horizontal context does not yield any gain of accuracy, and has a degrading tendency because of the sparseness of the data. Using gold part of speech tags has positive effect of approximately 3%. Another measure useful for English, state split, does not cause any significant improvement. The authors also investigated a possibility to add information about grammatical functions. Though, due to sparsity, neither efficiency nor accuracy improvement was gained. In the experiments only sentences of length less than 40 words were processed.

In [18], a PCFG parser with sister-head dependencies showed 75% F-score and they also report decrease of coverage with the introduction of grammatical functions. The learning curve suggests that the training set is sufficient to gain better results with PCFG and that the flatness of the corpus may be the reason that many found constituents have wrong boundaries. It is also the reason why the sister-head dependencies performs better in their experiments

Lexicalization of grammars shows a controversial effect for German. It turns out to be very helpful for English [6], but its effect on PCFG for german is not that significant [53] or even negative [18].

In general, we can see that the German PCFG results are considerably worse than English (around 90% F-score on Penn treebank). Some inconsistencies may appear when we convert non-local trees to local. If we conduct propabilistic context-free parsing, then the learning material, which may contain crossing branches, must me transformed into continuous trees as we discuss in Section 3.2.6. All used methods are solely superficial approximations. They try to interfere with the analisys as little as possible, to obtain a CFG analysis of the same string.

2.3.3 LCFRS

To extract a LCFRS grammar from a corpus with crossing branches a simple algorithm was proposed in [41]. The algorithm is based on the idea to interpret the trees in the treebank as derivation trees of a LCFRS. Every node of a tree represents a non-terminal whose variables are continuous substrings of the sentence. It allows to explicitly extract large-scale grammars from corpora allowing crossing branches.

The first actual implementation was done by Kallmeier [29]. The authors read their grammar from the NEGRA corpus. The parser is basicly CYK parser where at each point in time the best item from the agenda is chosen and combined with all possible items in the chart. The production results are then again added to the agenda. They use different estimates to chose

the best item from the agenda. Those estimates speed up parsing up to four times. The paper shows that data-drived LCFRS parsing is feasible and the results (F-score of approximately 70 percent) are competitive with PCFG parsers. The sentence length is limited to 25 words, due to memory overload.

Maier [39] extends the training set to the TIGER corpus and also the sentence length to 30 words. He reports 71.5% and 73.5% labeled F-score for the two corpora. The total recall of parsed sentences differs only insignificantly from that of CFG. In comparison to their plain model CFG the accuracy of LCFRS is slightly higher. The unlabeled attachment score of the dependency evaluation is rather low, 77-78%, probably because of a naive dependency extraction algorithm. Accuracy on a few phonomena, such as forward conjunction reduction or PP attachment, was manually analysed. In general words, the phenomena are harder for LCFRS, than for instance for dependency parsing or PCFG.

Parsing sentences with a simple LCFRS longer than 30 words is still difficult because of computational complexity. Therefore usage of an approximating CFG was proposed with parsing in two rounds [60]. It is an instance of coarse-to-fine approach to parsing. It allowed to parse NEGRA-40 with speed on average one sentence per second, the parsing time curve suggesting that with a CFG approximation systems are much better scalable.

2.3.4 LFG

German was subject of LFG parsing experiments. It is worth mentioning that in the first place the treebank annotation was assisted by an LFG parser [17]. That grammar could achieve 50% of coverage. Later, a more impressive F-score of 84.2% was achieved in [57] with the coverage of over 86%. Their grammar is hand-written, and they resort to stochastic parse disambiguation. A grammar automatically extracted from corpus was presented in [55]. They achieve 73% F-score on the f-structure and 79% on the $c-$structure evaluating against a hand-written gold standard.

2.3.5 CCG

Combinatory categorial grammars (CCG) is a formalism based on logical deduction. The analysis of a sentence is a deduction tree. Several operators are used to combine phrases, e.g. composition, type-raising, or application, which substitute $\lambda-$abstraction. CCG is mildly context sensitive and is weakly equivalent to TAG. Hockenmaier in [25] constructed a TIGER-based

CCG grammar for German. They modify the trees, so that they become CCG derivation trees. They manage to translate 92% of the treebank to a CCGbank. The average lexical coverage on 10-fold cross-validation is 86%. In comparison the English CCGbank has a the coverage of 94% and the grammar is two times smaller.

2.3.6 Dependency parsing

The absence of hirarchical structure, or grouping of words, allows dependency parsing to obtain better results on long-distance dependencies. Indeed, a detailed comparative analysis of the two was given in [36] for the cases of coordination and extraposed relative clauses, and dependency parsing copes with the task considerably better.

Using MaltParser the TIGER corpus and TüBa-D/Z were parsed in [24]. Dependencies were extracted from the corpora and used to train the parser. The non-projectivity, or discontinuity was encoded into the arc labels. The label attachment scores are 91% for TIGER and 88.6% for TüBa-D/Z. They retained the possibility to return back to constituents and, hence, allowing for a close comparison with constituency parsing. It resulted in 65% and 75% respectively (taking into account grammatical functions).

2.4 Theoretical investigation

Head-driven phrase structure grammars (HPSG), introduced in [51] is a constraint-based linguistic framework. It is capable of modeling a large variety of linguistic phenomena and has a very sophisticated formalism based on the typed feature structures. But at the same time in its original form the phrase structure backbone is restricted to local trees. HPSG can benefit in this respect from other simpler formalisms with, however more flexible phrase structure component.

Recent works on linearization of HPSG attempt to loosen the linear precidence constraint in HPSG. Works of Reape starting with [54] introduce domains of signs, which are basicly sequences of signs one can manipulate on. Kathol [31] resorts to German topological fields to account for the clausal structure. The dissertation of Michael Daniels [11] provides an platform for processing linear precedence and dominance constraints for HPSG.

In [32] it is shown that head-driven phrase structure grammar theory, considered as a theory of the monadic second-order logic (MS_2) is in general undecidable. This result suggests that there should not be a faithful and effective encoding of HPSG into another formalism with computable mem-

bership problem, such as, in particular, linear context-free rewriting systems and abstract categorial grammars. At the same time the authors also show that an HPSG theory becomes decidable if the language generated by this grammar (feature structures encoded as formulas of MS_2) is also generated by a hyperedge replacement grammar.

2.4.1 Hyperedge Replacement Grammars

Hyperedge replacement grammars of [1], HRG, are grammars which generate languages of graphs. They appeared as counterparts of context-free grammars over the domain of graphs. Languages generated by hyperedge replacement grammars constitute so called equational sets of graphs, which are values of a regular set of graph expressions.

HRGs are capable of generating string languages as well, which do not have to be context-free, and at the same time they have context-free derivations. Roughly, it means that the derivations of every pair of nonterminals in a graph do not depend on one another (the property LCFRS's also have).

An important result that [30] establishes is that the trees generated by second-order abstract categorial grammars are those generated by HRGs. Then, we only need to give a mapping from HPSG to HRG, which will be transited to ACG automatically. However even in this case the complexity results suggest that it may not be possible in the general settings.

2.4.2 Abstract Categorial Grammars

Categorial grammars in general can better cope with word order phenomena, than for instance HPSG (Lambda grammars [46], Logical Grammar [50]). The ability of categorial grammars to separate superficial string operation from abstract syntax which guides the derivation allows to handle word order somewhat separately.

ACG is a grammatical framework dedicated to syntax and semantics of natural languages. It generates λ-terms which can be used to describe strings as well as trees. The appealing aspects of ACG (as other combinatorial grammars) are that ACG separates syntactic component from semantics and surface strings, adding to modularity of language processing, and that the abstract syntax is performed in the implicative fragment of linear logic [21], which is rather easy to understand and work with. Linear logic is prominent for its resource-sensitivity, meaning that once a formula is used, it can no longer be used, a property similar to linearity on rules of LCFRS's.

The two essential components of an ACG are its abstract language and

object language. The abstract language can be perceived as a set of abstract grammatical structures from which concrete structures of the object language are generated. A lexicon is a morphism from the abstract language to the object language which accounts for this generation.

The ACG framework is rather flexible and was originally designed to be the platform in which other grammatical formalisms can be encoded. And indeed it turned out that a number of formalisms can be represented as ACGs, including context-free grammars [16], tree-adjoining grammars [13], and to certain extent hyperedge replacement grammars [30].

As for the complexity results, string generative power of second-order abstract categorial grammars is equivalent to that of the context-free rewriting systems [15, 62, 58]. Then, it is no wonder that they have polynomial time recognition complexity. It means that second-order ACGs may generate some mildly context-sensitive languages. At the same time linear context-free rewriting systems have context-free derivations, as hyperedge-replacement grammars do. However, for ACGs of higher orders a clear picture of what complexity they have and what kind of languages they may generate has not yet been formed.

Another piece of motivation for the transformation we are about to make in this paper is that ACGs have been shown capable of representing underspecified semantic representations (URL semantics for TAG, [49]). Any semantic theory designed specifically for deep linguistic processing with HPSG, should account for underspecification.

Certain extensions to the base ACG formalism were proposed [14]. These extensions can represent feature structures with a few extra primitives. However, we are rather interested in the expressive power of the unextended formalism. Possibly, by making clear to which extent ACG may already imitate HPSG we can help better motivate possible extensions.

3 Approach

3.1 Conversion of HPSG into ACG

In the first part of the work we plan to concentrate on the theoretical possibility of embedding HPSG into ACG. However, we think that as the first step one should try to do it via the chain $HPSG \rightarrow HRG \rightarrow ACG$. We have already mentioned that the second step in the chain has been provided. Moreover, the complexity results suggest that if we try to build such a conversion in a more general fashion, then the second-order ACG would not be sufficient. But higher-order ACGs are not yet well studied. At the same time [32] also shows that an HPSG theory becomes decidable if the language generated by this grammar (feature structures encoded as formulas of MS_2) is also generated by a hyperedge replacement grammar.

It seems also clear that there may not in principle exist a fair conversion of $HPSG$ into HRG, basically because head-driven phrase structure grammars do not have context-free derivations. Thus, in this work, we try to establish conditions, possibly not too restrictive, under which a grammar in HPSG formalism may be modeled as an HRG.

As we have seen ACG and LCFRS generate the same string languages, though their tree-generative power was not yet compared. If we know that on theoretical level it is possible to transform an HPSG grammar in a restricted form into an ACG grammar, then we can also do a practical investigation of whether we can profit from such a transformation.

3.1.1 HPSG grammars and their models

Although HPSG applications have some kind of phrase-structure backbone, which can in principle be processed by a context-free component, in theoretical foundations of the formalism there need not be this component. HPSG is a constraint-based theory. It means that even if its data structures (feature structures) are constructed, or generated, by certain principles, a grammar itself only provides restrictions on the generated structures and it does not matter how this structures were brought to life.

HPSG has strong commitment to strict formalization of linguistic theories. There have been a long discussion about which logic is better for HPSG. The most influential ones are (Carpenter, [5]) and (King, [34]). We decided to use King's speciate reentrant logic as our base to represent head-driven phrase structure grammars. This logic directly explicates the founding HPSG principles and was appreciated by Pollard. However, it now seems

to be the case that this paper would have been shorter and more straightforward if we initially used the logic of (Carpenter [5]).

Anyway, no matter which logics one prefers, in practice NLP environments do not accept all possible functionalities that the formalism offers. In [23], the authors provide a direct conversion of HPSG type constraints into definite clause programs, which gives a partial case of closing the gap between description language theory and practical applications. We, in fact, do a similar work, although the target framework is different, HRGs rather than definitive clause programs.

Definition 3.1. A **signature** is a triple (S, F, A) where S is a finite set, those elements are called types (or species), F is a finite set of elements, called features, and A is total function form $F \times S$ to $\wp(S)$ called **appropriateness function**.

A feature path (or just path) is a dot-separated sequence of features $f_1.f_2 \ldots f_n$. There is an empty sequence, denoted by : and of zero length. Denote all finite feature paths as T_Σ. We sometimes use concatenation of paths, denoted as $path_1.path_2$. An atomic description is either $path_1 \approx path_2$, or $path_1 \sim \sigma$, where $\sigma \in S$ and $path_1, path_2 \in T_\Sigma$. Descriptions are defined as follows:

- atomic descriptions are descriptions,
- if ϕ and ψ are descriptions, then $\phi \vee \psi$, $\phi \wedge \psi$, $\phi \rightarrow \psi$ and $\neg\psi$ are descriptions,
- nothing else is a description.

A **theory** Θ is a set of descriptions. An HPSG **grammar** consists of a signature Σ and a theory Θ.

The signature gives another way to encode usual feature declaration of the form:

$$\sigma : \begin{bmatrix} F_1 \ \tau_1 \\ \vdots \\ F_n \ \tau_n \end{bmatrix}$$

where σ and τ_i are types and F_j are features. Such a declaration means that for the type σ, the features F_j are appropriate and the values $F_i(\sigma))$ must have types τ_j. Types in HPSG form a hierarchy and subtypes inherit feature declaration. However, nonmaximal types are superfluous and can be replaced by disjunctions of maximal types, especially if one expects the

semantics of a type be a set of tokens. Therefore, the set of types S in the signature stands for all maximal types. And the appropriateness function A, given F_i and σ, outputs all maximal subtypes of τ_i.

Definition 3.2. An **interpretation** of signature Σ is a triple $(\mathfrak{U}, \mathfrak{S}, \mathfrak{F})$ such that $\mathfrak{U}$ is a set, called the universe, $\mathfrak{S} : \mathfrak{U} \to S$ is a function distributing elements of $\mathfrak{U}$, called **tokens**, into types from S, and $\mathfrak{F}$ is a function from F to partial functions form $\mathfrak{U}$ to $\mathfrak{U}$. Moreover, for every $f \in F$ and $u \in \mathfrak{U}$, $\mathfrak{F}(f)(u)$ is defined iff $A(f, \mathfrak{S}(u))$ is defined. If both are defined, then $\mathfrak{S}(\mathfrak{F}(f)(u)) \in A(f, \mathfrak{S}(u))$.

An interpretation is the semantics of HPSG. The set $\mathfrak{U}$ is a set of linguistic tokens, e.g. there may be instances of words, like 'donkey' of type $word \in S$ with the case 'nominative' of type $nominative \in S$, or instances of sentences like 'Fido is a donkey' of type $sentence \in S$ with many possible appropriate features, such as $subject \in F$.

Given an interpretation I, function T_I is defined as a total function from T_Σ to partial functions from $\mathfrak{U}$ to $\mathfrak{U}$. It is recursively defined as follows:

- for every $u \in \mathfrak{U}$ $T_I(:)(u)$ equals u
- for $u \in \mathfrak{U}$ and $f = g.f_k \in T_\Sigma$, $g \in T_\Sigma$, $T_I(f)(u)$ is defined iff both $T_I(g)(u)$ and $\mathfrak{F}(f_k)(T_I(g)(u))$ are defined, and then $T_I(f)(u) = \mathfrak{F}(f_k)(T_I(g)(u))$.

In order to talk about models of a theory we need the notion of truth.

- Description $p_1 \approx p_2$ is true of token $u \in \mathfrak{U}$ iff $T_I(p_1)(u)$ and $T_I(p_2)(u)$ are defined and $T_I(p_1)(u) = T_I(p_2)(u)$.
- Description $p_1 \sim s$ is true of token $u \in \mathfrak{U}$ iff $T_I(p_1)(u)$ is defined and $\mathfrak{S}(T_I(p_1)(u)) = s$.
- $\neg\phi$ is true of u iff it is not the case that ϕ if true of u;
- $\phi \vee \psi$ is true of u iff ϕ and ψ are true of u;
- $\phi \wedge \psi$ is true of u iff ϕ or ψ are true of u;
- $\phi \to \psi$ is true of u iff ψ is true of u or it is not the case that ϕ is true of u;

Definition 3.3. An interpretation I **models** Θ if every description $d \in \Theta$ is true of every token $u \in \mathfrak{U}$. In this case we call I a **model** of Θ.

Only models of a grammar are relevant for it. It is imperative that the language being described is a model of the grammar. However, for a grammar, it is desirable that its models should be part of the language.

Definition 3.4. A **type definition** for $s \in S$ is a set of descriptions which are either of the kind

$$:\sim s \to path_1 \approx path_2$$

or disjunctions of descriptions of the kind

$$:\sim s \to ((path_1 \sim \sigma_1) \vee (path_1 \sim \sigma_2) \vee \cdots \vee (path_1 \sim \sigma_k)),$$

for one and the same $path_1$ in every disjunct, and $\sigma_i \in S$.

In simple words, the descriptions in Θ define each type in terms of sharing ($path_1 \approx path_2$) and subtyping ($path_1 \sim$ one of $\sigma_1 \dots \sigma_k$).

We consider HPSG theories Θ of a certain class. Namely, we demand that Θ must be a union of type definitions and none of the types has more than one definition. Denote by Θ_s all descriptions on the right-hand side of implications in the type definition of s. For our purposes it is also possible to accept a more general form of type definitions, which however may only operate within one type ($:\sim s \to X$, where X can have more elaborated structure with conjunctions, disjunctions or negation of descriptions).

This definition already underlines two important notions of HPSG: sharing and subtyping. Thus, for simplicity we decide to adhere to it. Type definitions of this or similar form are widely used in practice, for instance in such systems as LKB (developed in Stanford) or PET (Saarbrucken). It is also critical that the basic principles of HPSG of [51] may be encoded within a type, e.g. the head-feature principle of HPSG, saying that the head feature of a headed phrase must be token-identical to the head feature of the head daughter, can easily be written as

$$:\sim \textit{headed-phrase} \to$$
$$\textit{synsem.local.category.head} \approx \textit{daughters.head-dtr.synsem.local.category.head}$$

Fact. In this setting an interpretation is a model of Θ iff every description in Θ_s is true for every token $u \in \mathfrak{U}$ such that $\mathfrak{S}(u) = s$.

If $P(s)$ stands for all feature paths which occur in Θ_s, then $Pref(s) = \{f_1.f_2 \dots f_k \mid \exists f_1.f_2 \dots f_k.f_{k+1} \dots f_n \in P(s)\}$ is a set of all prefixes of $P(s)$ including the empty path. The set of feature paths that we mainly need is $Path(s) = Pref(s) \cup \{p.f \mid p \in Pref(s) \setminus :, f \in F\}$.

Example. We illustrate some of ideas of the paper in a toy grammar, actually even a fragment of a grammar. The grammar does not make much sense linguistically, but should demonstrate some basic principles oh how things work. It has a number of primitive types and the following type definitions:

$$\begin{array}{ccc} \textit{sg-noun} & \textit{intr-verb} & \textit{subj-rule} \\ \left[\begin{array}{l} \textit{cat} : \textit{noun} \\ \textit{number} : \textit{sg} \\ \textit{head} : \textit{n-head} \\ \textit{subcat} : \emptyset \end{array}\right] & \left[\begin{array}{l} \textit{cat} : \textit{verb} \\ \textit{number} : [1] = \textit{num} \\ \textit{head} : \textit{v-head} \\ \textit{subcat} : \left[\begin{array}{l} \textit{cat} : \textit{noun} \\ \textit{number} : [1] \end{array}\right] \end{array}\right] & \left[\begin{array}{l} \textit{cat} : \textit{sent} \\ \textit{head} : [1] \\ \textit{dtr-1} : \left[\begin{array}{l} \textit{head} : [1] \\ \textit{subcat} : [2] \end{array}\right] \\ \textit{dtr-2} : [2] = \textit{sign} \end{array}\right] \end{array}$$

We assume that types *noun*, *verb*, *sent*, *sg*, *pl*, *n-head*, *v-head*, $\emptyset$ are maximal and have no appropriate features. Let also *sign* be the supertype of *sg-noun*, *intr-verb* and *subj-rule*, the type *num* be the supertype of *sg* and *pl*, and *head* — the supertype of *n-head* and *v-head*. Then, the values of *subcat* in *intr-verb*, and *dtr-1* in *subj-rule* are of type *sign*. This grammar is supposed to parse one sentence, 'John sleeps', though for conciseness we did not add features for string values into the types.

From this data one can construct a signature $\Sigma = (S, F, A)$, where S consist of the maximal types we already mentioned. The set of features is all left-hand sides of the feature structures. Although we did not present type declarations, we may, for instance, assume that A is given by the following table

types	$features$	*result*
intr-verb, *sg-noun*, *subj-rule*	*cat*	$\{noun, verb, sent\}$
intr-verb, *sg-noun*	*number*	$\{sg, pl\}$
intr-verb, *sg-noun*, *subj-rule*	*head*	$\{n-head, v-head\}$
sg-noun	*subcat*	$\{\emptyset\}$
intr-verb	*subcat*	$\{sg-noun, intr-verb, subj-rule\}$
subj-rule	*dtr-1* , *dtr-2*	$\{sg-noun, intr-verb, subj-rule\}$

and the rest results in $\emptyset$.

The theory Θ has then three type definitions. For example, the type definition for *intr-verb* contains

$$\begin{gathered} :\sim \textit{intr-verb} \to \textit{cat} \sim \textit{verb} \\ :\sim \textit{intr-verb} \to (\textit{number} \sim \textit{sg}) \vee (\textit{number} \sim \textit{pl}) \\ :\sim \textit{intr-verb} \to \textit{head} \sim \textit{v-head} \\ :\sim \textit{intr-verb} \to \textit{subcat.cat} \sim \textit{noun} \\ :\sim \textit{intr-verb} \to \textit{number} \approx \textit{subcat.number} \end{gathered}$$

3.1.2 Graphs as descriptions of tokens

Let us define graphs. In our definition of a graph, labels are essential and inseparable components of edges.

Definition 3.5. A directed labeled rooted **graph** G, in the text just graph, over signature (S, F, A) is a triple (V, E, r), where $V = V(G)$ is a set of vertices, $r = r(G) \in V$ and $E = E(G) \subset V \times V \times F$ is a set of edges.

Everywhere in the text graphs are considered up to isomorphism. The set of all graphs is $\mathfrak{Gr}$. For an edge $e = (v_1, v_2, f)$ we use component-wise notation as in $e(1) = v_1$, $e(2) = v_2$, $e(3) = f$. Note also, that an edge e is uniquely defined by $e(1)$, $e(2)$, $e(3)$.

Definition 3.6. A graph $G \in \mathfrak{Gr}$ is **root-connected** if for every vertex v there is a sequence of edges $e_1, e_2, \dots e_k$ such that $e_{n+1}(1) = e_n(2)$ for $0 \leq n < k$, $e_0(1) = r$ and $e_k(2) = v$.

We say that a graph has **functional features** if there are no two edges e_1 and e_2, such that $e_1 = (v_1, v_2, f)$ and $e_2 = (v_1, v_3, f)$ with common v_1 and f.

Definition 3.7. A graph G becomes **completely labeled** if one provides a vertex labeling function $l : V(G) \to F$.

Completely labeled root-connected graphs with functional features may be a possible candidate to represent feature structures. They must be rooted, because one graph describes exactly one linguistic entity. Root-connectedness provides that nothing unrelated can also be in the description. Edges correspond to features, and those in turn are functional.

Definition 3.8. A **right congruence** $\sim$ defined on T_Σ is an equivalence relation, such that $\forall p_1, p_2 \in T_\Sigma$, $f \in F$ if $p_1 \sim p_2$, then $p_1.f \sim p_2.f$.

Definition 3.9. A graph $G = (V, E, r)$ is a **defining graph** for $s \in S$ if it has the following properties:

- there is a right congruence $\sim$ defined on T_Σ, which contains $path_1 \sim path_2$ for every $path_1 \approx path_2$ in Θ_s and such, that $V(G) = \{[p] \mid p \in Path(s)\}$, where $[p]$ is an equivalence class of p modulo $\sim$;
- the class $[:]$ is the root of G;
- $e = (v_1, v_2, f)$, $v_1 \in V(G)$, $v_2 \in V(G)$ is an edge in $E(G)$ if and only if there are p_1 and $p_2 = p_1.f$ in $Path(s)$, such that $v_1 = [p_1]$ and $v_2 = [p_1.f]$.

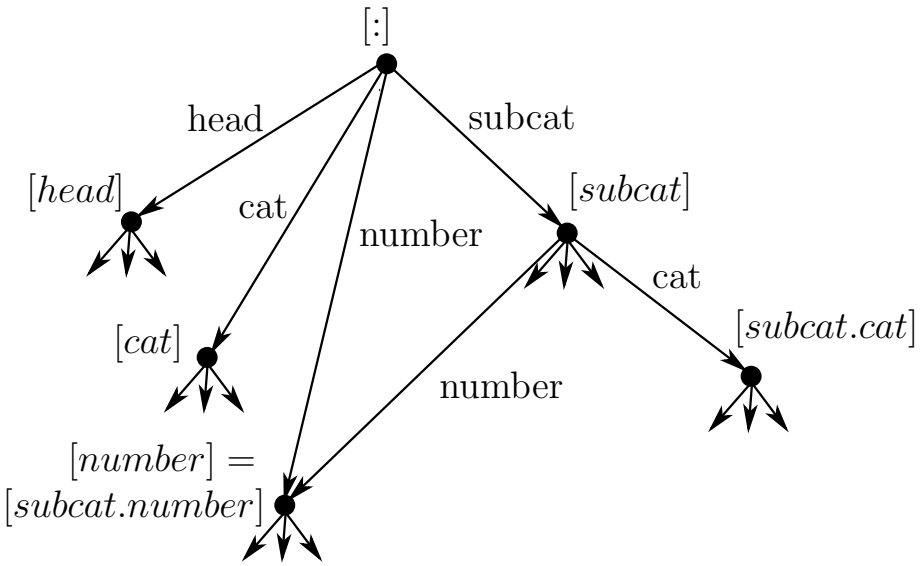

Figure 2: A defining graph generated by minimal congruence

The set of all defining graphs for s is denoted by Def_s.

Defining graphs are supposed to reflect all obligatory sharing coming from the type definition and additionally, since the congruence $\sim$ is otherwise arbitrary, they may contain sharing, possibly inferred from some other token above this one.

Fact. Every defining graph G is root-connected. If $v = [f_1.f_2 \dots f_k] \in V(G)$ for some $f_1.f_2 \dots f_k \in Path(s)$, then the sequence e_i, $1 \leq i < k$, with $e_i = ([f_1 \dots f_i], [f_1 \dots f_i.f_{i+1}], f_{i+1})$, and $e_0 = ([:], [f_1], f_1)$, provides the required connection of v and the root.

Fact. The graph G is completely defined by $\sim$. Moreover, if any two right congruences $\sim_1$ and $\sim_2$ coincide on $Path(s) \times Path(s)$, then they yield the same defining graph. Therefore, we can write $G = G(\sim)$ and $\sim = \sim_G$.

Example. The set Def_s is usually huge and most of the graphs there are irrelevant for our purposes. For our toy grammar, let us show one graph from $Def_{\textit{headed-phrase}}$. Among all congruences chose the minimal containing the pair (*number*, *subcat.number*). Then, the corresponding defining graph will look like in Figure 2. The forks represent edges for all features $f \in F$

For each $s \in S$ we want to chose a set of subgraphs of Def_s supplied with labeling functions in accordance with the appropriateness function. In particular, they will all have functional features. Definine the **center** of a defining graph G as $C(G) = \{w \mid w = [path], path \in Pref(s)\}$.

Definition 3.10. A graph D is an **extended defining graph** for $s \in S$ iff

1. there is $G \in Def_s$, such that $D \subseteq G$,
2. D is completely labeled by some labeling function l, such that for every $path \sim t$ from Θ_s holds $l([path]) = t$,
3. all appropriate edges of G are the edges of D, i.e. for every $v = [path] \in V(D)$ where $:\neq path \in Pref(s)$ and f, such that $A(f, l(v)) \neq \emptyset$, there is one edge $e = (v, u, f) \in E(D)$. Additionally, it is required that $l(u) \in A(f, l(v))$,
4. every edge $e = (v, u, f)$ where v and u are in the center of G belongs to $E(D)$.

Note, that because of Item 4 of Definition 3.10 we can understand the center of G as also the center of D.

We write Ext_s for the set of all extended defining graphs. Every graph D in Ext_s additionally to sharing information of Def_s has also subtyping restrictions of the type s. Moreover, 'central' vertices have out-edges for every appropriate feature and their ends have appropriate labels. Thus, Ext_s has all minimal graphs complying to the restrictions of the type definition of s and also to some of those of the appropriateness functions A. They are maximally specific in the sense that all vertices have labels residing in S, the set of (maximal) types. All in all, every token $u \in \mathfrak{U}$ which is an instantiation of $\mathfrak{S}(u) = s$ must at least partially 'look' like some $D \in Ext_s$. The root node is not expanded with all appropriate edges because they are allowed to be introduced by other 'higher' type definitions. It will become more clear when we formulate the condition under which an HPSG may be imitated by an HRG (Definition 3.20).

Example. Considering our running example, Figure 3 displays what an extended defining graph for the type *intr-verb* can be (the vertices' names are changed to labels). Note, that this graph (without labels) is the only possible extended graph one can obtain from the defining graph in Figure 2. However, the labels of the vertices, may be different, e.g. in the place of *sg-noun* could stay *intr-verb*, changing $\emptyset$ into something else, or instead of *sg* there could be *pl*.

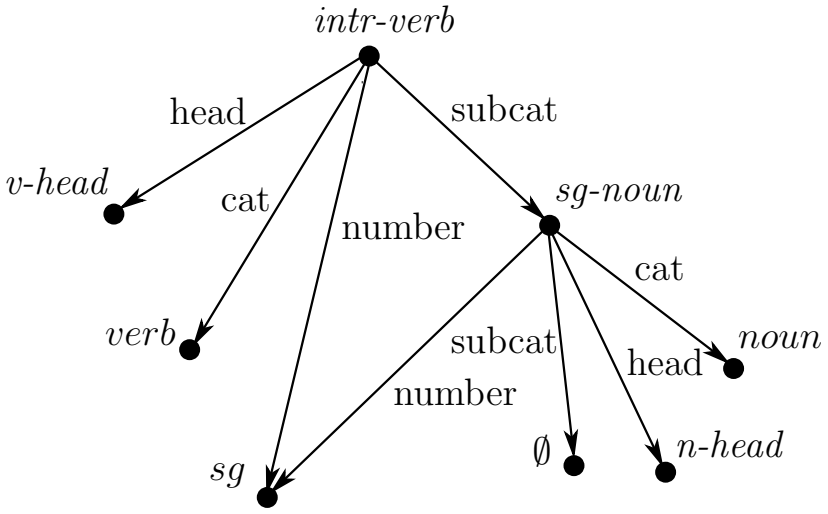

Figure 3: An extended defining graph

As we will use Def_f as building bricks of the hyperedge replacement grammar constructed from this HPSG, these sets better be finite.

Lemma 3.11. *Ext_s has only a finite number of graphs.*

Proof. Every graph D in Ext_s has no more vertices than $\#(Path(s))$. Since graphs are considered up to isomorphism, every edge is defined by a pair of vertices and a feature f taken from a finite set F, there may only be a finite number of possible graphs with a limited number of vertices. □

The final result of the conversion we are working on is a graph grammar, but in the logic used an HPSG generates no graphs, rather it limits the variety of possible interpretations. Then, one should be able to formally say, what it means that a graph corresponds to a token.

Definition 3.12. Let D be a graph, completely labeled by l, and $u \in \mathfrak{U}$ a token of some interpretation $I = (\mathfrak{U}, \mathfrak{S}, \mathfrak{F})$. We say that D **describes** u, iff there is a map $\phi : V(D) \to U$ such that:

- $\phi(r(D)) = u$
- $\mathfrak{S}(\phi(v)) = l(v)$,

- for every edge $e = (v_1, v_2, f) \in E(D)$ holds $\mathfrak{F}(f)(\phi(v_1)) = \phi(v_2)$

A map ϕ satisfying these three properties is called a **morphism** of D into $\mathfrak{U}$. In other words, D describes u if there is a morphism from D into $\mathfrak{U}$ which sends $r(D)$ to u. We say that D **completely describes** u if D describes u with such a morphism ϕ that for every $w \in \mathfrak{U}$, if $w = \phi(v_1)$ and for some $f \in F$ the value of $\mathfrak{F}(f)(w)$ is defined and equals q, then there is $e = (v_1, v_2, f) \in E(D)$. Then, of course holds $q = \phi(v_2)$.

The correspondence between graphs and tokens mentioned before is that a graph completely describes a token. The class of graphs which completely describe a token is much narrower than that of graphs just describing this token. For instance, a graph with one vertex v, such that $l(v) = \mathfrak{S}(u)$ for some $u \in \mathfrak{U}$ describes u, but it completely describe u only if there are no features $\mathfrak{F}(f)$ applicable to u.

A peculiar aspect here is that this morphism does not have to be injective. It would of course be very appealing to have full-blow isomorphisms. But in that case we would have to be able to reconstruct tokens with unpredictable sharing, i.e. whose with sharing which is not declared by any type definition. Such tokens are possible, as they do not contradict any description in Θ. We admit that we cannot model them, though, their existence is actually one of the things in King's theory of token which is subject to critics, for instance in [56].

Every isomorphism must preserve sequences of consequent edges as it is first established in Lemma 3.13 and later in Lemma 3.17.

Lemma 3.13. *Assume $D \in Ext_s$ describes some u with the morphism ϕ and v is in the center of D. Then, $\phi(v) = T_I(path)(u)$.*

Proof. We give an inductive proof by the length of $path$. If $path =:$ has length zero, then $[path] = r(D) = r$ and $\phi(v) = \phi([:]) = \phi(r) = u = T_I(:)(u)$

Assume now that $path = path_0.f$. Then, $path_0$ is also in $Pref(s)$. In this case, exploiting the inductive supposition we conclude, that $\phi(v_0) = T_I(path_0)(u)$ for $v_0 = [path_0]$. The graph D is an extended defining graph for s, hence by Definition 3.10, Item 4, there is an edge $e = (v_0, v, f)$ in $E(D)$, $v \in V(D)$. As a morphism, ϕ complies to $\mathfrak{F}(f)(\phi(v_0)) = \phi(v)$. Finally, we obtain

$$\phi(v) = \mathfrak{F}(f)(\phi(v_0)) = \mathfrak{F}(f)(T_I(path_0)(u)) = T_I(path)(u).$$

□

Having in mind an interpretation $I = (\mathfrak{U}, \mathfrak{S}, \mathfrak{F})$ and a token $u \in \mathfrak{U}$, satisfying all descriptions of Θ_s, we can construct G_u — a **defining graph initiated by token** u. It is defined by a congruence $\sim_u$. For every $p_1, p_2 \in T_\Sigma$, $p_1 \sim_u p_2$ iff either

- $T_I(p_1)(u)$ and $T_I(p_2)(u)$ are defined and $T_I(p_1)(u) = T_I(p_2)(u)$,
- or both are undefined.

It is obvious, that $\sim_u$ contains $(path_1, path_2)$ for $path_1 \approx path_2$ in $\Theta_{\mathfrak{S}(u)}$.

Lemma 3.14. *$\sim_u$ is a right congruence.*

Proof. As $\sim_u$ is defined with help of some other equality relation, the proof that it is an equality relation is obvious. Not that obvious is to see why it is a congruence. Consider an $f \in F$ and $p_1, p_2 \in T_\Sigma$ such that $p_1 \sim_u p_2$. If though $T_I(p_1)$(u) is undefined, then such is $T_I(p_1.f)(u)$ and the same holds for p_2. If $T_I(p_1)(u)$ and $T_I(p_2)(u)$ are defined and $T_I(p_1)(u) = T_I(p_2)(u) = T$, then there are again two cases: $\mathfrak{F}(f)(T)$ may exist or not. The former allows us to induce $T_I(p_1.f)(u) = \mathfrak{F}(f)(T) = T_I(p_1.f)(u)$ and the latter means that both $T_I(p_1.f)(u)$ and $T_I(p_2.f)(u)$ do not exist. □

The congruence $\sim_u$, as one can see, consist of all equations which are true in the interpretation where u is located. Then, $G_u \in Def_s$ is a natural choice of a defining graph to be extended to $D_u \in Ext_s$ which describes u (in fact, it is not extended, rather shrinked, though a vertex labeling function is added).

The following theorem basically confirms the intuitions we put into Ext_s.

Theorem 3.15. *Let $I = (\mathfrak{U}, \mathfrak{S}, \mathfrak{F})$ be an interpretation of signature Σ and $u \in \mathfrak{U}$, $\mathfrak{S}(u) = s$. Then, every description of Θ_s is true of u if and only if there is a $D \in Ext_s$ which injectively (with an injective morphism) describes u.*

Proof. First, let us assume D describes u. Consider a description $d \in \Theta_s$. The description d can be either $path_1 \approx path_2$ or $path_1 \sim t$ for some $t \in S$.

If the former if true, then $path_1, path_2 \in Pref(s)$ and $path_1 \sim_G path_2$ for the defining graph $G \supseteq D$. Since D describes u, there is a morphism $\phi : V(D) \to U$. Define a token $v = \phi([path_1]) = \phi([path_2])$. By Lemma 3.13, $v \in V(D)$, $\phi(v) = T_I(path_1)(u)$ and $\phi(v) = T_I(path_2)(u)$, implying $T_I(path_1)(u) = T_I(path_2)(u)$, which means that d is true of u.

If d is of the kind $path_1 \sim t$, then we may again consider the vertex $v = [path_1]$ in $V(D)$. The fact that $D \in Ext_s$ and Definition 3.10, Item 2,

imply that D is completely labeled with some l, in such a way that for every $path_0 \sim t_0$ in Θ_s holds $l([path_0]) = t_0$. In particular, $l([path_1]) = t$. While ϕ is a morphism, $\mathfrak{S}(\phi(v)) = l(v)$ for all $v \in V(D)$, leading to $\mathfrak{S}(\phi([path_1])) = t$. Using Lemma 3.13 we observe that, $\phi([path_1]) = T_I(path_1)(u)$. Therefore, holds $\mathfrak{S}(T_I(path_1)(u)) = t$ and it in turn means that $path_1 \sim t$ is true of u.

Now, we shall prove the backward implication. Assume that every description in Θ_s is true of some token $u \in \mathfrak{U}$, $\mathfrak{S}(u) = s$. Let us define D as a subgraph of G_u with the labels given by $l([path]) = \mathfrak{S}(T_I(path)(u))$ and only those vertices of G_u are left in D for which $T_I(path)(u)$ is defined. The edges of D are those of G_u whose nodes are in $V(D)$.

In order l to be uniquely defined, it is required that $l([path_1]) = l([path_2])$ if $[path_1] = [path_2]$ for some $path_1$, $path_2$ in $Path(s)$. Thus, we need to establish that $\mathfrak{S}(T_I(path_1)(u)) = \mathfrak{S}(T_I(path_2)(u))$ holds if $path_1 \sim_u path_2$. From definition of $\sim_u$ we conclude, that $T_I(path_1)(u)) = T_I(path_2)(u))$ and it is even more than required.

Now, we observe that $D \in D_s$. Indeed,

1. $D \subseteq G_u$ by construction;

2. for the labeling function l that we defined and every description $path \sim t$ in Θ_s, as I is a model, follows $l([path]) = \mathfrak{S}(T_I(path)(u)) = t$;

3. Take any $v \in V(D)$, $v = [path]$, $path \in Pref(s)$ and an $f \in F$ with $A(f, l(v)) \neq \emptyset$. There is an edge $e = (v, v_2, f) \in E(G) \supseteq E(D)$, then $path = f_1.f_2 \dots f_k$, $v_2 = [f_1.f_2 \dots f_k.f]$ and

   $$l(v_2) = \mathfrak{S}(T_I(f_1.f_2 \dots f_k.f)(u)) = \mathfrak{S}(\mathfrak{F}(f)(T_I(f_1.f_2 \dots f_k))(u)).$$

   Moreover,

   $$l(v) = l([f_1.f_2 \dots f_k]) = \mathfrak{S}(T_I(f_1.f_2 \dots f_k)(u)).$$

   By Definition 3.2,

   $$\mathfrak{S}(\mathfrak{F}(f)(T_I(f_1.f_2 \dots f_k))(u)) \in A(f, \mathfrak{S}(T_I(f_1.f_2 \dots f_k)(u))),$$

   and thus $l(v_2)$ is defined, $e \in E(D)$, and after rewriting we get the desired equality $l(v_2) \in A(f, l(v))$.

4. If $u = [path_1]$ and $v = [path_2]$, then while $path_i$ are in $Pref(s)$ and u satisfies all desriptions both $l(u)$ and $l(v)$ are defined.

Further, we need to establish that D describes u. For $v \in V(G)$, $v = [path]$, put $\phi(v) = T_I(path)(u)$. If $v = [path_1] = [path_2]$, then $path_1 \sim_u path_2$, $T_I(path_1)(u) = T_I(path_1)(u)$ and ϕ is correctly defined. Let us deal with the required properties for ϕ to be a morphism one by one.

- $\phi(r) = T_I(:)(u) = u$.
- If $v = [path] \in V(D)$, $\mathfrak{S}(\phi(v)) = \mathfrak{S}(T_I(path)(u)) = l(v)$.
- For each edge $e = (v_1, v_2, f) \in E(G)$ holds

$$\mathfrak{F}(f)(\phi(v_1)) = \mathfrak{F}(f)(T_I(path)(u)),$$

  where $v_1 = [path]$. Further, $\mathfrak{F}(f)(T_I(path)(u)) = T_I(path.f)(u) = \phi(v_2)$, because $v_2 = [path.f]$.

The injectivity follows immediately from the definition of G_u □

3.1.3 Routes in graphs and joint union

Definition 3.16. A **route** in a graph $D \in \mathfrak{Gr}$ starting at $v \in V(D)$ is a feature path $p = f_1.f_2 \dots f_k \in T_\Sigma$ such that there exist a sequence of edges e_1, e_2, …, e_k, $e_i \in E(D)$, such that the labels $e_i(3) = f_i$, $e_1(1) = v$ and $e_{i+1}(1) = e_i(2)$. Moreover the empty feature path : is also a route.

Denote all routes of a graph starting at v by $Route_v(D)$. Then, $Route(D)$ is a shortening for $Route_{r(D)}(D)$. We also denote the end point of a route $p \in Route_v(D)$ of length k by $End_v(p) = e_k(2)$ if $k \geq 1$, and $End_v(:) = v$. By analogy, $End(p) = End_{r(D)}(p)$. The value $End_v(p)$ is not always uniquely determined, because there may be various sequences of edges with given properties. If, however D has functional features, then ends are unique. Otherwise we still use this notation and by $End_v(p)$ mean any end or a particular one.

The following lemma is a counterpart of Lemma 3.13.

Lemma 3.17. *If D describes u via ϕ, $v \in V(D)$, and $\phi(v) = w$, then $\phi(End_v(route)) = T_I(route)(w)$ for $route \in Route_v(D)$.*

Proof. We give a proof by induction by the length of *route*. If $route =:$, then $\phi(End_v(:)) = \phi(v) = w = T_I(:)(w)$. Now assume $route = route_0.f$ is licenses by the sequence $(e_1, e_2, \dots, e_k, e_{k+1})$, where $(e_1, e_2, \dots, e_k)$ is the sequence for $route_0$, $e_{k+1}(3) = f$, $e_{k+1}(1) = End_v(route_0)$ and $e_{k+1}(2) = End_v(route)$. Then, by inductive assumption

$$\phi(e_{k+1}(1)) = \phi(End_v(route_0)) = T_I(route_0)(w).$$

Finally, due to existence of the edge e_{k+1} and the fact that D describes u,

$$\phi(End_v(route)) = \mathfrak{F}(f)(\phi(e_{k+1}(1))) = \mathfrak{F}(f)(T_I(route_0)(w)) = T_I(route)(w).$$

□

Definition 3.18. Completely labeled graphs D_1 with $v \in V(D_1)$ and D_2 are **compatible** in v if for every $route_1, route_2 \in Route_v(D_1) \cap Route(D_2)$, the following applies:

$$End_v(route_1) = End_v(route_2) \iff End(route_1) = End(route_2),$$

$$l(End_v(route_1)) = l(End(route_1)),$$

Compatibility of two graphs in v means that we can align them in such a way that v is aligned with the root of the second graph and edges go to edges with identical labels. Moreover, by doing so we do not have to change the graphs, or in other words, the intersecting parts are identical in both graphs. In the construction of the final HRG we will only try to combine compatible extended defining graphs. Additionally, it is intuitively clear that even if $D_1 \in Ext_{s_1}$ is not compatible with $D_2 \in Ext_{s_2}$, then it may be possible to find another graph $D_3 \in Ext_{s_2}$, which can be more (or less) restrictive than D_2, and compatible with D_1. Otherwise, it should be the case that the type definition of s_2 prohibits D_1 to describe any token.

Lemma 3.19. *If D injectively describes u with ϕ, $v \in V(D)$, $\phi(v) = w$, and D_2 injectively describes w, then D and D_2 are compatible in v.*

Proof. By Lemma 3.17, $End_v(route_1) = End_v(route_2)$ iff $T_I(route_1)(w) = T_I(route_2)(w)$. At the same time $End(route_1) = End(route_2)$ in D_2 iff $T_I(route_1)(w) = T_I(route_2)(w)$. Similarly,

$$l(End_v(route_1)) = \mathfrak{S}(T_I(route_1)(w)) = l(End(route_1)).$$

□

As a consequence of Lemma 3.19 we see that under certain conditions two graphs describing one token (a token and another one under it) may be combined.

We need to identify how one type definition interleaves with another. To this end we introduce **influence** of vertex $v \in V(D)\backslash r(D)$ on a graph $D \in Ext_{s_1}$ as $D{\cap_v} = Route_v(D) \cap Route(D_2)$, where $D_2 \in Ext_{l(v)}$ is any compatible with D in v graph. The definition depends on the choice of D_2, but it will be fixed in the next definition. The influence set under vertex v is $V(D{\cap_v}) = \{w | w = End_v(r) \in V(D), r \in D{\cap_v}\}$.

Definition 3.20. We say that an HPSG grammar has **non-interactive derivations** if for every $s \in S$, $D \in Ext_s$ there is a set of vertices $P(D) \subseteq C(D)$, such that

- if D_2 and D_3 are graphs in $Ext_{l(v)}$ both compatible with D in $v \in P(D)$, then $Route_v(D) \cap Route(D_2) = Route_v(D) \cap Route(D_3)$ (uniqueness of intersection),
- $V(D\cap_v)$ does not intersect with $V(D\cap_u)$ for any two different $v, u \in P(D)$,
- holds $\cup_{v\in P(D)} V(D\cap_v) = V(D)\backslash\{r\}$,
- and for every route $r = f_1.f_2 \dots f_k \in Route_v(D)$, $v \in P(D)$, either $r \in D\cap_v$ or $End_v(f_1) = u \in P(D)$, $u \neq v$.

Every restriction has its own motivation which comes from general observations, rather than technicalities.

The first restriction actually says that if a type definition allows a certain configuration under a token of this type, then another such configuration should be similar to it. In HPSG this is what is achieved by having a type hierarchy. Then, all type definitions should be consistent with the hierarchy. Even though, there are only maximal types in S, we still have subtyping restrictions in Θ as well as unspecified types of other implicitly participating tokens corresponding to prefixes of $P(s)$, or vertices of $C(D)$.

Clause two does most of the job. In its essence the restriction is that we can find a set of 'nonterminals' which do not interact. Any context-free grammar has the property, that for any rule, say, $S \to NP\ VP$ neither before these NP and VP appeared in the derivation, nor after they merged in S, there was no interaction between them. This also means that S does not depend on how its daughters were generated. In HPSG every sharing is a token-sharing. In particular this implies that the shared part may be arbitrary big, and at the same time it might be crucial for further derivation. It seems impossible to encode this kind of interactions between either NP and VP, or S and NP in the abstract language of a second-order ACG. The last but not least restriction is also about interactions. Namely, if there is a sharing relation between 'nonterminals', then it may only exist for their 'roots'. Otherwise, it would be just impossible to track down the relation between them. We admit that the argument is not exhaustive, though.

Definition 3.21. Joint union $S_1 \cup_R S_2$ of two disjoint ($S_1 \cap S_2 = \emptyset$) sets S_1 and S_2 modulo a bijection $R : A_1 \to A_2$ of two subsets $A_1 \subseteq S_1$, $A_2 \subseteq S_2$

is a set Z and two injective maps $\beta^1_Z : S_1 \to Z$, $\beta^2_Z : S_2 \to Z$ such that $\beta^1_Z(S_1) \cup \beta^2_Z(S_2) = Z$, and $\beta^1_Z(s_1) = \beta^2_Z(s_2)$ iff $s_2 = R(s_1)$.

Fact. Joint union always exists. Moreover, let Z_1 and Z_2 both be the joint union of S_1 and S_2 modulo R. Then, there is a bijection $B : Z_1 \to Z_2$ such that $B(\beta^1_{Z_1}(s_1)) = \beta^1_{Z_2}(s_1)$, $B(\beta^2_{Z_1}(s_2)) = \beta^2_{Z_2}(s_2)$.

Lemma 3.22. *Suppose there are functions $f_1 : S_1 \to S$ and $f_2 : S_2 \to S$, such that if $s_2 = R(s_1)$ then $f_1(s_1) = f_2(s_2)$. Then, there is $f : S_1 \cup_R S_2 \to S$ with $f(\alpha^i(s_i)) = f_i(s_i)$ for $i \in \{1, 2\}$.*

Proof. As $\alpha^1(S_1) \cup \alpha^2(S_2) = S_1 \cup_R S_2$, then we can simply define $f(\alpha^i(s_i)) = f_i(s_i)$ and assure its correctness. From $\alpha^i(s_i) = \alpha^i(s'_i)$ follows $s_i = s'_i$, and $\alpha^1(s_1) = \alpha^2(s_2)$ implies $s_2 = R(s_1)$, but then our assumption leads to $f_1(s_1) = f_2(s_2)$. □

Definition 3.23. If two graphs $D_1 \in \mathfrak{Gr}$ and $D_2 \in \mathfrak{Gr}$ are compatible in $v \in V(D_1)$, then the **joint union of graphs** $D = D_1 \cup_v D_2$ consist of:

- $V(D) = V(D_1) \cup_R V(D_2)$, where $R(v_1) = v_2$ iff $End_v(route) = v_1$ and $End(route) = v_2$ for some $route \in Route_v(D_1) \cup Route(D_2)$,
- set of vertices $E(D)$, such that $e \in E(D)$ iff there is $e' \in V(D_i)$ for some $i \in \{1, 2\}$, such that $e(1) = \beta^i(e'(1))$, $e(2) = \beta^i(e'(2))$ and $e(3) = e'(3)$,
- $r(D) = \beta^1(r(D_1))$.

Lemma 3.24. *Definition 3.23 is correct.*

Proof. Initially, we need to establish that R is correctly defined and a bijection. First, the relation R is functional. Indeed, if $R(v_1) = v_2$, $End_v(route_1) = v_1$, $End(route_1) = v_2$ and $R(v_1) = v_3$, $End_v(route_2) = v_1$ and $End(route_2) = v_3$, then from compatibility of D_1 and D_2 follows that $v_2 = v_3$. The same arguments provide injectivity of R and this means, that it is a bijection of two subsets of $V(D_1)$ and $V(D_2)$. □

Labeling of vertices is external to the definition of graphs, therefore the joint union should also be supplied with a labeling function. It would be natural to inherit it from those of the two components.

Lemma 3.25. *Assume D_1 and D_2 are compatible in v. Let l_1 and l_2 be vertex labeling functions of D_1 and D_2 respectively, then $D = D_1 \cup_R D_2$ is defined and there is $l : V(D) \to S$, such that $l(\alpha^i(v_i)) = l_i(v_i)$ for any $i \in \{1, 2\}$.*

Proof. By Lemma 3.24 the $D = D_1 \cup_R D_2$ is defined, and Lemma 3.22 provides such an l. □

Now, we have the notion of joint union of compatible graphs. At the same time Lemma 3.19 gives the necessary compatibility under the condition that the two graphs describe parts of one and the same token. So, we may combine extended defining graphs by means of joint union in order to gain more enhanced descriptions of the current token. In the end, the process should stop, yielding a graph which is a complete description.

3.1.4 Hyperedge replacement grammars

Hyperedge replacement grammars were first introduced in [1]. A hypergraph is a generalization of a graph which allows edges to be incident to multiple nodes. A hyperedge replacement consist in substituting a hyperedge in a graph by another hypergraph and a hyperedge replacement grammar is in simple words a set of pairs (hyperedge label, hypergraph) which are used to build a graph language from the initial edge. Hyperedge replacement grammars have a number of nice properties. First, substitution is not dependent on the context in which the edge is found. Further, substitution is sequentializable and parallelizable, meaning that substitution of a bunch of edges can be performed simultaneously or one after another and the result will be the same. Finally, the operation can be shown to be associative.

The idea of substitution nicely agrees with resource sensitivity of ACGs, because a edge is only created once, and in the end it is always possible to track down where exactly it comes from. Supported with the result of [30] the trees generative powers of the two formalisms are proven to be equivalent. ACGs can be imitated by HRGs with a quite straightforward embedding, where hyperedge replacement corresponds to λ-substitution $M[x := N]$, though, the reverse is a bit more involved.

Definition 3.26. Let Γ be an alphabet of edge labels, and Δ be an alphabet of selectors. A **hypergraph** H over Γ and Δ is a tuple (V, E, lab, nod, ext), where V is a finite set of vertices, E is a finite set of edges (hyperedges), $V \cap E = \emptyset$, $lab : E \to \Gamma$ is an edge labeling function, incidence function nod associates a partial function $nod(e) : \Delta \to V$ with an edge $e \in E$, and the external function is a partial function $ext : \Delta \to V$. The type of H is $type(H) = dom(ext)$ and the type of a node e is $type(e) = dom(nod(e))$.

The set of all graphs over Γ and Δ is denoted by $\mathfrak{G}_{\Gamma,\Delta}$ or just $\mathfrak{G}$. In our case, we put $\Delta = T_\Sigma$ and $\Gamma = S \cup F \cup Acc$, where Acc will be defined later.

Although $\Delta = T_\Sigma$ is infinite, a concrete grammar will only make use of a finite subset of Δ.

To distinguish between the components of different graphs we may use different notations for $V = V(H) = V_H$, $E = E(H) = E_H$, $lab = lab_H$, $nod = nod_H$, and $ext = ext_H$.

Let H and K be hypergraphs, $e \in E(H)$, $type(e) = type(K)$, then hypergraph $(H - e) + K = (V, E, lab, nod, ext)$, where $V = V(H) \cup V(K)$, $E = (E(H)\backslash\{e\}) \cup E(K)$, $lab = lab_H \cup lab_K$ restricted to E, $nod = nod_H \cup nod_K$ restricted to E and $ext = ext(H)$. The **substitution** of K in H for e is $H[e/K] = ((H - e) + K)/R = G$, where R is the minimal equivalence relation containing $\{nod(H)(e, s), ext(K)(s)) \mid s \in type(e)\}$.

One can define **projections** $\beta^1_V : V_H \to V_G$, $\beta^1_E : (E_H\backslash\{e\}) \to E_G$ and $\beta^2_V : V_K \to V_G$, $\beta^2_E : E_K \to E_G$. They satisfy $nod(\beta^i_E(e))(s) = \beta^i_V(nod(e)(s))$ and preserve lab. Moreover, β^i_E are injective. We may occasionally use β^i which denotes β^i_V, or β^i_E, or both, depending on the context.

So far we have defined hypergraphs and the operation of substitution. Projections provide a means to talk about which parts of the resulting graph come from which graph used. They are pretty similar to the projections of two sets into their joint union, though, it is not always the case that they are injective. In case the functions nod of the node being substituted or ext of the substituting hypergraph are not injective then the substitution may merge some vertices.

In Definition 3.27 we define a class of substitutions, called plain substitutions, which do not merge vertices. We plan to use only these, partly because they do not introduce information which is not contained in the two hypergraphs, and partly because it is easier to describe a sequential process of substitutions if previously created hypergraphs are passed further unchanged.

Definition 3.27. If hypergraphs H and K, and the edge $e \in E(H)$ satisfy the following condition:

$$nod_H(e)(s_2) = nod_H(e)(s_2) \Leftrightarrow ext_K(s_1) = ext_K(s_2),$$

for any s_2, s_2 in $type(e) = type(K)$, then we say that the substitution of K in H for e is **plain**.

Definition 3.28. If a substitution is plain, then we can use R as a bijection between $\{nod_H(e)(s) \mid s \in type(e)\}$ and $\{Ext_K(s) \mid s \in type(e)\}$. Precisely, it is defined as in $R(nod_H(e)(s)) = Ext_K(s)$. Further, $V(H[e/K]) = V(H) \cup_R V(K)$, $E(H[e/K]) = (E(H)\backslash\{e\}) \cup E(K)$. For $e_2 \in E(H)$, the

labeling function $lab(e_2)$ on $H[e/K]$ is defined as $lab_H(e_2)$ and on $E(K)$ — as $lab_K(e_2)$. The incidence function nod satisfies $nod(e_1) = \beta^1(nod_H(e_1))$ or $nod(e_2) = \beta^2(nod_K(e_2))$.

All in all, a plain substitution is one with injective β^i_V. The vertex set of the substitution is then the joint union of those of the participating graphs.

Definition 3.29. A **hyperedge replacement grammar** over Γ and Δ is a tuple $HRG = (N, T, P, Start)$, where N is a Δ-typed alphabet of nonterminal edge labels. A Δ-typed alphabet is an alphabet $N \subseteq \Gamma$ together with a mapping $type : N \to \wp(\Delta)$. $T \subseteq \Gamma$ is the alphabet of terminal edge labels, disjoint with N, P is a finite set of productions and $Start \in N$ is the initial nonterminal. A **production** $p \in P$ is a pair (n, G), $n \in N$, G — a hypergraph over $N \cup T$ and Δ. It must also satisfy the equation $type(n) = type(G)$ and for every edge $e \in E(G)$ holds $type(e) = type(lab_G(e))$.

Again, we restrict ourselves to using only certain types of edges. We use typed terminals, and namely, $T = S \cup F$, each terminal $t \in S$ has type of the empty path $\{:\}$ and it corresponds to a vertex label, whereas each terminal $f \in F$ has type $type(f) = \{:, f\}$.

Definition 3.30. A hypergraph G meets **feature structure requirements** if for every $v_1 \in V(G)$, there is exactly one $e \in E(G)$, $lab(E) \in S$, $nod(e)(:) = v_1$ and there are no two edges with labels from F and identical nod and lab values.

Frankly speaking, Definition 3.30 is set up in such a way that it directly corresponds to graphs we defined to describe tokens. The requirement demands that vertices have unique labels and that edges are uniquely defined by vertices incident to them and the labels.

Theorem 3.31. *There is one-to-one correspondence between hypergraphs $\mathfrak{G}$ without nonterminal edges, meeting feature structure requirements, modulo all possible port sequences, and completely labeled graphs of $\mathfrak{Gr}$ modulo any choice of the root.*

Proof. For a graph $H \in \mathfrak{G}$ let us define $\eta(H) = G \in \mathfrak{Gr}$. First, put $V(G) = V(H)$. Then, for every edge $e \in E(H)$, such that $lab(e) \in F$, create an edge $(nod(e)(:), nod(e)(lab(e)), lab(e))$ in $E(G)$. Further, if $lab(e) \in S$, then define $l(nod(e)(:)) = lab(e)$. Thus, we have a labeled graph G and a function $l : V(G) \to S$. It is completely labeled because for every vertex $v \in V(G) = V(H)$ there is an edge $e \in E(H)$ (due to feature structure requirements), such that $lab(s) \in S$, hence $l(v) = l(nod(e)(:)) = lab(e)$ is

defined. Moreover, feature structure requirements also provide that the label is uniquely defined, as the edge e is unique. Although it is not crucial, we also notice that every edge $i = (i_1, i_2, i_3) \in E(G)$ is created only once. Otherwise $i = (nod(e_1)(:), nod(e_1)(lab(e_1)), lab(e_1)) = (nod(e_2)(:), nod(e_2)(lab(e_2)), lab(e_2))$, and then $e_1 = e_2$. The root of G can be any vertex.

Now we provide the function $\mu : \mathfrak{G}\mathfrak{r} \to \mathfrak{G}$, such that $\eta\mu = (ID)_{\mathfrak{G}}$, $\mu\eta = (ID)_{\mathfrak{G}\mathfrak{r}}$, where (ID) are identity maps. For a graph $G = (V(G), E(G), r(G))$ and $l : V(G) \to S$ we build $H = (V(H), E(H), lab_H, nod_H, ext_H) \in \mathfrak{G}$. First, ext is any possible function form Δ to V_H. Second, $V(H) = V(G)$. Then, for every $v \in V(G)$, introduce $e \in E(H)$, $lab(e) = l(v) \in S$ and $nod(e)(:) = v$. Similarly, for every edge $e \in E(G)$, build an edge $i \in E(H)$, such that $nod(i)(:) = e(1)$, $nod(i)(e(3)) = e(2)$ and $lab(i) = e(3)$. One can see that all edges are terminal. Feature structure requirements are fulfilled for vertex labels by construction and for other edges because every $e \in E(G)$ is identified by its $e(1)$, $e(2)$ and $e(3)$.

It is not very difficult to see that μ and η are reverse functions. Consider $M = \eta\mu(H) \in \mathfrak{G}\mathfrak{r}$. Trivially, $V(M) = V(H)$, $l_H(v) = lab(e)$ and $nod(e)(:) = v$ for $v \in V(M)$, $e \in E(\mu(H))$. By definition of η, $l_M(v) = l_M(nod(e)(:)) = lab(e) = l_H(v)$. Finally, every $e = (e(1), e(2), e(3)) \in E(H)$ corresponds to $i \in E(\eta(H))$ with $nod(i)(:) = e(1)$, $nod(i)(e(3)) = e(2)$ and $lab(i) = e(3)$ and i in turn corresponds to $e_0 = (e(1), e(2), e(3)) = e$ in M. The proof of the other equality in very little way differs from that given above. □

The correspondence given by Theorem 3.31 does not restrict the external function and also it conceals the information of the root of a graph. Although hypergraphs have no notion of root we assume that all hypergraphs obtained by this correspondence also have a dedicated vertex which is the image of the root of the object graph. The property of having functional features (as well as root-connectedness) is not formulated for hypergraphs, but for some hypergraphs G it will be established as such of $\eta(G)$.

Although certain hypergraphs with only terminal edges can be identified with graphs, there may of course occur hypergraphs with nonterminals. For such a hypergraph H we may use its **skeleton** $sk(H)$ which the same hypergraph as H only without nonterminal edges.

Definition 3.32. Let G be a hypergraph, $e \in E(G)$ and $p = (lab(e), R) \in P$ be a production. Then, G directly derives $G' = G[e/R]$ and we explicate it by writing $G \Rightarrow_p G'$ or just $G \Rightarrow G'$. If there is a sequence of derivations $G_1 \Rightarrow G_2 \Rightarrow \cdots \Rightarrow G_k$, then we say that G_1 derives G_k and denote it by

$G_1 \Rightarrow^* G_k$. The language generated by a HRG is

$$La(HRG) = \{H \in \mathfrak{G} \mid H = sk(H), Start^\bullet \Rightarrow H\},$$

where $Start^\bullet$ is a hypergraph which has one edge, labeled $Start$. For our start symbol holds $type(Start) = \{:\}$, it has one vertex, and undefined ext.

We say that $H \in \mathfrak{G}$ describes (completely describes) a token $u \in U$, under vertex $v \in V(H)$ if $sk(H)$ satisfies feature structure requirements, $G = \eta(sk(H))$, supplied with $r(G) = v$, describes (completely describes) u.

3.1.5 Construction of HRG form HPSG

It is finally the time when we can construct the hyperedge replacement grammar from the set of primitives we have formed on basis of a head-driven phrase structure grammar. As atomic graphs we want to employ $D \in Ext_s$. However they are not sufficient in the form they exist now. During derivation a graph may be accessed from above by a certain set of routes. All vertices and edges along these routes should be removed from D since, otherwise, they will be repeated creating not functional features and contradicting to unique information source commitment.

Assume $D \in \mathfrak{G}\mathfrak{r}$, $\mu(D) \in \mathfrak{G}$ and $R \in Route(D)$. Then, after removing all edges along R in $\mu(D)$ one gets $\mu(D) - R$. More precisely,

- $V(\mu(D) - R) = V(\mu(D))$,
- $e \in E(\mu(D) - R)$, $lab(e) \in F$ iff $e \in E(\mu(D))$ and there are no $r_1 = f_1.f_2 \dots f_k$ and $r_2 = f_1.f_2 \dots f_k.f$ in R such that $nod(e)(:) = End(r_1)$, $nod(e)(lab(e)) = End(r_2)$,
- $e \in E(\mu(D) - R)$, $lab(e) \in S$ iff $e \in E(\mu(D))$ and there is no $r_1 = f_1.f_2 \dots f_k \in R$, such that $nod(e)(:) = End(r_1)$.

There is another minor change to be done. Because we must make sure that all vertices are expanded with all appropriate edges, introduce an exception for $\mu(D) - \emptyset$. It may only appear in the initial productions or for daughers of the root of such productions, as described later. $\mu(D) - \emptyset$ is defined as a set of $\mu(D_0)$, where D_0 is D united with $\{(r, v_f, f) \mid A(f)(l(r)) \neq \emptyset\}$ if those do not exist yet in D. The labels are restricted by $l(v_f) \in A(f)(l(r))$.

Now, we introduce possible expansions of graphs from Ext_s in the realm of hypergraphs by creating nonterminals in them. We already suggested that nonterminals will span over the sets $V(D\cap_v)$, $v \in P(D)$ and they be labeled by a graph and the set of routes they are accessed by from outside.

Definition 3.33. Assume we have an HPSG grammar which has non-interactive derivations. For every graph $D \in Ext_s$ (or $D = G_0$, $G \in Ext_s$) and a set of routes $R \subseteq Route(D)$ (may be $\emptyset$) we introduce the set $\mathfrak{G}^{D,R}$ of all graphs $H \in \mathfrak{G}$ such that

- $sk(H) = \mu(D) - R$ or $sk(H) = \mu(D_0)$ if $R = \emptyset$,
- nonterminal edges of H are obtained in the following way: By the assumption of the lemma, there is a set $P(D) \subseteq C(D)$ satisfying the properties of Definition 3.20. Each new edge e corresponds to a vertex $v \in P(D)$.
  - The edge e has label $lab(e) = D_2(D\cap_v)$, which consist of a graph $D_2 \in Ext_s$ extending and a set of routes $D\cap_v = Route_v(D) \cap Route(D_2)$.
  - The type of e is $Routes(D_v) \cap Routes_v(D)$ and the incidence function complies to $nod(e)(route) = End_v(route) \in V(D) = V(\mu(D))$. We say that e is **rooted** in $nod(e)(:) = End_v(:) = v$.
  - D is compatible with D_2.
- If $R = \emptyset$, then there are more nonterminals for every new vertex v_f of D_0 with labels $D_2(\emptyset)$, $D_2 \in Ext_s$, $s = l(v_f)$.
- The external function is set by $dom(ext) = R$ and $ext(route) = End(route)$.

Note that the set $\mathfrak{G}^{D,R}$ is defined for a pair D, R of a graph and a set of routes, and every nonterminal of $M \in \mathfrak{G}^{D,R}$ is also labeled by such a pair, with the same meaning of R. The meaning of empty R is that we did not extend the root vertex with all appropriate features. Since $P(D) \subseteq C(D)$ and all vertuces from $C(D)$ are extended, an empty R may occur only initially in a derivation.

The desired HRG is a tuple (N, T, P, S). We have already described the set of terminals. Now let us define N as $\{(D(R)) \mid s \in S, D \in Ext_s \vee D = G_0, routes \subseteq Route(D) \vee routes = \emptyset\} \cup \{Start\}$, $type(D(R)) = R$. Recall that $\Gamma = S \cup F \cup Acc$. We are now able to set Acc to $\{D(R)\} \cup \{Start\}$. The set of productions contains $P = \{(D(R), M)\} \cup In$, where $M \in \mathfrak{G}^{D,R}$, and initial productions In are used to start derivations of all possible types. More specifically, $In = \{(Start, M) \mid M \in \mathfrak{G}^{D,\emptyset}\}$.

An aspect of the definition worth mentioning now and which we have not touched before is what initial productions are. If we from the beginning

assume that we are willing to describe every type in a model, then the given definition is exactly what is needed. We did not extend defining types in the root vertex then, and it should be done sometime in order to have a complete description. If however we want our ACG grammar to generate only sentences or some fixed set of types (as it is done in implementations of PET or LKB systems) or some maximal configurations like in [56], it could be fixed locally in the definition of initial productions.

Fact. The HRG constructed is finite.

Proof. The set T is the same as $S \cup F$. Nonterminals are defined by some $D \in Ext_s$ and a set of routes in D. By Lemma 3.11, Ext_s is finite and thus so is $\mathfrak{G}^{D,R}$ because for each vertex $v \in P(D)$ there is only finite choice of nonterminals to be rooted there. Although $Route(D)$ does not have to be finite, we may restrict the set of nonterminals to those participating at least in one production □

For any derivation $Start^\bullet \Rightarrow H_1 \cdots \Rightarrow H_k = H$ and any $v \in V(H)$ or $e \in E(H)$, assuming $\beta_i^V : V_{H_i} \to V_{H_{i+1}}$ and $\beta_i^E : E_{H_i} \to E_{H_{i+1}}$, $\gamma_i^V : V_{M_i} \to V_{H_{i+1}}$, $\gamma_i^E : E_{M_i} \to E_{H_{i+1}}$ are all the projections and $p_i = (D_i(R_i), M_i)$, there is a production p_n which introduced this element (v or e). For convenience purposes we may define

$$\delta^V = \gamma_n^V \beta_{n+1}^V \dots \beta_{k-1}^V : V_{M_n} \to V_H,$$
$$\delta^E = \gamma_n^E \beta_{n+1}^E \dots \beta_{k-1}^E : E_{M_n} \to E_H,$$

two maps which give the origin of v or e, $\delta^V(v') = v$, $\delta^E(e') = e$.

Fact. δ^E is always injective, whereas δ^V is injective if all substitutions are plain. Moreover, $nod(\delta^E(e'))(s) = \delta^V(nod(e')(s))$ since any α and γ preserve the incidence function nod.

Lemma 3.34. *Assume $H \in \mathfrak{G}\mathfrak{r}$ is derived from $Start^\bullet$. For every nonterminal e rooted in v, $lab(e) = D(R)$ in H, if there is a production $p = (D(R), M)$, then the substitution of M in H for e is plain.*

Proof. Let the derivation of H be $Start^\bullet \Rightarrow H_1 \cdots \Rightarrow H_k = H$ of minimal length for which the conclusion is false. Consider δ^E and δ^V which introduce e. Then, nonterminal $e \in E_H$ is $\delta^E(e')$ for some $e' \in E_{M_n}$. Then, $nod_H(e)(s) = \delta^V(nod(e')(s)) = \delta^V(End_{v'}(s))$, where $End_{v'}(s)$ belongs to $V(D_n) = V(M_n)$ and $v = nod(e)(:) = \delta^V(End_{v'}(:)) = \delta^V(v')$, e' is rooted in v'. At the same time $ext_M(s) = End_D(s) \in V(D) = V(M)$. By construction D is compatible with D_n in v', thus for every s_1, s_2 in $type(e) =$

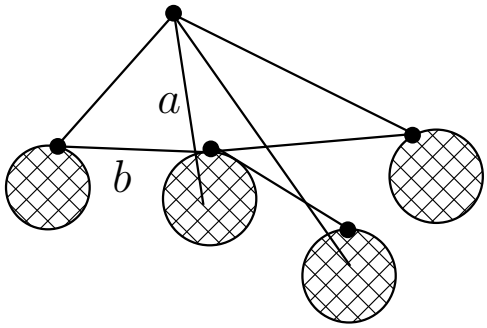

Figure 4: Schematic representation of a hypergraph which can be derived by the constructed HRG

$type(e') = Route(D) \cap Route_{v'}(D_n)$ holds: $nod_H(e)(s_1) = nod_H(e)(s_2)$ iff $End_{v'}(s_1) = End_{v'}(s_2)$ iff $End_D(s_1) = End_D(s_2)$ iff $ext_M(s_1) = ext_M(s_2)$. Therefore, the last substitution is plain and the conclusion of the lemma always holds. □

The next couple of lemmas and definitions all aim at proving Lemma 3.39. They are technical and do not deserve any specific comments for each one separately. To put together the ideas expressed by the lemmas, they give a rough description of graphs, generated by our HRG. In such a graph nonterminals may not have common nodes. Then, any edge either connects two nodes of one terminal and is included in some path of the corresponding accessing R (inside a circle), or it connects two nonterminals, starts in the root of one of them and is not in R, or it does not start in a node of a terminal. Schematically, a derived graph looks like the one in Figure 4. The checked circles represent nonterminals and their nodes. They may not intersect. The edge a performs the function of passing the information above and the edge b is the only allowed way of communication between nonterminals.

Definition 3.35. A hypergraph $G \in \mathfrak{G}$ is **tree-like** if for every two different non-terminals n_1 and n_2 never holds $nod(n_1)(s_1) = nod(n_2)(s_2)$.

Lemma 3.36. *If G_2 is derived from $Start^\bullet$ then G_2 is tree-like.*

Proof. Assume the lemma does not hold and G_2 has the minimal length of the derivation with this property. Let $p = (D(R), M)$ and n be a nonterminal of G_1, $lab(n) = D(R)$ and and $G_1 \Rightarrow_p G_2$. Note, that the base case, i.e. derivation of length zero, will also be considered along the lines.

There are several case of where n_1 and n_2 may come from. First, let us assume that $n_1 = \beta^1_E(n'_1)$, $n_2 = \beta^1_E(n'_2)$, then $nod(n_1)(s_1) = \beta^1_V(nod(n'_1)(s_1))$, $nod(n_2)(s_2) = \beta^1_V(nod(n'_2)(s_2))$. Since β^1_V is injective $nod(n'_1)(s'_1) = nod(n'_2)(s_2)$ and it contradicts to minimality.

In case, however, both $n_1 = \beta^2_E(n'_1)$, $n_2 = \beta^2_E(n'_2)$ come from M, $n'_2 \in E(M)$, $n'_1 \in E(M)$ we have $nod(n'_1)(s_1) = nod(n'_2)(s_2)$. While $M \in \mathfrak{G}^{D,R}$, $nod(n'_1)(s_1) = End_{v'_1}(s_1)$ and $nod(n'_2)(s_2) = End_{v'_2}(s_2)$. Therefore, we now have $End_{v'_1}(s_1) = End_{v'_2}(s_2)$ for different v'_1 and v'_2 in $P(D)$. Since the HPSG has non-interactive derivations, $End_{v'_1}(s_1) \in V(D{\cap}_{v_1})$, $End_{v'_2}(s_2) \in V(D{\cap}_{v_2})$ it contradicts to $V(D{\cap}_{v_1}) \cap V(D{\cap}_{v_2}) = \emptyset$. Note, that, if we were interested in the result of application of an initial production, an almost identical argument would apply. There would be a couple of additional nonterminals for each appropriate feature, not included yet in the corresponding extended defining graph. So, thereby we also proved the base case.

If now $n_1 = \beta^1_E(n'_1)$, $n_2 = \beta^2_E(n'_2)$, then $nod(n_1)(s_1) = \beta^1_V(nod(n'_1)(s_1))$. For n_2 we get $nod(n_2)(s_2) = \beta^2_V(nod(n'_2)(s_2))$, which leads to $nod(n'_1)(s_1) = nod(n)(s)$ for some $s \in type(n)$, therefore G_1 is not tree-like. □

Lemma 3.37. *Assume $G \in \mathfrak{G}$ is derived from $Start^\bullet$, n is a nonterminal of G, $lab(n) = D(R)$ and $e \in E(sk(G))$. If $v_1 = nod(n)(s_1)$, $v_2 = nod(n)(s_2)$, $nod(e)(:) = v_1$, $nod(e)(f) = v_2$, then $s_1.f \in R$.*

Proof. Initially, let $G = M \in \mathfrak{G}^{D_2,\emptyset}$. Let n be rooted in v. This implies $R = Route_v(D_2) \cap Route(D)$, s_1 and s_2 belong to R and $End_v(s_1) = v_1$. As $sk(G) = \mu(D_2) - \emptyset = \mu((D_2)_0)$, then $\eta(sk(G)) = (D_2)_0$ and η maps e into (v_1, v_2, f), hence $s_1.f \in Route_v(D_2)$. Due to non-interactive derivations, either $s_1.f = f_1.f_2 \dots f_k.f \in R$ or $End_v(f_1) = u \in P(D)$, $u \neq v$. But $f_1 \in R$ and $V(D{\cap}_v) \cap V(D{\cap}_u) \supseteq u$, which is prohibited by non-interactive derivations.

If now $G_0 \Rightarrow_p G$ for nonterminal m rooted in u, then we may assume that for G_0 the conclusion is true as well as for M from $p = (D_0(R_0), M)$. Suppose $n = \beta^i_E(n')$ and $e = \beta^j_E(e')$. Under an assumption that $i = j$ we get

$$\beta^i_V(nod(e')(:)) = nod(e)(:) = nod(n)(s_1) = \beta^i_V(nod(n')(s_1)),$$
$$\beta^i_V(nod(e')(f)) = nod(e)(f) = nod(n)(s_2) = \beta^i_V(nod(n')(s_2)).$$

Then $s_1.f \in R$ because the conclusion holds for G_0, β^i_V are injective and the labels of n and n' coincide. If, though, $i = 1$ and $j = 2$, then again

$$\beta^2_V(nod(e')(:)) = \beta^1_V(nod(n')(s_1)),$$

$$\beta^2_V(nod(e')(f)) = \beta^1_V(nod(n')(s_2)).$$

It means $nod(n')(s_1) = nod(m)(s_3)$ and $nod(n')(s_2) = nod(m)(s_4)$. By Lemma 3.36 it is impossible. In case $i = 2$ and $j = 1$ we obtain

$$\beta^1_V(nod(e')(:)) = \beta^2_V(nod(n')(s_1)),$$

$$\beta^1_V(nod(e')(f)) = \beta^2_V(nod(n')(s_2)).$$

This in turn leads to $nod(e')(:) = nod(m)(s_3)$ and $nod(e')(f) = nod(m)(s_4)$. By induction, $s_3.f \in R_0 \subseteq Route(D_0)$. Moreover, $End(s_3) = Ext_{M_0}s_3 = nod(n')(s_1) = End_{v'}(s_1)$, where n' is rooted in v'. Therefore $s_1.f$ is a route in $Route_{v'}(D_0)$ and due to non-interactive derivations we either have $s_1.f \in R$ or prefix f_1 of s_1 goes to $End_{v'}(f_1) = u' \in P(D_0)$ but then $u' \in D_0 \cap_{v'} \cap D_0 \cap_{u'}$. □

Lemma 3.38. *Assume there is a nonterminal n in G, which is derived from $Start^\bullet$, and a terminal e, $f = lab(e) \in F$. Suppose also $nod(e)(:) = nod(n)(s)$ and $nod(e)(f) \neq nod(n)(t)$ for any $t \in type(n)$. Then, s if the empty path $:$ and $f \notin type(n)$.*

Proof. In the derivation of G there are two productions, which introduced e and n (or one production for both). We need to consider three cases depending on the order of their emerging.

- Suppose n was introduces before e. Then, some other later production p introduces e while canceling a nonterminal m. But this means that all nodes of e are in nodes of m. Then, nodes of m intersect with those of n and this contradicts to the property of being tree-like.

- Assume n and e are introduced by one production. Then their pro-images n' and e' both belong to some $D \in Ext_s$. They also have the same property that $nod(e')(x_1) = nod(n')(s)$ and $nod(e')(x_2) \neq nod(n')(t)$ for any $t \in type(n') = type(n)$. Since the HPSG grammar has non-interactive derivations, there is another m', such that $nod(e')(x_2) = nod(m')(t)$. Further $s.f$ cannot be in the influence of n', therefore $nod(e')(x_1)$ and $nod(e')(x_2)$ are both in $P(D)$. In particular, $s =:$.

- Finally, let e appear first as e'. Then at another time, n' appeared while canceling some m'. This leads to the conclusion that n' either had both ends of e' in its nodes or only one of them. The first subcase together with Lemma 3.37 gives us that there is already some $e'' \in E(D)$, such that $e''(1) = End_v(s)$ and $e''(1) = End_v(s.f)$ ($e''(1) = End_v(s)$ and $e''(1) = End_v(s.f)$). The second subcase reduces the problem to identical, but with shorter derivation.

□

Now with all arsenal of facts about the structure of generated trees, we may prove that they meet feature structure requirements and in a hyperedge sense have functional features. It meant that it is possible to convert any generated hypergraph into a graph with functional features. It is not required but it will also be root-connected. Then, if we provide any root vertex it becomes possible to evaluate our HRG on all HPSG models by revealing connection between hypergraphs generated and tokens of those models. In the end we would like to know exactly which tokens we may describe.

Lemma 3.39. *Suppose $D_2 \in \mathfrak{G}$ was derived from $Start^\bullet$, then $sk(D_2)$ meets feature structure requirements and additionally there are no two edges with $nod(e_1)(:) = nod(e_2)(:)$ and $lab(e_1) = lab(e_2)$ (functional features in $\mathfrak{G}$).*

Proof. Suppose the conclusion is false and consider the minimal length derivation for such D_2. Then, $sk(D_1)$ meets feature structure requirements, $v \in V_{D_1}$, n is a nonterminal in E_{D_1}, $lab(n) = D(R_1)$, rooted in v. There is a production $p = (D(R_1), M)$, where $M \in \mathfrak{G}^{D,R_1}$, and $D_1 \Rightarrow_p D_2$.

Assume $e_1, e_2 \in \beta^1_E(E(sk(D_1)))$. If $lab(e_1), lab(e_2)$ are in S, $nod(e_1)(:) = nod(e_2)(:) = v_1 \in V_{D_2}$, then $nod(e_j)(:) = \beta^1_V(nod(e'_j)(:))$. Then, Lemma 3.34 provides us with injective projections and $nod(e'_1)(:) = nod(e'_2)(:)$. Moreover $lab(e_i) = lab(e'_i)$, and since $sk(D_1)$ by inductive supposition satisfies the conclusion of the lemma, $e'_1 = e'_2$, leading to $e_1 = e_2$. If, though $lab(e_1) = lab(e_2)$ are in F and $nod(e_1) = nod(e_2)$, then also $nod(e'_1) = nod(e'_2)$, hence $e'_1 = e'_2$.

If $e_1, e_2 \in \beta^2_E(E(sk(M))) = \beta^2_E(E(\mu(D) - R_1))$ then the same argument applies, because $E(\mu(D) - R_1) \supseteq E(\mu(D))$ and $\mu(D)$ meets feature structure requirements and has functional features.

If then $e_1 \in \beta^1_E(E(sk(D_1)))$, $e_2 \in \beta^2_E(E(sk(M)))$ and their labels belong to S, then

$$\beta^1_V(nod(e'_1)(:)) = nod(e_1)(:) = nod(e_2)(:) = \beta^2_V(nod(e'_2)(:)),$$

where e'_i are pro-images of e_i. By Lemma 3.34 we have a plain substitution and $R(nod(e'_1)(:)) = nod(e'_2)(:)$. Further, $nod(e'_1)(:) = nod(n)(s)$, and $nod(e'_2)(:) = Ext_M(s) = End_D(s)$, $nod(e'_2)(:) = End_D(s)$. For the function δ^V constructed for n holds $nod(n)(s) = \delta^V(End_{v'}(s))$. This means that s lies in $Route_{v'}(D_n) \cap Route(D) = R_1$. But $sk(M) = \mu(D) - R_1$, $nod(e'_2)(:) = End_D(s)$ for $s \in R_1$, which contradicts to the definition of $\mu(D) - R_1$.

If though $f = lab(e'_1) = lab(e'_2) \in F$, then similarly

$$nod_M(e'_2)(:) = End_D(s_1),$$

$$nod_H(e_1)(:) = \delta^V(End_{v'}(s_1)).$$

Since $sk(M) = \mu(D) - R_1$, $\eta(sk(M))$ is defined and the edge e'_2 is mapped into $i_2 = (End_D(s_1), End_D(s_2), f) \in E(D)$. But by Lemma 3.37 and Lemma 3.38, or $s_1.f \in R_1$, contradicting the definition of $\mu(D) - R_1$.

The uniqueness is proven, although we need to establish existence of a 'label' of every edge. Assume $v_1 \in V(D)$, then $v = \beta^1_V(v'_1)$ or $v = \beta^2_V(v'_2)$. In the first case, there is $e \in E(sk(D_1))$, $lab(e) \in S$, $nod(e)(:) = v'_1$ in $sk(D_1)$, hence, $nod(\beta^1_E(e))(:) = \beta^1_V(nod(e)(:)) = v$. In the second case the same does not always hold. It does not only if there is $r \in R_1 = type(n)$, such that $v'_2 = End(r) = Ext_M(r)$. Then, set v'_1 to $nod(n, r)$. From this follows that $v_=\beta^2(v'_2) = \beta^1(v'_1)$ and the case is deduced to the first one. □

Conceptually, Lemma 3.40 should be particularly useful for the proof of Lemma 3.39, but get the opposite direction implication. It claims that the routes by which a nonterminal is accessed during its creation in some $M \in \mathfrak{G}^{D,R}$ remain the only routes it is accessed by until its removal.

Lemma 3.40. *If n is a nonterminal of D_1 rooted in v, $lab(n) = D(R)$, $p = (D(R), M)$, then $Route_v(\eta(sk(D))) \cap Route(D) = R$.*

Proof. Certainly, $R \subseteq Route_v(\eta(sk(D))) \cap Route(D)$ because it was there during the creation of n. Assume, however, there is some r in $Route_v(\eta(sk(D))) \cap Route(D) \backslash R$ such that $r = r_0.f$ and $r_0 \in R$. Use p to derive D_2 from D_1. That means that there is an edge $e_f = (End_{r_0}, End_r, f)$ in D and it is passed to $\mu(D) - R$. There is an edge with the same endpoints and the label in $\eta(sk(D))$, contradicting to the fact that D_2 meets feature structure requirements. □

The last and decisive connection between hypergraphs and graphs that we need to make (the first being that compatibility implies plainess) is that

substitution is equivalent to joint union of skeletons. This fact is not completely trivial as one might immediately think. Technically, non-obvious is what exactly happens with the access routes (K in Lemma 3.41).

Lemma 3.41. *If $D_1 \Rightarrow_p D$, then $\eta(sk(D))$ is isomorphic to $\eta(sk(D_1)) \cup_v K$, where $p = (K(routes), M)$, n is a nonterminal rooted in v, $lab(n) = K(routes)$.*

Proof. First of all $\eta(sk(D_1)) \cup_v K$ is defined only if K is compatible with $\eta(sk(D_1))$. From Lemma 3.40 follows, that $Route_v(\eta(sk(D_1))) \cap Route(K) = routes$. As their common routes did not change from the introduction and K was compatible with some D_n, moreover all substitutions are plain, then K is compatible with $\eta(sk(D_1))$.

Let us start by establishing a bijection between vertices of the two graphs. First,

$$V(\eta(sk(D))) = V(sk(D)) = V(D) = V((D_1)[n/M])) =$$
$$= V(D_1) \cup_R V(M) = V(sk(D_1)) \cup_R V(K).$$

Recall that $R = \{(nod_{D_1}(n)(s), ext_M(s)) \mid s \in type(n) = routes\}$. By definition, $ext_M(s)$ equals $End(s)$ in K. Analogously, $nod(D_1)(n)(s) = \delta^V((End_{v'}(s)) = End_{\delta^V(v')}(s) = End_v(s)$ in $\eta(sk(D))$. This is exactly $V(\eta(sk(D_1)) \cup_v K)$.

Now, consider $E(\eta(sk(D)))$. Assume edge $e = (v_1, v_2, f)$ belongs to $E(\eta(sk(D))) = E(\eta(sk((D_1)[n/M])))$. Then there is $e' \in E(sk((D_1)[n/M]))$, and $nod(e')(:) = v_1$, $nod(e')(f) = v_2$, $lab(e') = f$. The edge e' belongs to $E(sk(D_1))$ or $E(sk(M)) = E(\mu(K) - routes)$.

At the same time e belongs to $E(\eta(sk(D_1)) \cup_v K)$ iff there is either

- $e^\bullet \in E(\eta(sk(D_1)))$, such that $v_1 = \beta^1(e^\bullet(1))$, $v_2 = \beta^1(e^\bullet(2))$, $f = e^\bullet(3)$, that equals to the statement that there is $e' \in sk(D_1)$ with $nod(e')(:) = v_1$, $nod(e')(f) = v_2$, $lab(e') = f$, or
- or $e^\bullet \in E(K)$, such that $e(1) = \beta^2(e^\bullet(1))$, $e(2) = \beta^2(e^\bullet(2))$, $e(3) = e^\bullet(3)$, that equals to the statement that there is $e' \in \mu(K) (= \mu(e^\bullet))$ with $nod(e')(:) = v_1$, $nod(e')(f) = v_2$, $lab(e') = f$.

We observe that $E(\eta(sk(D_1)) \cup_v K)$ contains at least the same edges as $E(\eta(sk(D)))$. What is not immediately clear from here is which edge corresponds to those with $e' \in \mu(K)$ but not $e' \in \mu(K) - routes$. For such an edge e' there are r_1 and $r_2 = r_1.f$ in $routes = type(n)$, $End(r_1) = nod(e')(:)$, $End(r_2) = nod(e')(f)$. There was an edge $e'' \in D_k$ for some D_k at the moment of creation of n. Assume the root of n' was v', then

$End_{v'}(r_1) = nod(e'')(:)$, $End_{v'}(r_2) = nod(e'')(f)$. Then $e^\bullet = \delta^E(e'') \in sk(D_1)$ satisfies $nod(e^\bullet)(:) = \delta^V(nod(e'')(:)) = \delta^V(End_{v'}(r_1)) = nod(n)(r_1)$ and $nod(e^\bullet)(f) = \delta^V(nod(e'')(f)) = \delta^V(End_{v'}(r_2)) = nod(n)(r_2)$. Now, we can see that $\beta^1(e^{bullet}) = \beta^2(e')$, and the case is converted to the first one.

Finally, we need to assure that the labeling functions of $\eta(sk(D))$ and $\eta(sk(D_1)) \cup_v K$ coincide. The proof is parallel to one given for edges. □

3.1.6 Main result

This section is fully devoted to the main theoretical result of the paper and its proof. We want to have a two-directional correspondence between tokens and hypergraphs. For certain class of tokens we manage to indicate exact derivations generating descriptions of this tokens, and for every produced hypergraph we bring a token described by this hypergraph.

Lemma 3.42. *Suppose D_1 describes a token u under vertex r, via $\phi_1 : V(\eta(sk(D_1))) :\to \mathfrak{U}$, $ver \in V(D_1)$ and $v = \phi(ver)$. Assume also that D_2 describes v via ϕ_2 and there is a production $p \in P$, $p = (D_2(R), M)$ and $D_1 \Rightarrow_p D$ for a nonterminal n rooted in ver. Then, there is a morphism ϕ from $\eta(sk(D))$ to $\mathfrak{U}$ such that $\phi(\beta^1_V(r)) = w$.*

Proof. Using Lemma 3.41, we obtain that $\eta(sk(D))$ is isomorphic to $\eta(sk(D_1)) \cup_{ver} D_2$. Each vertex v is either $\beta^1(v'_1)$ or $\beta_2(v_2)$ for $v_1 \in V(\eta(sk(D_1)))$, $v_2 \in V(D_2)$. Define $\phi(v) = \phi_1(v_1)$ or $\phi(v) = \phi_2(v_2)$ respectively. If $\beta^1(v_1) = \beta_2(v_2)$, then $v_2 = Ext_M(s) = End_{D_2}(s)$ and $v_1 = End_{ver}(s)$ in $\eta(sk(D_1))$. By Lemma 3.17, $\phi_2(v_2) = T_I(s)(\phi_2(r(D_2))) = T_I(s)(v)$ and $\phi_1(v_1) = T_I(s)(\phi_1(ver)) = T_I(s)(v)$, we establish correctness of the definition.

Let us check necessary conditions:

- $\phi(r) = \phi_1(r) = u$,
- $\mathfrak{S}(\phi(c)) = \mathfrak{S}(\phi_i(c_i)) = l_i(c_i) = l(\beta^i(c_i)) = l(c)$.
- for every edge $e' = (v, u, f) \in E(\eta(sk(D)))$ there is $e \in E(D_2)$ or $e \in E(\eta(sk(D_1)))$ such that $e(3) = e'(3)$, $e'(k)$, $k \in \{1, 2\}$ equals $\beta^i(e(k))$. Therefore,

$$\mathfrak{F}(f)(\phi(v)) = \mathfrak{F}(\phi_i(v_i)) = \phi_i(u_i) = \phi(u).$$

□

Lemma 3.42 claims that every morphism of a graph produced by the HRG at one step can be extended to a morphism of the graph on the next step of the derivation. Together with Theorem 3.15 and Lemma 3.43 this is basically all one may need to get the first statement of the main theorem. The first two provide a morphism and the lemma that follows discovers that every morphism is actually a complete description, since by construction, we introduced all appropriate features of every vertex as soon as it appeared and only the initial productions required some external interference. We only need the version of Lemma 3.43 (Lemma 3.44) where G does not have nonterminals, but its current formulation gives additional insight of when all features appear (not mentioning that it makes it easy to have a good inductive proof).

Lemma 3.43. *Assume $G \in \mathfrak{G}$, $Start^\bullet \Rightarrow^* G$ and $D = \eta(sk(G))$, then for every vertex $v_1 \in V(D)$, for which there is no nonterminal n of G such that $nod(n)(s) = v_1$ for some $s \in type(n)$, and every f such that $A(f)(l(v_1)) \neq \emptyset$ there is exactly one $e = (v_1, v_2, f) \in E(D)$ and $l(v_2) \in A(f)(l(v_1))$.*

Proof. We proceed by induction by the length of the derivation of G. If the length is one, then $D \in Ext_s$ for some $s \in S$. There, $nod(n)(s) = End_{v'}(s)$ and sets $\{v \mid m = End_{v'}(s), v' \in P(D)\}$ cover $V(D)\backslash\{r\}$. Since the only production which was used is an initial production, then r was supplied with the required edges.

If the derivation

$$S \Rightarrow_{p_1} G_1 \Rightarrow_{p_2} \cdots \Rightarrow_{p_k} G_k \Rightarrow_{p_{k+1}} G_{k+1} = G$$

is of length $k+1$. Then, G_k satisfies the conclusion of the lemma as well as M_k from $p_{k+1} = (D(R), M_k)$ does, for every vertex but the root of D. Let also n be the nonterminal $lab(n) = D(R)$ to which the production applies. By Lemma 3.41, $\eta(sk(G))$ is isomorphic to $\eta(sk(G_k)) \cup_v D$.

Now, pick any $v_1 \in V(\eta(sk(G_k)) \cup_v D)$ and $f \in F$ such that $A(f)(l(v_1)) \neq \emptyset$ and with the property that it is not a node of some nonterminal. Let v_1 be $\beta^1(v_1')$. If it belongs to some non-terminal $nod(m)(s) = v_1'$, then the same holds for v_1 and $\beta^1(m)$, unless $m = n$ and $v_1' = nod(n)(:)$, $v_1 = \beta^1(r(D))$. Therefore, as the graph D is an extended defining graph, there if an edge e' with the right properties, which can be then mapped by β^2. Otherwise, by the inductive assumption there is $e = (v_1', v_2', f) \in E(\eta(sk(G_k)))$ and $l(v_2) \in A(f)(l(v_1))$. Applying μ and then β^1 gives the right edge.

That is it for existence. The uniqueness of such an edge follows from Lemma 3.39. □

Lemma 3.44. *Assume $G \in La(HRG(HPSG))$. If G describes u then G completely describes u.*

Proof. Suppose $u = \phi(n) \in U$ and f, such that $w = \mathfrak{F}(f)(u)$ is defined. Then $l(n) = \mathfrak{S}(u)$, $A(f)(\mathfrak{S}(u)) \neq \emptyset$. By Lemma 3.43, there is $e = (n, m, f) \in E(\nu(G))$. Thus, $\phi(m) = \mathfrak{F}(f)(\phi(n)) = w$. □

Now, in order to learn to reconstruct tokens we will again use Theorem 3.15 and Lemma 3.45. We will construct an interpretation for each hypergraph, which will in fact be 'the hypergraph itself'. Then, this interpretation must be a models, since every token has a graph from Ext_s, which 'contains' it. First, we restore this graph from Ext_s in the following.

Lemma 3.45. *Assume $G \in La(HRG(HPSG))$, $D = \eta(G)$, $v \in V(D)$, then there is $D_0 \in Ext_{l(v)}$ such that $D_0 \subseteq D$ and $r(D_0) = v$.*

Proof. Let the derivation of G be:

$$S \Rightarrow_{p_0} G_1 \Rightarrow_{p_1} \cdots \Rightarrow_{p_{k-1}} G_k = G.$$

Since all substitutions are plain, that the set of vertices and terminal edges of G_i grows monotonically with i. Moreover, in G_1 every vertex, but the root belongs to some nonterminal's nodes and after each production exactly one vertex stops being a node of a nonterminal. Thus, there is one particular p_i, which does this for v: $G_i \Rightarrow_{P_i} G_{i+1}$. Then, by Lemma 3.41 $\eta(sk(G_{i+1}))$ is isomorphic to $\nu(sk(G_{i-1})) \cup_v D$ for some $D \in D_s$. Finally, we put $D_0 = D \subseteq G$ after application of $\eta\delta\mu$. □

Definition 3.46. A token u is **finite** if there is only a finite number of tokens v such that there is a path p, $T_I(path)(u) = v$. A token u is called **acyclic** if there are no paths, such that $T_I(path)(u) = T_I(path.path_2)(u)$ and $path_2$ is not empty.

Complete description of infinite tokens is an impossible task for our generative approach and at the same time, if a token has unpredictable sharing which makes it cyclic, then it has little distinction with an infinite token because we may not embrace such a cycle in one type definition.

Fact. There exist exhaustive models which contain an isomorphic copy of each token of every model.

The fact was proven by King [34] on also Pollard [52]. It is however debatable which exhaustive model is the best one and what should be considered the generative capacity of a grammar. This choice is not for our paper to make.

Theorem 3.47. *Let I be any exhaustive model of HPSG grammar H. Then, for every finite and acyclic token $u \in \mathfrak{U}$ there is a hypergraph $G \in La(HRG(H))$, such that G completely describes u. For every hypergraph $G \in La(HRG(H))$ there is a finite token u which G completely describes.*

Proof. First, it is sufficient to find a graph G which describes u and then by Lemma 3.44 G completely describes u. Moreover, by Lemma 3.42 we only need to provide a derivation where every nonterminal n rooted in v has label $D(R)$ such that D describes v. We need to make sure that every production saves this property. For every production $p = (D_1(R_1), M_1)$ we have choice of different $M \in \mathfrak{G}^{D_1,R_1}$. All such possible M differ in what nonterminals we introduce for every vertex $v \in P(D_1)$. Additionally, D must be compatibly with D_1 in v. As we proved in Theorem 3.15 there is a D which injectively describes $\phi(v)$. At each step we chose exactly this D.

If D_1 describes $\phi(r(D_1))$, then by Lemma 3.19 D is indeed compatible with D_1. The derivation will stop. Otherwise, since each time we release one vertex from being a node of a nonterminal, the hypergraph constantly grows (the number of its vertices). Finally there will be a path p from the root of length more that the number of tokens under v. Then at least for two of the vertices on p will hold $\phi(v_1) = \phi(v_2)$. By Lemma 3.17 $T_I(p_1)(u) = T_I(p_1.p_2)(u)$.

Put $D = \eta(G)$ and let us define an interpretation $I = (\mathfrak{U}, \mathfrak{S}, \mathfrak{F})$. We set $\mathfrak{U} = V(D)$, $\mathfrak{S}(v) = l(v)$, and $\mathfrak{F}(f)(v) = v_2$ for an edge (v, v_2, f) in $E(D)$ (D has functional features, which follows from Lemma 3.43). In order I to be an interpretation it must satisfy $\mathfrak{S}(\mathfrak{F}(f)(v)) \in A(f)(\mathfrak{S}(v))$, which follows from Lemma 3.43 as well.

The map ϕ is defined as $\phi(v) = v$ and is obviously injective and we want to establish that D describes $u = \phi(r(D))$.

- $\phi(r(D)) = \phi(r(D))$;
- $\mathfrak{S}(\phi(v)) = \mathfrak{S}(v) = l(v)$;
- for every edge $e = (v_1, v_2, f)$, $\phi(v_2) = v_2 = \mathfrak{F}(f)(v_1) = \mathfrak{F}(f)(\phi(v_1))$.

For I to be a model, every token v, $l(v) = s$ must satisfy Θ_s. By Theorem 3.15 it is sufficient to prove that some $D_0 \in Ext_s$ describes v. Exploiting Lemma 3.45, we get such a D_0. It describes v via ϕ restricted to $V(D_0)$.

The token u is finite because $\mathfrak{U}$ is finite. Since an exhaustive model contains isomorphic copies of every token of every model, then there is a

token u' isomorphic to u. Thus, G describes u'. And by Lemma 3.43, G completely describes u'. □

3.2 Implementation of an LCFRS parser

We suggest to re-implement a linear contex-free rewriting system parser, because LCFRS is now being under investigation of a number of other authors and there are various approaches proposed of how to improve accuracy as well as effeciency. At the same time implementational properties of ACGs are not well investigated.

LCFRS [62, 63] were introduced as a class of formalisms in which a few other known theories can be encoded, e.g. tree-adjoining grammars. It retains simplicity of context-free grammars and posseses desirable properties of semilinearity and polynomial recognition. LCFRS's can in principle operate on structures more complex than strings and use operations different from concatenation. The significant requirement is though, that all rules must be linear and non-erasing, e.g. they cannot add and remove variables (to be demonstrated in definition).

Therefore, LCFRS is a family of formalisms. In different interpretations it can generate string as well as tree languages. The generalisation over context-free string languages that it makes is achieved by the ability of non-terminals to represent discontinuous segments of a string. This additional ability makes it possible to generate non-context-free languages. All in all, LCFRS generates mildly context-sensitive languages, which are assumed to be the class natural languages belong to.

3.2.1 Definition

Formally, LCFRS is a tuple $G = (N, T, V, P, S)$ where N is a finite set of nonterminal symbols supplied with a function $dim : N \to \mathbb{N}$, $dim(A)$ is called fan-out of A. Sets T and V are terminals and variables respectively, $S \in N$ the start symbol, $dim(S) = 1$, and P is a finite set of rewriting rules. A rule has a non-terminal on the left side and a sequence of non-terminals on the right side:

$$A(\alpha_1, \ldots, \alpha_{dim(A)}) \to A_1(X_1^1, \ldots, X_{dim(A_1)}^1) \ldots A_m(X_1^m, \ldots, X_{dim(A_m)}^m),$$

where $X_i \in V$ are variables and $\alpha_j \in (T \cup V)*$ are terms over this variables and terminals. Every variable on the RHS must occur exactly once on the LHS. The rank of a rule is m, the number of non-terminals in the RHS. The fan-out of a rule is $dim(A)$, where A is the nonterminal symbol on the LHS.

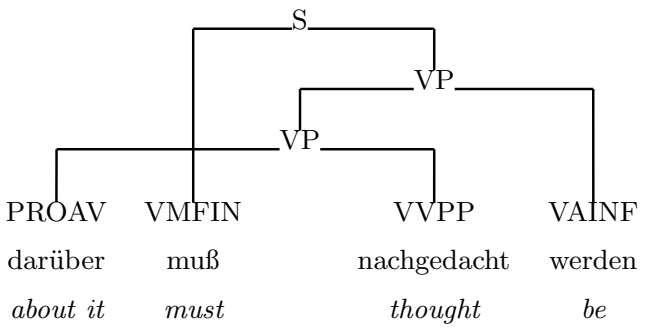

Figure 5: A tree used to extract LCFRS rules

A probabilistic liner context-free rewriting system is a LCFRS together with a function $P \to \mathbb{R}$ wich provides probabilities of the grammar rules.

3.2.2 Grammar extraction

An algorithm to read a PLCFRS grammar from a corpus with crossing branches was proposed in [41]. It is a rather straighforward generalization of the usual algorithm for CFG, except only that discontinuities must be properly represented. Here, is a short description of the algorithm with an example. Consider the discontinuous tree from Figure 5.

It has a discontinuous VP node *darüber nachgedacht*. For each node in a tree we construct a rule, which may be used to build this constituent from its daughters. Assume the constituent contains the words w_1, w_2, ..., w_k. For each $1 \leq i \leq k$ we create a variable X_i. Each daughter N is assigned the sequence of varibles $\{X_j \mid w_j \in N\}$ where by $w_j \in N$ we mean that the word w_j is a descendant of N. The LHS of the new rule has the same label as the parent node and takes the sequence of all the variables such that X_i preceeds X_j if $i < j$. For instance the other VP *darüber nachgedacht werden* is initially assigned variables $\{X_1, X_2, X_3\}$ corresponding to the three words of the phrase. Its VP daughter is assinged $\{X_1, X_2\}$ and VAFIN daughter is assigned $\{X_3\}$. In the next step every pair of variables whose words are adjacent is merged. The merging is repeated with new variables representing now subsrings of the original string $w_1 w_2 \dots w_k$ until there is nothing to merge. Finally variable of the LHS of the rule are concatenated if they are

adjacent in the string. In our example the upper-most rule

$$S(X_1, X_2, X_3, X_4) \rightarrow VP(X_1, X_3, X_4)VMFIN(X_2)$$

is reduced to

$$S(X_1, X_2, X_3) \rightarrow VP(X_1, X_3)VMFIN(X_2)$$

and then to

$$S(X_1X_2X_3) \rightarrow VP(X_1, X_3)VMFIN(X_2)$$

The grammar extracted from our tree is then:

$lexical\ rules$	$non-lexical\ rules$
$PROAV(darüber) \rightarrow \epsilon$	$S(X_1X_2X_3) \rightarrow VP(X_1, X_3)VMFIN(X_2)$
$VMFIN(muß) \rightarrow \epsilon$	$VP(X_1, X_2X_3) \rightarrow VP(X_1, X_2)VAINF(X_2)$
$VVPP(nachgedacht) \rightarrow \epsilon$	$VP(X_1, X_2) \rightarrow PROAV(X_1)VVPP(X_2)$
$VAINF(werden) \rightarrow \epsilon$	

One can see that the grammar extracted from a corpus can be used to analyse this corpus, providing correctness to the method. Only one additional non-empirical generalization is made — context-freeness of derivations, which is an essential part of LCFRS. No unneseccary assumptions are made about discontinuities, e.g. we could have as well extracted the rule

$$VP(X_1, X_2, X_3) \rightarrow VP(X_1, X_2)VAINF(X_2)$$

for *darüber nachgedacht werden*.

The ability to systematically represent discontinuities in a language model is a great advantage, which allows reading grammars directly from treebanks with crossing branches. Usually in treebanks used for probabilistic context-free parsing long dependencies are reduced to minimum, whereas here we gain more control over them. Moreover, a better consent can be achieved among annotators when it is always possible to directly represent argument relations. For instance the topicalized *Bus* in Example 4 is a complement of *gefahren* and we will extract the rule $VP(X_1, X_2) \rightarrow NP(X_1)\ VVPP(X_2)$.

(4) *Bus ist Karl gefahren*
Bus has Karl driven
Karl has driven a bus

An ordered rule is a rule such that for every nonterminal A in the RHS and every pair of variables X_1 and X_2, X_1 preceeds X_2 in the RHS, iff

X_1 preceeds X_2 in the LHS. While extracting the grammar we may only form ordered rules. This assures that even non-concatenated variables of a nonterminal are always in a preceedence relation, though not a direct preceedence. Moreover, we order the RHS nonterminals by the position of their left-most continuous segment. It allows for more concise presentation with improved visial perception of rules and also a tighter connection to CGF rules

To obtain the probabilies for our language model we use simple maximum likelihood estimates. During grammar extraction we count occurencies of every grammar rule, and then after all preprocessing steps the probability of a rule $A(\dots) \to B(\dots)\ C(\dots)$ is estimated as

$$\frac{\#(A(\dots) \to B(\dots)\ C(\dots))}{\sum \#(A(\dots) \to \dots)}.$$

where the sum is taken over all rules with the LHS nonterminal A. It is worth noticing here that even though discontinuous and continuous nodes in the corpora may have the same label, e.g. NPs can be continuous or discontinuous, the LCFRS formal definition demands each nonterminal to have exactly one fan-out. In particular, we must introduce different nonterminals for continuous and discontinuous nodes in the corpus with the same label, e.g. NP_1 for continuous NPs, or NP_3 for NPs containing three non-adjacent parts. Then the impact of this on the probability estimation is that for continuous and discontinuous rules probabilities are counted separately, $dim(A)$ is the same for all entrances of A in the fraction.

3.2.3 Preprocessing

After extracting the grammar, there are several techniques to improve its quality including lexicalization, markovization, and category split.

Lexicalization is a widely used method first successfully applied to PCFG parsing of English (Collins [7]). It consists in adding additional information to non-terminal nodes, namely its lexical head. In particular, it can be useful to recognize subcategorization patters, as the plain model does not provide any information of this kind and, thus, is incapable of distinguishing, for instance, transitive and intransitive verbs. Lexicalization makes the grammar much more sparse, thus, it requires more training data and cannot also be used on a par with certain other modifications.

Markovization is another technique to improve accuracy, previously applied to PCFG parsing [7, 35]. The problem it solves is two-folded. First, due to effeciency constraints or limited size of the training corpus, we cannot

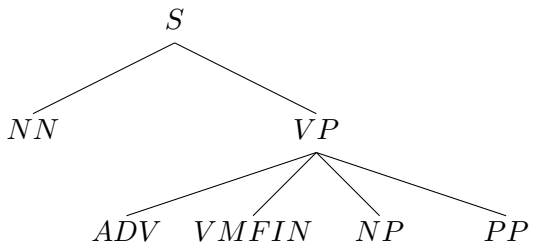

Figure 6: Non-markovized tree

find some (sometimes many) rules that we need to parse an unseen sentence, hence sparseness of the data and low coverage. On the other side, the limited amount of (thus not very informative) categories, generalize to strongly (the same example as with lexicalization applies here). Markovization consists in enriching categories with contextual information. There are two kinds of context: vertical and horizontal. Vertical context contains the first v categories on the path from a node to the root and horizontal context containg the categories of h neighbours of a node.

Vertical context is usuallty more effective with PCFG parsing, both English and German, than horizontal. That suggests that the trainig data is sufficient to more extent than the categories are informative. However with LCFRS the two classes of continuous and dicontinuous rules may expose behaviour different from one another, especially taking into account that dicontinuities may be underrepresented in the training data. We hypothesize that horizontal context may have a stronger effect on discontinuous constituents.

Consider the tree in Figure 6 and its markovized counterpart in Figure 7. In the markovized tree, the square brackets identify the head node, triangular brackets show that the node is intermediate. Superscripts are used to show the parent node, which is included as the vertical context. They are discharged from lexical entries, because they are not given to the parser as input. The intermidiate nodes are used to limit horizontal context. They group the head node first with all h its right neighbours and then with its h left neighbours. Markovization can be considered as partial binarization of the grammar, which however also affects the extracted probabilities.

Another possible solution to the coarse label deficiency is to split categories into more informative sub-categories, according to manually or auto-

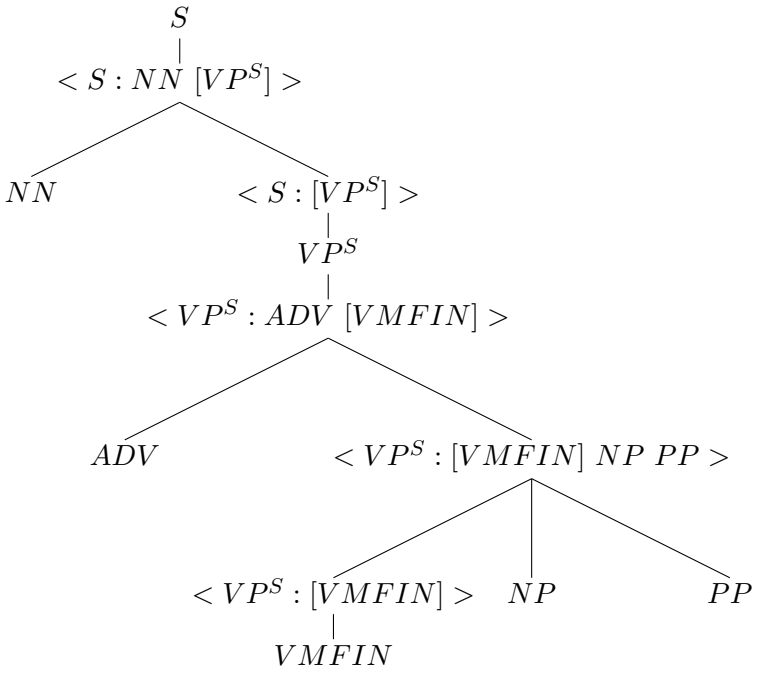

Figure 7: Markovized tree, $h = 2$, $v = 2$

matically chosen criteria. This is a less productive approach, though it may have more linguistic support and produce clearer descriptions. In [40] they split VPs into VPs with participles, with *zu*-infinitives and with infinitives (without *zu*). Nodes labeld S can be split into relative clauses and just sentenses.

Markovization, lexicalization and node split were successfully applied to PCFG parsing of English. Markovization, for instance, is reported to improve accuracy by approximately 6%. As parsing of German is concerned, these techniques do not exhibit the same influence. Overall, markivization adds 3%, for lexicalization the results in the literature flactuate between 0% and 2% and node split improvement is not very significant.

3.2.4 Binarization

Every LCFRS can always be binarized, i.e. all rules will have at most rank two. This allows for quicker and simpler parsing algorithms. Accuracy is not affected by binarization, because the initial set of rules is not changed and the probabilities remain the same, all intermediate rules having the probaility 1.

The binarization can be done in various ways. The most strainghtforward approach is to binarize from left to right (or right to left). Then the rule $S \to VP\ NP\ PP$ will be modified to $S \to VP\ (NP + PP)$ and $(NP + PP) \to NP\ PP$. This strategy does not optimize the parsing complexity and does not form coherent structures. A better way is head-driven (head-outward) binarization, i.e. the head of the rule consumes its sisters one by one, usually first from one side, then the other. Fewer rules are created and they are more sensible. Note that markovization may have already introduced some or the intermadiate substructures.

There is also an optimal binarization for LCFRS which creates a grammar with rules of minimal possible fan-out [22]. Althought the fan-out affects the parsing complexity, other binarization techniques can lead to better performance. In [20] an algorithm miminising the theoretical parsing complexity was proposed, however it only gives the best theoretical complexity without giving any respect to distribution of the grammar rules. A comparison of binarization methods was presented in [60].

3.2.5 Parsing

As in its essence derivations in an LCFRS analysis are context-free, usage of a chart for representing partial parsing results is possible, allowing for

polinomial parsing time. The time to parse a sentense ω with a binarized grammar G is $O(|G| \cdot |\omega|^p))$, where $|\omega|$ is the length of the sentence and $|G|$ is the size of the grammar. The power p is the maximal parsing complexity of the rules in G. The parsing complexity in turn is defined as $f_L + f_{R1} + f_{R2}$, where f_L is the fan-out of the left-hand side of the rule and f_{R1} and f_{R2} are the fan-outs of the right-hand side nonterminals.

We implement a modified probabilistic CYK parser for context-free grammars in Chomsky normal form [59, 64]. The technique which enables the genaralization made is weighted deductive systems. In general weighted deductive systems are logical systems consisting of axioms, which are logical formulae, and inference rules, which explain how more formulae can be obtained from the given ones.

The CYK algorithm in the form of a deductive system [47] works as follows. One by one, it builds items, which are partial analises of the input string. In the context-free case an item is a triple $[A, i, j]$, where A is a category label, $i \leq j$ are indices denoting two positions in the string. Assume the original string is $w_1 w_2 \dots w_k$. Then, if we form an item $[A, i, j]$ it means that there is complete parse of the substring $w_{i+1} w_{i+2} \dots w_j$ with the root label A. The deductive system, then, consists of:

- **Axioms**: $[A, i, i+1]$ for each grammar rule $A \to w_{i+1}$
- **Goals**: $[S, 0, k]$ if S is the sentence label
- **Inference rules**: $\frac{[B,i,j] \;\; [C,j,l]}{[A,i,l]}$ for each grammar rule $A \to B \; C$

Here the goal items are those that we are searching for. Since there are parses of every substring w_{i+1} rooted in A for each grammar rule $A \to w_{i+1}$, then the axioms can always be formed. Moreover, if we have two parses for adjacent substrings $w_{i+1} \dots w_j$ and $w_{j+1} \dots w_l$ and a grammar rule $A \to B \; C$, where B and C are the roots of these parses, then there is also a parse of the substring $w_{i+1} \dots w_l$. Thus, the inference rules exactly tell us how to generate items.

A weighted decuctive system also assignes weights to its items. They signify the cost of creation of an item. In our case the weights can be substituted by the probabilities of the corresponding parses with a probabilistic context-free grammar. To avoid underflow, we apply the logarithm to the probabilities. Then, in the deductive system we have:

- **Axioms**:

$$|\log(p)| : [A, i, i+1]$$

  for each grammar rule $p : A \to w_{i+1}$

- **Inference rules**:

$$\frac{x_1 : [B, i, j] \; x_2 : [C, j, l]}{x_1 + x_2 + |\log(p)| : [A, i, l]}$$

  for each grammar rule $p : A \to B \; C$

Here, $w : I$ denotes an item I with its weight equal to w, and $p : A \to B$ means that the rule $A \to B$ has that probability p in the PCFG that we have. Assuming independence of grammar rules application one can conclude the in this deduction system an item $|\log(p)| : [A, i, j]$ is formed, iff there is a partial parse of $w_i \ldots w_j$ rooted in A whose probability is p.

Thus far we can only do exhaustive parsing. But first, it is not always needed, and second, exhaustive parsing is slower than searching only for the best parse. When only looking for the best parse one may need an algorithm which decides in which order new items must be created.

We are only interested in the most probable parse and for that one can use Knuth's algorithm. It requires two sets: chart and agenda. The chart contains items whose minimal possible weight we already obtained. For instance, for every rule $p_i : A_i \to w_i$ we put the items $|\log(p_i)| : [A_i, i, i + 1]$ in the chart because there is no other parse of the substrings w_i with a smaller weight and rooted in A_i. The agenda contains the items whose wieght can still be changed. The algorithm is given below.

```
while A ≠ ∅ do
    remove the best item x : I from A and add it to C;
    if I is a goal item then
        stop and output true;
    else
        for all items y : J deduced from I and items in C do
            if C contains J then
                do nothing;
            else
                if A contains z : J for some z then
                    if z ≤ y then
                        do nothin;
                    else
                        update the weight of J in A to y;
                    end
                else
                    add y : J to A;
                end
            end
        end
    end
end
```

Algorithm 1: Knuth's algorithm

We proceed as follows: take the best item from the agenda and try to combine it with all items of the chart, the results are added to the agenda, and the item is moved to the chart. We stop if the best item on the agenda is a goal item. The algorithm returns the most probable item, and its correctness is provided by monotonicity of the weighting function, i.e. every new item has higher weight than the items from which it was formed, and only the best item is taken from the agenda at each time. To reconstruct the parse, one needs to store in every item pointers to the ones that were used to form it.

LCFRS parsing can be done in an almost identical fashion. However the form of items must be changed. Now an item is a tuple $[A, \rho]$, where the range vector $\rho \in (\mathbb{N} \times \mathbb{N})^{dim(A)}$ is a sequence of left and right coordinates of the segments (ranges) that A spans over. For instance, in Figure 5, to parse the sentence we would at least need to find the items $[VP, (0,1), (2,3)]$, $[VP, (0,1), (2,4)]$, and $[S, (0,4)]$ (assuming *darüber*$= w_1$). Obviously, if all symbols A have $dim(A) = 1$, then all items have the exact form as those

used for CFG parsing.

Any inference rule

$$\frac{x_1 : [B, \rho_B] \; x_2 : [C, \rho_C]}{x_1 + x_2 + |\log(p)| : [A, \rho_A]}$$

is obtained from an instanciation $A(\rho_A) \rightarrow B(\rho_B) \; C(\rho_C)$ of a grammar rule. To obtain an instanciation of a rule one must substitute all variables by ranges and concatenation of variables by concatenation of ranges. For example, the rule $S(X_1X_2X_3) \rightarrow VP(X_1X_3) \; VMFIN(X_2)$ can be instantiated as $S((0,4)) \rightarrow VP((0,1),(2,4)) \; VMFIN((1,2))$. Similarily, the axioms are of the form $0 : [A, (i, i+1)]$, for each rule $A(w_{i+1}) \rightarrow \epsilon$ if we use gold POS tags. Or, alternatively, another non-zero weight may be applied if the POS tags are not given in the input. It is also worth noticing that we do not allow instantiations like $[VP, (0,1), (1,2)]$, when distinct ranges in an nonterminal can theoretically be concatenated. By doing so with the grammars extracted from our corpora we do not miss any generalization, because there are no rules that can cancel such zero-length gaps. At the same time a substantial speed-up is archieved, because of the reduction in the size of the chart.

As an example, assume the grammar consists of the following rules:

$$\begin{array}{rl}
0.33 : S(X_1X_2X_3X_4) \rightarrow A(X_1, X_3) \; B(X_2, X_4) & 1.0 : C(c) \rightarrow \epsilon \\
0.33 : S(X_1) \rightarrow D(X_1) & 1.0 : D(d) \rightarrow \epsilon \\
0.33 : S(X_1) \rightarrow E(X_1) & 1.0 : E(e) \rightarrow \epsilon \\
0.5 : A(X_1, X_2) \rightarrow C(X_1) \; C(X_2) & \\
0.5 : A(X_1, X_2) \rightarrow E(X_3) \; E(X_4) & \\
1.0 : B(X_1, X_2) \rightarrow D(X_1) \; D(X_2) &
\end{array}$$

The parser is given the input line *cdcd*. Figure 8 shows in steps how the parsing algorithm works. The table shows the changes of the chart, which grows monotonously one item at a time and all the newcoming items have higher or the same weights as any chart item. The agenda is a priority queue. We insert items in the agenda and we also need to pop the best item, i.e. the item with the lowest weight. For efficient parsing these operations better be done in sub-linear time.

The algorithms outputs $\log(12) : [S, (0,4)]$ with the poiners to the items used to form it.

This algorithm is rather simple to implement and it is also intuitively clear. The most important implementational properties of such parsing algorithm are the speed of locating items in the chart and agenda and the

	chart	*agenda*
0		$0:[C,(0,1)], 0:[D,(1,2)],$ $0:[C,(2,3)], 0:[D,(3,4)],$
1	$0:[C,(0,1)]$	$0:[D,(1,2)], 0:[C,(2,3)],$ $0:[D,(3,4)]$
2	$0:[C,(0,1)], 0:[D,(1,2)]$	$0:[C,(2,3)], 0:[D,(3,4)],$ $\log(3):[S,(1,2)]$
3	$0:[C,(0,1)], 0:[D,(1,2)],$ $0:[C,(2,3)],$	$0:[D,(3,4)], \log(2):[S,(1,2)]$ $\log(2):[A,(0,1),(2,3)],$
4	$0:[C,(0,1)], 0:[D,(1,2)],$ $0:[C,(2,3)], 0:[D,(3,4)]$	$\log(3):[S,(1,2)], \log(2):[A,(0,1),(2,3)],$ $\log(3):[S,(3,4)], \log(2):[B,(1,2),(3,4)],$
5	$0:[C,(0,1)], 0:[D,(1,2)],$ $0:[C,(2,3)], 0:[D,(3,4)],$ $\log(2):[A,(0,1),(2,3)]$	$\log(3):[S,(1,2)], \log(3):[S,(3,4)],$ $\log(2):[B,(1,2),(3,4)]$
6	$0:[C,(0,1)], 0:[D,(1,2)],$ $0:[C,(2,3)], 0:[D,(3,4)],$ $\log(2):[A,(0,1),(2,3)],$ $\log(2):[B,(1,2),(3,4)]$	$\log(3):[S,(1,2)], \log(3):[S,(3,4)],$ $\log(12):[S,(0,4)]$
7, 8	$0:[C,(0,1)], 0:[D,(1,2)],$ $0:[C,(2,3)], 0:[D,(3,4)],$ $\log(2):[A,(0,1),(2,3)],$ $\log(2):[B,(1,2),(3,4)]$ $\log(3):[S,(1,2)],$ $\log(3):[S,(3,4)].$	$\log(12):[S,(0,4)]$

Figure 8: Parsing with LCFRS

quantity of unused items in them. In [4] a range of parsing strategies optimized with respect to these properties was presented. Different strategies use other types of items and a bigger number of inference rules. More refined items can improve the speed of locating them and also cancel earlier some redundancies. A speed-up of up to 20 times is archieved there.

Although it might be more practical to use one of the proposed algorithms, we restrain outselves to the basic algorithm for its simplicity. Moreover, for the data we worked on, we did not face any serious implementational memory or time deficiencies. As was already mentioned, the absolute charactiristics of our implementation only play a secondary role, and parsing strategies do not affect the quality of the parse either.

3.2.6 Baseline

The TIGER corpus was parsed using many formalisms as we show in Section 2. We can use as a baseline the results of one of the works discussed, but it would be impossible to do direct comparison for very different parsers. The closest to the LCFRS formalism is CFG. The comparison of the two will exactly show what kind of impact LCFRS makes. Fortunately, the parser that we implement can be used to parse with CFG without any change. The only discontinuities the parser can possibly create have to be in the grammar rules. Parsing under the same conditions allows for both accuracy and performance comparison.

To obtain a context-free grammar from a corpus with crossing branches there exist three natural methods based on either copying nodes or moving them. When accepting the node-raising approach, one takes a discontinuous node and re-attaches its daughters to the node's parent until the node becomes continuous. Consider the tree in Figure 9. It has two discontinuous VP nodes. First, the node labelled $PROAV$ is raised to the higher VP node and, then, to the S node. The resulting tree is given in Figure 10. It can be used both for CFG training and testing.

When doing node-splitting and node-adding, one splits a node according to its continuous segments, or adds a copy of a node respectively (see [27] for more detail). The accuracy of all three approaches differs by at most 3 percent, node-raising being the most accurate. At the same time only node-splitting provides a method to recover the intended discontinuous trees, but at the price of worse parsing performance and heavier interference with the rule frequency distribution.

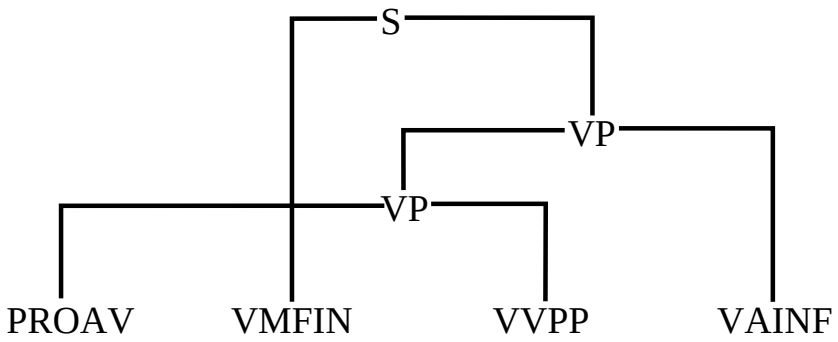

Figure 9: Discontinuous tree

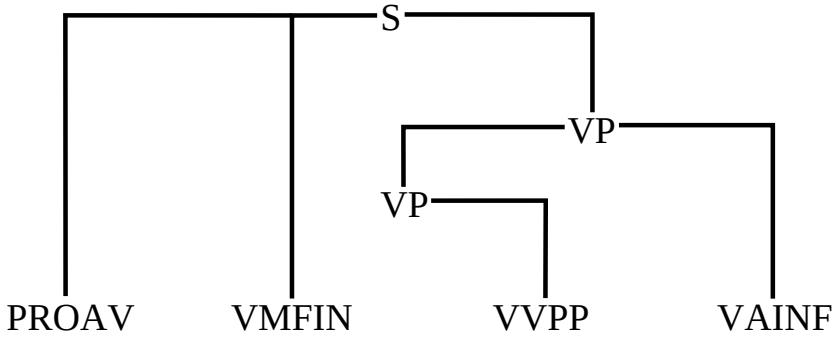

Figure 10: Continuous tree

3.2.7 Evaluation

In [39] three different evaluation metrics were considered to evaluate the performance of a PLCFRS: EVALB, tree-distance measure and dependency evaluation.

EVALB is a modification of the PARSEVAL measure to account for possible crossing branches. This measure compares two sets of tuples for the result of parsing and the expected result. Each tuple

$$(Label, \{(l_1, r_1), (l_2, r_2), \ldots, (l_n, r_n)\})$$

consist of a node label and the continuous segments of the sentence it spans over. It identifies one constituent of the tree, so that continuous constituents have one segment and discontinuous have more than one. Given the set of tuples P of the parse and the set of tuples G of the gold standard tree the recall is defined as $r = \frac{\#(P \cap G)}{\#G}$, the precision is $p = \frac{\#(P \cap G)}{\#P}$, and the f-score $\frac{2*p*r}{p+r}$. Even though EVALB was criticized in [28], it allows one to compare results with previous work in PCFG as well as LCFRS. Moreover, we are solely interested in the relative changes in accuracy, while the absolute results are less important.

Tree-distance measure [2] compates two trees on the basis of operations of node deletion, insertion and label swap. The more operations are needed to convert one tree into another, the more different the trees are. This measure compares any trees and is independent from the notion of discontinuity, thus it can nicely be used for LCFRS parsing evaluation. It is a less linguistically-oriented measure, though it is very similar to EVALB. A direct node-wise analysis and intuitively correct evaluation of how good the parser captures discontinuities are not possible.

Dependency evaluation [37] consists in comparison of dependency graphs. It evaluates pair-wise connections of the words in a string. However, to obtain dependency graphs one must identify the dependencies in the phrase-structure tree, which makes this technique a bit subjective. The most important difference from constituent evaluation is that there is no hierarchical grouping of words. Therefore the flat structures of the TIGER corpus will be punished less in the case of a small mistake.

Any of the tree measures could be possibly used. We are mostly interested in revealing specificities of LCFRS and in particular its performance on discontinuities. Therefore, the choice of EVALB is clearly justified.

4 Empirical evaluation

4.1 Data

In this section we present the set-up and results of our experiments. First, we describe the common setting of all the experiments. Then we talk about our general observations concerning the corpus and the grammars extracted. After that, we have a closer look at different modifications of the initial grammar and the corpora.

The treebank we use is the TIGER treebank(Section 2.2). In the version of the corpus that we use, there are overall 50429 sentences, which we reduce to exactly 50k. For different experiments we may need the full version or a smaller subcorpus. Therefore we take the first 18k sentences, which is approximately the size of the NEGRA corpus (minus 2k for testing), to be the small training corpus (denoted NEGRA here). The bigger corpus is at least two times larger and comprises the first 48k sentences (TIGER). The sentences with IDs from 48001 to 50000 are chosen for testing. The size of the test set is, thus, 10% of NEGRA and 5% of TIGER. The numbers coincide with those of previous experiments in this area and reside in the usual range accepted for our task. The test set is denoted as TEST and the development set DEV is taken to be sentences from 46001 to 48000.

The original data is not ideal in several respects. The sentence annotation includes punctuations which require additional attention. First, if we use the original annotation where punctuations are usually attached high at the sentence level, then the amount of discontinuous rules extracted grows significantly, and importantly they do not really correspond to any 'true' discontinuities. Another approach is to re-attach punctuation low to nearest nodes with the help of a heuristics. We do not expect any noticeble improvement of parsing results after that. And even the opposite, as we will discover further in the experiments, the problem of sparsity already clearly manifests itself, and adding new information with unknown impact can complicate the problem. There are also a few sentences containing nothing else but punctuation and nonwords, which we remove from consideration. Therefore, we exclude all punctuation from the annotation. Neither do we add additional punctuation.

Another dubious piece of annotation is the artificial root nodes labled $VROOT$ that appear in many sentences. The annotators were allowed to put prhrases of any entry in the treebank under the $VROOT$ if they, for instance, did not form a sentence. Consider the tree in Figure 11. Notice that the discontinuous PP disappears if the punctuation is removed and we

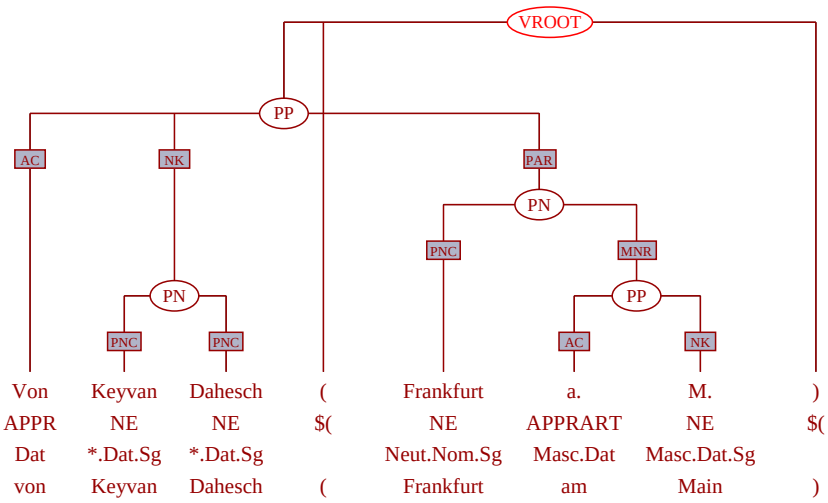

Figure 11: Discontinuous tree

have to create an artificial rule $VROOT \rightarrow PP$ in the grammar.

A detailed consideration shows that this artificial category was not created by a good, consistent pattern. Its usage is either triggered by the newspaper specific extracts of text, like, for example, advertisments, or can be explained on the pragmatical level, which as we think should not be handled by a parser anyway. A radical but simple solution is chosen, to retain in TEST only utterances rooted in S or direct daughters of the $VROOT$ node, also labeled by S. We also create no corresponding rules with $VROOT$. TEST reduces to 1581 sentences. This decision also clarifies how the goal items of the parser must look like, namely exactly $[S, 0, k]$ where k is the sentence length. At the same time, evaluation seems to be a bit less fair because the parser is always given the absolute true value of the sentence root and, thus, in every parse there is always one correctly parsed constituent. Effeciency could also suffer from passive items labeled $VROOT$ and now we avoid this problem.

To provide a baseline we prepare modifications of each set with resolved crossing branches. We resort to the node raising technique as describes in Section 3.2.6. All the sets including the test set are then modified. They re-

ceive the names NEGRA-CFG, TIGER-CFG, and TEST-CFG respectively. We do not recover the original discontinuous structure and evaluate on the modified set.

Due to time and computer performance capabilities we have to restrict the maximal size of a sentense fed to the parser to 25. Finally, the TEST set contains 1357 utterances. The average sentence length in TIGER is 15.23, thus we are covering much longer sentences than an average one.

4.2 General overview

Now we extract the grammars from the training sets. The average RHS length computed for TIGER is 4.28. At the same time the average tree depth is 5.1, and 4.66 for sentences of length no more than 25. As expected we may conclude that the trees are rather flat. The proportion $\frac{\#nonterminals}{SentenceLength}$ equals 0.49.

Then, the grammar rules are binarized using the head-outward binarization. The following table summarizes the data after all preprocessing steps.

	TIGER	TIGER-CFG	NEGRA	NEGRA-CFG
$Size$	47957	47957	17980	17980
$Rules$	32728	30977	17399	16290
$Rules - bin$	51413	56709	27697	29814
$LHSfan - out$	1.2	1.0	1.19	1.0

Discontinuous trees comprise 27.8% of the corpus and discontinuous nodes make up 1.8% of all nodes and 5.5% of all nonlexical nodes. The number of rules with $dim(LHS) > 1$ is 5649 (17.2% of all rules, in TIGER). The average frequency of discontinuous rules is 3.44, whereas for the continuous rules it is 12.4. We observe a lack of frequent discontinuous rules, as compared to continuous ones. That was of course expected as discontinuities are rarer than continuous nodes. At the same time every sixth rule and only every 20th node is discontinuous, hence, greater sparseness. This may lead to poorer coverage and precision on discontinuous constituents.

4.3 Coverage

We perform the following experiment. We want to predict to which extend this underrepresentation of discontinuous rules affects the parsing coverage. To this end we generate two grammars, one of the sentences 0-46k of TRAIN

and the other of the DEV set. The second grammar represents the set of rules that are indispensable to correctly parse the small corpus. For both corpora used in the experiment we do all the preprocessing steps as with the corpora that we use for parsing. Then the percentage of rules of the second grammar missing in the first is calculated separately for continuous and discontinuous rules, once considering their frequncies in the DEV grammar and once not.

	Continuous	*Discontinuous*
$TRAIN$	26126	5426
DEV	3505	576
$Missing$	952 (27%)	224 (39%)
$Missing - Freq$	7%	25%

The row $Missing$ shows how many rules are missing. Then, $Missing - Freq$ counts each rule as many times as it appears in the second grammar. Note here the importance of these numbers, especially $Missing - Freq$, which tells us how many nodes (contituous or discontinuous) may not in principle be found, hence, an upper bound on the recall of corresponding nodes.

The table confirms our expectations. 39% discontinuous of rules are missing as opposed to 27% of missing continuous rules. The numbers improve with frequencies considered, as more frequent rules are more likely to appear. We notice also that this number drops quicker for continuous rules, suggesting that discontinuities have a more uniform distribution. The findings imply that the highest possible recall that we may have on discontinuous nodes is 75%, which is a bit discouraging.

This problem in turn will also affect the overall parse quality quite a lot, because discontinuous node's segments are usually located in the context independent of the node's context. If it is not recognized correctly, the entire parse is very likely to mix up completely. If we, for instance, miss a discontinuous rule which has two segments, then, ther will be two remote unrecognised nodes, deteriorating the parse from different positions, whereas a missing continuous rule only introduces errors in one place. It seems possible that the overall parse quality may to a larger degree depend on the ability to recognize discontinuities, in comparison to usual continuous rules. It is surprising that so far in the literature there have been no node-wise report on that matter. We are going to give it in the following sections.

Among the discontinuous missing rules the most frequent are VPs with separated prepositional or adverbial modifiers and topicalized NPs. There

are overall many classes with rather consistent patterns that one can distinguish. The following table contains several examples of rules that we found.

	Rule
VP-PP	$VP(X_0X_1, X_2) \to NP(X_0)VVINF(X_1)PP1(X_2)$
VP-ADV	$VP(X_0, X_1X_2X_3) \to ADV(X_0)NP(X_1)PP(X_2)VVINF(X_3)$
VP-NP	$VP(X_0, X_1X_2X_3) \to NP(X_0)PROAV(X_1)NP(X_2)VVPP(X_3)$
NP-S	$NP(X_0X_1X_2, X_3) \to PDAT(X0)AP1(X1)NN(X2)S1(X3)$

Another prominent class presented there is NPs with extraposed relative clauses. These discontinuities are very easily recognized, they have an NP-like combination of categories and a separated S, whose function in the corpus is labeled by RC (relative clause). There are rather few extraposed relative clauses also for PPs and VPs.

The conflict arising with the appearence of discontinuous rules is that, there are too many possibilities to create a discontinuity and too few are actually presented in the corpus. If, for instance, there are x possibilities to form a continuous VP, how many discontinuous VP rules we may need? Assuming a VP can be intermixed with the subject and possibly another verb, which, with the average length of the RHS of 4, can take at least 5-6 different positions. Then the number of required rules becomes $5x$. But, as we know discontinuous nodes comprise only 5% of all nodes and, therefore the discontinuities have so to say $5 \times 20 = 100$ times less training data.

Searching for a possible way to recover the missing rules we introduce a notion of a similar rule. Given a rule

$$R = A(\alpha_1, \ldots, \alpha_{dim(A)}) \to A_1(X_1^1, \ldots, X_{dim(A_1)}^1) \ldots A_m(X_1^m, \ldots, X_{dim(A_m)}^m)$$

we form the word $w = \alpha_1\alpha_2 \ldots \alpha_{dim(A)} \in (T \cup V)^*$. Note that we only have such rules when α_i belongs to V^*, thus $w \in V^*$, and $w = X_1X_2 \ldots X_k$ is a permutation of all variables of the RHS. Recall that a nonterminal may only have one value of the dim function. Therefore A has actually in our notation the name AX, where $X = dim(A)$. All divisions of the vector w into subvectors

$$S = X_1 \ldots X_{n_1}, X_{n_1+1} \ldots X_{n_2}, \ldots, X_{n_{l-1}+1} \ldots X_{n_l=k}$$

there is a rule

$$AY(S) \to A_1(X_1^1, \ldots, X_{dim(A_1)}^1) \ldots A_m(X_1^m, \ldots, X_{dim(A_m)}^m)$$

with the same RHS as R, the same ordering of the variables but different restrictions of the distance between them, direct precedence may be interchanged with just precedence. Here, Y is the length of the sequence S. The set of all these rules form the class of similar rules, which are also similar to R. Consider the rules $NP(X_0X_1, X_2) \rightarrow PPOSAT(X_0)NN(X_1)CS(X_2)$ and $NP(X_0X_1X_2) \rightarrow PPOSAT(X_0)NN(X_1)CS(X_2)$, dropping *dim* values in the labels. They are similar because they only differ in the number and positions of the commas in the LSH. TIGER lacks the first rule but has the second one.

We discover that among all 224 missing discontinuous rules 85 (38%) have similar in the grammar. By recovering those we may potentially greatly increase the theoretical recall upper-bound on the discontinuous nodes. Unfortunately adding all similar rules of those present in the grammar is both linguistically inadequte, will cause overgeneration, and technically implausible due to significant slowdown of parsing speed. We have to choose carefully only those similar rules that are empirically supported and do not overload parsing.

Extraposition in German is very frequent. Extraposed relative clauses are one of the most frequent instanses of the usage of discontinuous rules as we will see in Section 4.4. There are many NP (a few PP and VP) variations which accept extraposed relative clauses, that is why there are missing rules for the phenomena. The range of such NPs is similar (or identical) to that of NPs with adjacent relative clauses. This, together with the fact that relative clauses occur chiefly at the end of an NP allows us to add to the grammar a small number of rules that are responsible for extraposition which should be there but are absent because of low frequency and lack of training data. All in all, we add 434 rules to NEGRA, among them 262 rules for NPs, and to TIGER we add 780 rules and 422 rules respectively.

However, direct addition of new rules to the grammar is only one way to solve the problem, which can be applied to other phenomena, but we observed that it is not as fruitful as the next approach. It in some sense adds all possible missing rules for extraposed relative clauses. Given a node in a tree, i.e. obtained by the application of the rule $NP(X_0X_1X_2, X_3) \rightarrow PDAT(X0)AP1(X1)NN(X2)S1(X3)$, we create an intermediate NP as in $NP(X_0X_1X_2, X_3) \rightarrow PDAT(X0)AP1(X1)NN(X2)$ and attach the relative clause later with $NP(X_1, X_2) \rightarrow NP(X_1)S(X_2)$. There is only one discontinuous rule to cover the phenomenon, thus, nothing is missing. Additionally, this way we do not need to assign any ad hoc probabilities to the new rules and interfere with the maximum likelihood estimates. A relative comparison of parsing results with and without the modification is given in

Section 4.5.5.

The method described may be used jointly with node split. We divide all S and CS nodes into relative clauses SRC and CSRC and just sentences according to their function. We try node split separately and together with the tree modifications for extraposition.

4.4 Effeciency

In this section we focus on the effeciency of parsing with LCFRS. The generalization over CFG made by LCFRS is that discontinuous constituents are allowed. CFG is parsed very quickly, much faster than LCFRS. The source of this inefficiency is obviously the discontinuous rules. The size of a gap is not limited by anything. Thus in the chart and the agenda there will always be much more items. If, for instance, in the CFG parse of a sentence of lenght k all possible item form the set of the size $\frac{k(k+1)}{2}$, for LCFRS with the maximal fan-out two, the same number is already $O(k^4)$.

We concentrace on disconinuous rules of the generated grammars. We collect certain statistics about the training corpus. First, the most frequent rules are found and their meaning established. Second, all gaps occuring in the corpus in the gap positions of these rules are collected. We believe that using the information about the rules' essence and the gaps one can significantly improve on effeciency and accuracy. A number of steps can be made. One can integrate information about the gaps in the rule, so that it will only apply if the gaps are filled with certain categories or have certain lengths. At the same time one can decide that a few discontinuities can be resolved from the beginning, as with preprocessing for CFG but less radically. Both ways lead to inproved parsing speed.

Here, we list some of the most frequent rules in TIGER with their frequencies.

	Rule	*Frequency*
1	$NP(X_0X_1, X_2) \rightarrow ART(X_0)NN(X_1)S(X_2)$	376
2	$PP(X_0X_1X_2, X_3) \rightarrow APPR(X_0)ART(X_1)NN(X_2)S(X_3)$	186
3	$VP(X_0, X_1) \rightarrow PP(X_0)VVPP(X_1)$	609
4	$VP(X_0, X_1X_2) \rightarrow VP(X_0, X_1)VAINF(X_2)$	350
5	$VP(X_0, X_1X_2) \rightarrow VP(X_0, X_1)VAPP(X_2)$	223
6	$PP(X_0, X_1) \rightarrow PROAV(X_0)VP(X_1)$	186
7	$NP(X_0, X_1) \rightarrow PPER(X_0)VP(X_1)$	355

Most of the rules are easily interpreted. Consider the first two rules.

They represent the most frequent NP and PP patterns, $ART + NN$ and $APPR{+}ART{+}NN$, followed by an extraposed relative clause. For instance, it can be extracted from the following pair of an extract and a constinuent of this extract:

(5) *... ihnen auch die Rechte geben die mit diesem Mandat verbunden sind.*
die Rechte die mit diesem Mandat verbunden sind

The gaps may have 66 different variations (for the NP rule), which do not suggest any particular pattern. This finding confirms that extraposition is an unbound dependency.

The third rule is created because one of the PP modifiers of a participle is located far from the verb. In the corpus the finite verb and the subject of a sentence are located above the VP with the infinite verb, its complements and modifiers. As infinite verbs are placed at the end, a gap is formed if a modifier appears before the subject or the finite verb.

(6) *Auch in Schiras wurden vier Gottesfeinde gehenkt.*
Auch in Schiras gehenkt

Consider also the tree in Figure 9. The rule

$$VP(X_1, X_2) \rightarrow PROAV(X_1)\ VVPP(X_2)$$

is created because the finite verb $VMFIN$ is attached higher. In the same tree we observe application of the rule 4 of the table. It serves to pass the gap higher in the tree, because the subject and the main verb come even higher.

If one looks at this rule more closely, one may observe that it can be though of as a usual context-free rule, as two adjacent constituents are concatenated. The rule resembles $VP(X_0X_1) \rightarrow VP(X_0)VAINF(X_1)$, though, they are not similar in the sense that we defined. It would be also interesting to observe how many similar rules in this sense are in the grammar and whether we may profit from using this similarity. We leave this question to the future work.

The rules like 3, 4 and 5 have more consitent gap patterns than rules 1 and 2. For instance, in Figure 12 one can find all gap patterns with frequincies for rule 5. There, under the item *Other* we put all other than presented patterns, which are actually very alike but with different variation on the NP.

$VAFIN + NP$	109
NP	37
$VAFIN$	19
$VAFIN + NN$	6
$VAFIN + NE$	6
$PPER$	4
$VAFIN + PPER$	3
NN	3
$Other$	36

Figure 12: Gap patterns with frequencies

Although there are still many of them, their overall appearence in most cases can be described as $VAFIN + NP$ or with one of the components missing. Therefore, in theory there should be a method to use this information to help parsing, either by re-arranging the tree or by introducing information about gaps in the grammar rules. We do not have a direct and precise solution, but we want to point out that the formalism loses potentially very useful information. Moreover, the previously described type of rules that pass gaps also suggests that the gap in the LHS and that in the RHS should be equated, so that, the rule becomes a CFG rule.

Rule 7 is used for explitive *es* and the following is the typical construction in German that uses rule 6:

(7) *... müßte damit rechnen festgenommen zu werden.*
damit festgenommen zu werden

4.5 Parsing

4.5.1 Parser

We implement the parsing algorithm as described in section 3.2.5. The code is written in Java and all the experiments are conducted on an Intel i5-2520M CPU 2.50Ghz processor.

After binarization all rules as hashed by two tables and can be quickly found by their left or right nonterminal in the RHS. The agenda is implemented as a priority queue. It allows for logarithmic best element extraction time as well as logarithmic new element insertion time. Unfortunately, we

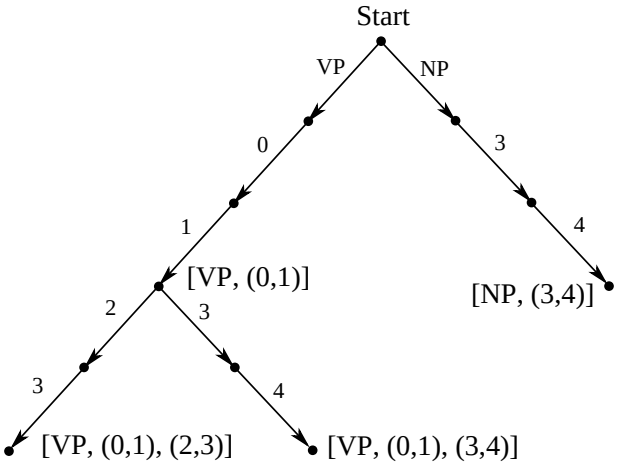

Figure 13: A tree from the TIGER corpus

constantly need to update the weights, which involves inserting new elements as well as removing old ones. Removal may take linear time because of possible rearrangements of the structure. We decide to allow duplicate items with different weights in the agenda and then quickly pass them if they are already in the chart. The insertion and extraction of the best element remain fast. The agenda does not grow too quickly because of this as well.

The chart is designed to be a decision tree. Every item has coordinates: the labels, left and right ends of the continuous segments. For example, the item $[VP, (0, 1), (2, 3)]$ may be encoded as the vector $(VP, 0, 1, 2, 3)$. In this representation all possible items can be thought of as vectors in a vector space, only not all vectors have the same length and may have string as well as numeral values. We locate items in a decision tree, a leveled tree with edges going from level k to level $k + 1$ are labeled with possible values of the k coordinate. For instance for the items $[VP, (0, 1), (2, 3)]$, $[VP, (0, 1)]$, $[NP, (3, 4)]$, and $[VP, (0, 1), (2, 4)]$ the tree from Figure 13 will be created. Three nodes there contain the corresponding items, the rest are empty.

A new item is inserted in the chart in the position determined by its coordinates. With this representation it is possible to search for items with unknown structure without iterating over the chart. For instance, when parsing we may extract the item $[A, (0, 1), (3, 4)]$ from the agenda and want

	TIGER	TIGER-CFG	NEGRA	NEGRA-CFG
Parsed	1337	1356	1355	1356
Complete match	434	450	402	387
Recall	0.768	0.780	0.768	0.779
Precision	0.784	0.781	0.762	0.769
$F - score$	0.765	0.776	0.761	0.769
Parsing speed	5622	1797	1897	868
Avg. chartsize	90696	22540	54437	14531

Figure 14: Statistics on all sentences

to apply the rule

$$C(X_1X_2, X_3X_4) \to A(X_1, X_3)\ B(X_2, X_4)$$

to it. Then, we need to find any item in the chart with the label B, first coordinate $X_2 = (1, x)$ where $1 < x \leq 3$, and $X_4 = (4, y)$ with $4 < y$. This search query can easily be executed by our decision tree.

4.5.2 First experiments

In this section we present the results of our parsing experiments with unmodified and non-markovized grammars. The following tables shows the sentence level performance (as opposed to node-wise performance) of CFG and LCFRS parsers separately for sentences with continuities and without as well as for all sentences.

The results are given for all sentences (1357 all, 997 continuous and 360 discontinuous). For those that were not parsed we add 1 to precision and 0 to recall. Most of the parsing failures time-outs (1 minute), that we put. The parsing speed in the tables is given in ms/sentence. In the CFG case by discontinuous sentences we mean those that were discontinuous before the crossing branches were resolved.

The first basic observation that we make is that CFG and LCFRS produce parses of almost the same quality. The quality of CFG parising is a little bit better for all kinds of data. As we observed in Section 4.4 many of the most important gaps have consistent patters. Thus, after the resolution

	TIGER	TIGER-CFG	NEGRA	NEGRA-CFG
Parsed	987	996	995	996
Complete match	376	360	349	339
Recall	0.801	0.810	0.801	0.809
Precision	0.811	0.808	0.793	0.796
$F - score$	0.791	0.804	0.792	0.797
Parsing speed	4210	1472	1437	711
Avg. chartsize	68020	18287	41406	11867

Figure 15: Statistics on continuous sentences

	TIGER	TIGER-CFG	NEGRA	NEGRA-CFG
Parsed	350	360	360	360
Complete match	58	50	53	48
Recall	0.674	0.694	0.676	0.694
Precision	0.708	0.706	0.676	0.694
$F - score$	0.675	0.697	0.674	0.692
Avg. speed	9528	2694	3169	1303
Avg. chartsize	153495	34322	90528	21908

Figure 16: Statistics on discontinuous sentences

$GOLD \backslash PARSE$	0 *gaps*	1 *gap*	2 *gaps*	3 *gaps*
0 *gaps*	910	72	5	0
1 *gap*	94	196	10	0
2 *gaps*	7	18	18	2
3 *gaps*	0	0	2	3

Figure 17: Corelation of the numbers of gaps betweet the gold standard and the parsing results

we obtain quite regular combinations. Raising nodes within VPs with verbal complexes should not cause irregularities. However raising extraposed relative clauses seems to be more random while they can then attach to a very wide range of phrases. The same can be said about the expletive 'es' under $PPER$ in the rule $NP(X_0, X_1) \rightarrow PPER(X_0)VP(X_1)$ for which there are also no regular gaps.

With discontinuous sentences both parsers seem to have significantly more trouble than with continuous ones. This is, of course, not surprising, because discontinuous rules are less frequent and therefore the LCFRS parser will first try to form more probable continuous ones.

The parsing speed of LCFRS is approximately 3 times slower that that of CFG. The chart also has on average 3-4 times more items. Although the parsing accuracy does not increase much after switching from NEGRA to TIGER, significantly more exact matches are obtained. Thus far, the parser does not seem to cope well with discontinuities, neither does it maintain the effeciency of CFG. One may ask whether the parser is at all capable of catching discontinuities, or maybe they can in many cases be substituted by more frequent continuous attachments. The Figure 17 shows that as we parse the numbers of existing discontinuities and those found correlate surprisingly well (even acounting for possible random concidence).

We see that it is not the case that discontinuities are never found. Among 297 sentences with one discontinuous node we miss only 104. Knowing from Section 4.3 that we would in any case miss approximately 25% of rules, the results are encouraging. The number of wrongly created discontinuous constituents is comparable to that of missing ones, therefore we have a balance between overgeneration and undergeneration.

All in all, we may conclude that the probabilistic model that we use is appropriate for the task, and the bad quality is cause by:

$Label$	$f-measure$	$precision$	$recall$	$goldFreq$	$parseFreq$
AVP	0.389	0.468	0.333	111	79
VP	0.687	0.666	0.709	626	667
CNP	0.692	0.698	0.687	252	248
NP	0.724	0.722	0.725	2407	2419
PN	0.739	0.800	0.687	262	225
PP	0.764	0.764	0.766	1988	1993
S	0.928	0.921	0.936	1833	1862
AP	0.570	0.631	0.521	315	260
VZ	0.991	0.982	1.000	109	111

Figure 18: Parsing results on continuous nodes

- mainly, very low coverage of discontinuous rules, and probably
- insufficient training on discontinuous data, thus, bad probabilty estimation and usage of wrong rules.

The latter can be approached by smooting of the probabilities, but we are more ambitious about solving the first problem as was described in Section 4.3.

4.5.3 Going node-wise

We also evaluate parsing results for each node label separately. This evaluation should reveal the categories causing most problems as well as present to some extent independently the discontinuous part of the parses. In Figure 18 we have the most frequent category labels of continuous nodes and the parse quality on them. The numbers are calculated for parsed sentences. The training set is TIGER.

The good quality on the S nodes is partially due to our decision to parse only sentences rootes in S. This may also be the reason why our results are several points higher than that of the similar systems refered to in Section 2.3.3. The most important and frequent categories NP, VP, and PP give approximately 72%, 69%, and 76%. Now let us turn to the discontinuities in Figure 19 (only most frequent categories).

$Label$	$f-measure$	$precision$	$recall$	$goldFreq$	$parseFreq$
PP	0.507	0.486	0.529	34	37
VP	0.429	0.467	0.397	290	246
NP	0.235	0.231	0.239	88	91

Figure 19: Parsing results on discontinuous nodes

	$v=1$	$v=2$	$v=3$
$h=1$	0.755	0.780	0.769
$h=2$	0.769	0.774	0.746
$h=3$	0.768	0.770	0.740
$h=\infty$	0.765	0.769	0.740

Figure 20: Different markovization parameters

These numbers are significantly lower. However, what we again observe is that even for less frequent, and not presented in this table, categories there is a very strong correlation between the numbers of rules found and expected. We are coming to the same conclusion as in the end of Section 4.5.2. We can also see the low recall on VPs, which agrees with the finding that very many of the missing rules are those that construct discontinuous VPs.

4.5.4 Markovization

We try different markovization parameters, adding vertical and horizontal contexts to the nodes as described in Section 3.2.3. The Figure 20 shows all different variations that we tried on the TIGER training set.

Unlike the previously reported results and results for English, markovization in our case does not add more than 2% to the initial results. There are various reasons. First, the trees are rather plain and the vertical contex is not as informative now. Second, as we excluded the $VROOT$ nodes, there is one level less. As with the growth of the vertical and horizontal context the results only deteriorate after $v=2$ and $h=1$, we quickly reach the maximal acceptable level of sparseness. Note, also, that the results for NEGRA show the same relative changes. Importantly, or best result $v=2$, $h=1$ requires very much the horizontal context, the result for $v=2$, $h=\infty$ are more than

$Label$	$f-measure$	$precision$	$recall$	$goldFreq$	$parseFreq$
AVP	0.137	0.240	0.097	125	50
VP	0.388	0.459	0.336	1543	1131
NP	0.279	0.338	0.238	408	287
PP	0.480	0.591	0.405	185	127
AP	0.095	0.155	0.068	132	58

Figure 21: Discontinuous nodes, no markovization

1% lower.

The vertical context provides a significant effeciency improvement increasing the speed 3 times with $v = 2$, and by 9 with $v = 3$. It is crucial for parsing with TIGER, as we are already approaching our computational limits, and becomes even more important with the addition of more training data or more rules, which is indicated by our findings as a necessity. Horizontal context, though, slows down parsing and, therefore, we could not parse all the sentences with $h = 0$, $v = 2$ and $h = 0$, $v = 3$ within our time limit. But the results on parsed sentences show alike numbers.

Compared to general results, the results restricted on discontinuous nodes, Figures 21 and 22, are improved greatly. Here we decide to used an extended test set of all discontinuous sentences (all 1870 of them) found in 10k sentences. We parse with the grammar extracted from NEGRA, as the entire TIGER corpus is not availiable (10k sentences are used for testing), the accuracy results do not differ much, and there are less sentences without a parse (the grammar is smaller). The markovization parameters are chosen to be $h = 1$ and $v = 2$. The evaluation is done on parsed sentences, thus, the frequencies may slightly differ between the two tables.

4.5.5 Re-attachment of relative clauses

Here we present the effects of adding new rules to better recognize extraposed relative caluses. The training and test sets remain those from the previous chapter. We try two approaches: differentiate sentences with relative clauses from those without, and reconstruct the trees with relative clauses the way we described in Section 4.3. The average parsing speed does not change much. The improved results are given in Figure 23. It is also worth noticing

$Label$	$f-measure$	$precision$	$recall$	$goldFreq$	$parseFreq$
AVP	0.309	0.491	0.226	124	57
VP	0.418	0.506	0.357	1535	1085
NP	0.327	0.380	0.287	407	308
PP	0.533	0.583	0.492	185	156
AP	0.230	0.396	0.161	130	53

Figure 22: Discontinuous nodes after markovization

$Label$	$f-measure$	$precision$	$recall$	$goldFreq$	$parseFreq$
AVP	0.308	0.483	0.226	124	58
VP	0.419	0.507	0.358	1541	1086
NP	0.349	0.423	0.297	408	286
PP	0.543	0.587	0.505	186	160
AP	0.227	0.396	0.159	132	53

Figure 23: Results with after re-attaching relative clauses

that the number of sentences which did not receive a parse after exhaustive parsing dropped from 8 to 5.

Now with node splitting, the results are even better (Figure 24). Some relative improvement after the splitting was already reported in [40]. Here, we see that the two methods can naturally suplement each other.

We conclude, that a significant parsing accuracy improvement can be achieved by an external automatic corpus modification.

$Label$	$f-measure$	$precision$	$recall$	$goldFreq$	$parseFreq$
AVP	0.281	0.461	0.202	124	54
VP	0.421	0.512	0.358	1539	1076
NP	0.366	0.440	0.314	408	291
PP	0.555	0.579	0.532	186	171
AP	0.242	0.440	0.167	132	50

Figure 24: Results with reconstructed trees and split S category

5 Conclusion

In this thesis we touched upon different properties of such formalisms as HPSG, LCFRS, and ACG. On the one side we have a more linguistically aware HPSG, whose mechanisms allow to process complex language phenomena. It produces very accurate parses but the coverage requires a lot of effort to improve. At the same time, it was only recently that HPSG was subjected to modification allowing for parsing dicontinuities in a direct way, which, however, results in a significant slow-down.

At the same time discontinuities are essential in natural languages, especially for so-called relatively free word order languages like German. The proportion of sentences with discontinuities in German is close to 30%. Thus, it is not very surprising the German grammars in the HPSG formalism face great difficulties.

There is a class of mildely context-sensitive formalisms that are considerd as appropriate for modeling natural languages. Among them we find LCFRS and ACG, which in fact were proved to be weakly equivalent. These formalisms handle discontinuities with realtive ease, but lack means to express other linguistic generalizations. It seems possible that establishing strict theortical connection between HPSG and ACG is profitable for both formalisms including LCFRS.

We made the first step towards revealing these connections. We came with certain conditions for a grammar in HPSG formalism, under which it may be represented as a hyperedge replacement grammar. And those in turn can be imitated as ACGs. Though, the conditions strongly restrict the grammar's ability to use sharing. It is of course related to the fact that ACG is resource-sensitive and sharing is the same as duplication. The remaining

'tree skeletons' of the feature structures remain practically untouched. The general picture is that HPSG can be converted into ACG if all its type definitions are 'like' LCFRS ruls, they only pass substructures (or variables in LCFRS) up in the derivation and choose their direct substructures (daughters). Moreover, certain type of shring is additionally allowed between these substructures. This may be thought as a mutual restriction on variables of an LCFRS rule. In some sense we already use such restrictions to say that the variables of a continuous rule must be adjacent segments in the string and those of a discontinuous rule may not be adjacent. We admit that our condition can possibly be loosened and hopefully, after a more detailed consideration it will be possible to distinguish exactly which features of HPSG machinery may be modeled in the formalism based on a resource-sensitive logic.

The condition we state may by fulfilled on the subpart of HPSG devoted to semantics, for instance the Mininal Recursion Semantics (MRS) for HPSG, hence MSR can be introduced as a component of LCFRS. For the syntax, whether the syntactic part can be modified or simplified to fulfill the condition is not trivial to answer. Sharing plays an important role there and is restrictive with respect to further derivation, thus, neither it is resource sensitive, nor are their derivations context-free. As future plans we consider implementing a convertion of a German HPSG grammar into an ACG or LCFRS, or at least its maximally possibly subparts.

In the second, practical part of this research we looked deeper into the intricacies of the LCFRS implementation. The recently developed methods of reading a grammar facilitate research in this area with automatically extracted large scale grammars. The TIGER corpus is a treebank of German with crossing branches. It is a great platform for experiments with LCFRS. We implemented an LCFRS parser with the aim of investigating the flaws of the formalism and its strength. We present the first, as to our knowledge, analisys of the node-wise performance of the parser and directly compare it to that of the CFG parser, which is a restriction of our LCFRS parser to continuous items. We pay most attention to discontinuous structures, as only they differenciate LCFRS from CFG. The result of our work consists of the analyses of the parsing results and the extracted grammars together with a description of actual and potential problems and suggestions about their solutions which are also supported by out theoretical results. We conduct a few experiments to improve extraposed relative clauses recognition to confirm the positive effect of grammar enrichments.

We investigate the most frequent rules on the different kinds of gaps they can accept. It turns out that in many cases discontinuous rules have rather

consistent gap patterns. This property suggests that one may inform the rules or the parser so that the gaps, which are now arbitrary, also convey restrictive information.

Our implementation achieves state-of-the-art accuracy results and even better, because of various simplifications that we make. These simplifications are introduced in order to make the work of the parser less dependent on the annotation and issues which are not adressed yet by current grammar extraction mechanisms.

The experiments obviously point out at the underrepresintation of discontinuities in the corpus. And the problem is of course not in the corpus but rather in the idea of reading grammars from corpora. All possible discontinuous rules constitute a very large class of rules. At the same time discontinuous nodes are 20 times rarer than continuous ones. The basic observation is that a great deal of discontinuous rules are already present in the grammar as continuous rules but with one or more gaps. It implies several things. First, the number of discontinuous rules may potentially grow over the number of continuous rules. Second, the grammars extracted even from the smaller corpora like NEGRA already contain a lot of missing rules, though, without gaps.

We try many combinations of markovization parameters and report parsing result. There have been a thorough report on the influence of markovization parameters on CFG parsing of German with grammars extracted from the TIGER corpus, but they cannot be applied to LCFRS parsing without neglecting discontinuities. These findings also confirm the hypothesis that the grammars lack discontinuous rules. They show that adding horizontal context is vital to the parsing results, especially restricted to discontinuous nodes. And horizontal context can be thought as adding, in some sense, similar rules generalizing the given rules.

This corresponds to what we do with extraposed relative clauses. Namely, we reconstruct the trees in such a way that relative clauses, adjacent or not, are attached after other daughters are combined into a phrase, NP or PP. By doing so, we use the fact that relative clauses are modifiers. For instance, an NP with modification or not, it still remains as an NP. Moreover, if the relative clause can be adjacent, then in principle it can also me separated. Our results give significant improvement on the discontinuous NP and PP nodes. Since other most frequient missing discontinuous rules represent ADV and PP modification, we find in our soonest plans an extension of our method to modification in general. Another possible solution that we plan to try in the future consists in selecting which discontinuities should be left and which should be resolved the same way as we do for CFG parsing.

References

[1] Michel Bauderon and Bruno Courcelle. Graph expressions and graph rewritings. *Mathematical Systems Theory*, 20(2-3):83–127, 1987.

[2] Philip Bille. A survey on tree edit distance and related problems. *Theor. Comput. Sci.*, 337(1-3):217–239, June 2005.

[3] Sabine Brants, Stefanie Dipper, Silvia Hansen, Wolfgang Lezius, and George Smith. The TIGER treebank. In *Proceedings of the workshop on treebanks and linguistic theories*, pages 24–41, 2002.

[4] Håkan Burden and Peter Ljunglöf. Parsing linear context-free rewriting systems. In *Proceedings of the Ninth International Workshop on Parsing Technology*, Parsing '05, pages 11–17, Stroudsburg, PA, USA, 2005. Association for Computational Linguistics.

[5] Bob Carpenter. *The Logic of Typed Feature Structures*. Cambridge University Press, Cambridge, 1992.

[6] Michael Collins. Three generative, lexicalised models for statistical parsing. In *Proceedings of the eighth conference on European chapter of the Association for Computational Linguistics*, EACL '97, pages 16–23, Stroudsburg, PA, USA, 1997. Association for Computational Linguistics.

[7] Michael Collins. Head-driven statistical models for natural language parsing. *Comput. Linguist.*, 29(4):589–637, December 2003.

[8] Ann Copestake and Dan Flickinger. An open source grammar development environment and broad-coverage english grammar using hpsg. In *IN PROCEEDINGS OF LREC 2000*, pages 591–600, 2000.

[9] Berthold Crysmann. On the efficient implementation of German verb placement in HPSG. In *Proceedings of RANLP 2003*, pages 112–116, Borovets, Bulgaria, 2003.

[10] Berthold Crysmann. Relative clause extraposition in german: An efficient and portable implementation. *Research on Language and Computation*, 3:61–82, 2005.

[11] Micheal W. Daniels. *Generalized ID/LP grammar: a formalism for parsing linearization-based HPSG grammars*. PhD thesis, The Ohio State University, 2005.

[12] Philippe de Groote. Towards abstract categorial grammars. In *Association for Computational Linguistics, 39th Annual Meeting and 10th Conference of the European Chapter, Proceedings of the Conference*, pages 148–155, 2001.

[13] Philippe de Groote. Tree-adjoining grammars as abstract categorial grammars. In *TAG+6, Proceedings of the sixth International Workshop on Tree Adjoining Grammars and Related Frameworks*, pages 145–150. Università di Venezia, 2002.

[14] Philippe De Groote and Sarah Maarek. Type-theoretic extensions of Abstract Categorial Grammars. In R. Muskens, editor, *Proceedings of Workshop on New Directions in Type-theoretic Grammars*, pages 19–30, 2007.

[15] Philippe de Groote and Sylvain Pogodalla. m-linear context-free rewriting systems as abstract categorial grammars. In R. T. Oehrle et J. Rogers, editor, *Proceedings of Mathematics of Language - MOL-8, Bloomington, Indiana, Etats-Unis*, pages 71–80, Jun 2003.

[16] Philippe De Groote and Sylvain Pogodalla. On the expressive power of Abstract Categorial Grammars: Representing context-free formalisms. *Journal of Logic, Language and Information*, 13(4):421–438, 2004.

[17] Stefanie Dipper. Grammar-based corpus annotation. In Anne Abeille, Thorsten Brants, and Hans Uszkoreit, editors, *Proceedings of the Workshop on Linguistically Interpreted Corpora LINC-2000 , Luxembourg*, pages 56–64, 2000.

[18] Amit Dubey and Frank Keller. Probabilistic parsing for german using sister-head dependencies. In *IN PROCEEDINGS OF THE 41ST ANNUAL MEETING OF THE ASSOCIATION FOR COMPUTATIONAL LINGUISTICS*, pages 96–103, 2003.

[19] Jay Earley. An efficient context-free parsing algorithm. *Commun. ACM*, 26(1):57–61, January 1983.

[20] Daniel Gildea. Optimal parsing strategies for linear context-free rewriting systems. In *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, HLT '10, pages 769–776, Stroudsburg, PA, USA, 2010. Association for Computational Linguistics.

[21] J.-Y. Girard. Linear logic. *Theoretical Computer Science*, 50:1–102, 1987.

[22] Carlos Gomez-rodriguez, Marco Kuhlmann, Giorgio Satta, and David Weir. Optimal reduction of rule length in linear context-free rewriting systems. In *In Proc. of NAACL 09:HLT*, 2009.

[23] Thilo Götz and Walt Detmar Meurers. Compiling hpsg type constraints into definite clause programs. In *Proceedings of the 33rd annual meeting on Association for Computational Linguistics*, ACL '95, pages 85–91, Stroudsburg, PA, USA, 1995. Association for Computational Linguistics.

[24] Johan Hall and Joakim Nivre. A dependency-driven parser for German dependency and constituency representations. In *Proceedings of the Workshop on Parsing German*, pages 47–54, Columbus, Ohio, June 2008. Association for Computational Linguistics.

[25] Julia Hockenmaier. Creating a ccgbank and a wide-coverage ccg lexicon for german. In *In Proc. of the 44th Annual Meeting of the ACL and 21st International Conference on Computational Linguistics (COLING/ACL-2006*, pages 505–512, 2006.

[26] Tilmann N. H"ohle. Der Begriff 'Mittelfeld'. Anmerkungen "uber die Theorie der topologischen Felder. In *Akten des Vii.Internationalen Germanisten-Kongresses G"ottingen 1985*, volume 3, pages 329–340. Niemeyer, T"ubingen, 1986.

[27] Yu-Yin Hsu. Comparing conversions of discontinuity in PCFG parsing. Indiana University, 2009.

[28] Josef van Genabith Ines Rehbein. Evaluating evaluation measures. In *Proceedings of the 16th Nordic Conference of Computational Linguistics NODALIDA-2007*, pages 372–379, Tartu, 2007. University of Tartu.

[29] Laura Kallmeyer and Wolfgang Maier. Data-driven parsing with probabilistic linear context-free rewriting systems. In *COLING*, pages 537–545, 2010.

[30] Makoto Kanazawa. Second-order abstract categorial grammars as hyperedge replacement grammars, 2007.

[31] Andreas Kathol. *Linearization-Based German Syntax*. PhD thesis, Ohio State University, 1995.

[32] Stephan Kepser and Uwe Mönnich. (Un-)Decidability results for head-driven phrase structure grammar. In Giuseppe Scollo and Anton Nijholt, editors, *Proceedings of Algebraic Methods in Language Processing (AMiLP-3)*, pages 141–152, 2003.

[33] Bernd Kiefer, Hans-Ulrich Krieger, and Melanie Siegel. An HPSG-to-CFG approximation of Japanese. In *In Proceedings of Coling 2000*, 2000.

[34] Paul John King. A Logical Formalism for Head-Driven Phrase Structure Grammar. University of Manchester, 1989.

[35] Dan Klein and Christopher D. Manning. Accurate unlexicalized parsing. In *Proceedings of the 41st Annual Meeting on Association for Computational Linguistics - Volume 1*, ACL '03, pages 423–430, Stroudsburg, PA, USA, 2003. Association for Computational Linguistics.

[36] Sandra Kubler, Jelena Prokic, and Rijksuniversiteit Groningen. Why is german dependency parsing more reliable than constituent parsing. In *In Proceedings of the Fifth Workshop on Treebanks and Linguistic Theories (TLT*, pages 7–18, 2006.

[37] Sandra Kübler and Heike Telljohann. Towards a dependency-oriented evaluation for partial parsing. In *Proceedings of Beyond PARSEVAL – Towards Improved Evaluation Measures for Parsing Systems (LREC 2002 Workshop)*, 2008.

[38] Sandra Kubler and Universitat Tubingen. How do treebank annotation schemes influence parsing results? or how not to compare apples and oranges. In *In RANLP*, pages 293–300, 2005.

[39] Wolfgang Maier. Direct parsing of discontinuous constituents in german. In *Proceedings of the NAACL HLT 2010 First Workshop on Statistical Parsing of Morphologically-Rich Languages*, pages 58–66, Los Angeles, CA, USA, June 2010. Association for Computational Linguistics.

[40] Wolfgang Maier and Laura Kallmeyer. Discontinuity and Non-Projectivity: Using Mildly Context-Sensitive Formalisms for Data-Driven Parsing - Errata. 2010.

[41] Wolfgang Maier and Anders Søgaard. Treebanks and mild context-sensitivity. In Philippe de Groote, editor, *Proceedings of the 13th Con-*

ference on Formal Grammar (FG-2008), pages 61–76, Hamburg, Germany, 2008. CSLI Publications.

[42] Mitchell Marcus, Grace Kim, Mary Ann Marcinkiewicz, Robert Macintyre, Ann Bies, Mark Ferguson, Karen Katz, and Britta Schasberger. The penn treebank: Annotating predicate argument structure. In *In ARPA Human Language Technology Workshop*, pages 114–119, 1994.

[43] Stefan Müller. *Deutsche Syntax deklarativ. Head-Driven Phrase Structure Grammar für das Deutsche*. Number 394 in Linguistische Arbeiten. Max Niemeyer Verlag, Tübingen, 1999.

[44] Stefan Müller. Parsing of an HPSG grammar for German: Word order domains and discontinuous constituents. In Jost Gippert and Peter Olivier, editors, *Multilinguale Corpora. Codierung, Strukturierung, Analyse. 11. Jahrestagung der Gesellschaft für Linguistische Datenverarbeitung*, pages 292–303, Prag, 1999. enigma corporation.

[45] Stefan Müller and Walter Kasper. HPSG analysis of German. In Wolfgang Wahlster, editor, *Verbmobil: Foundations of Speech-to-Speech Translation*, pages 238–253. Springer, Berlin, 2000.

[46] Reinhard Muskens. Separating syntax and combinatorics in categorial grammar. *Research on Language and Computation*, 5(3):267–285, 2007.

[47] Mark-Jan Nederhof. Weighted deductive parsing and knuth's algorithm. *Computational Linguistics*, 29(1):135–143, 2003.

[48] Slav Petrov and Dan Klein. Improved inference for unlexicalized parsing. In *Human Language Technologies 2007: The Conference of the North American Chapter of the Association for Computational Linguistics; Proceedings of the Main Conference*, pages 404–411, Rochester, New York, April 2007. Association for Computational Linguistics.

[49] Sylvain Pogodalla. Computing semantic representation: Towards ACG abstract terms as derivation trees. In *Proceedings of the Seventh International Workshop on Tree Adjoining Grammar and Related Formalisms (TAG+7)*, pages 64–71, May 2004.

[50] Carl Pollard. Covert movement in logical grammar. In *Logic and Grammar*, pages 17–40, 2011.

[51] Carl Pollard and Ivan A. Sag. *Head-Driven Phrase Structure Grammar*. The University of Chicago Press, Chicago, 1994.

[52] Carl J. Pollard. Strong generative capacity in HPSG. In Gert Webelhuth, Jean-Pierre Koenig, and Andreas Kathol, editors, *Lexical and Constructional Aspects of Linguistic Explanation*, pages 281–297. CSLI Publications, 1999.

[53] Anna N. Rafferty and Christopher D. Manning. Parsing three german treebanks: lexicalized and unlexicalized baselines. In *Proceedings of the Workshop on Parsing German*, PaGe '08, pages 40–46, Stroudsburg, PA, USA, 2008. Association for Computational Linguistics.

[54] M. Reape. A logical treatment of semi-free word order and bounded discontinuous constituency. In *Proc. of the 4th EACL*, pages 103–110, Manchester, UK, 1989.

[55] Ines Rehbein and Josef van Genabith. Automatic acquisition of lfg resources for german as good as it gets. In *Proceedings of LFG09: 13-16 July 2009, Cambridge, UK / 14th Lexical Functional Grammar Conference*, pages 480–500, Stanford, Ca, 2009. CSLI Publications. MP.

[56] Frank Richter. Closer to the Truth: A New Model Theory for HPSG. In James Rogers and Stephan Kepser, editors, *Model-Theoretic Syntax at 10. Proceedings of the ESSLLI'07 workshop MTS@10*, pages 99–108. Trinity College, Dublin, 2007.

[57] Christian Rohrer and Martin Forst. Improving coverage and parsing quality of a large-scale LFG for German. In *Proceedings of the Language Resources and Evaluation Conference (LREC-2006)*, Genoa, Italy, 2006.

[58] Sylvain Salvati. Encoding second order string ACG with Deterministic Tree Walking Transducers. In Shuly Wintner, editor, *The 11th conference on Formal Grammar*, FG Online Proceedings, pages 143–156, Malaga, Spain, 2007. CSLI Publications.

[59] Hiroyuki Seki, Takashi Matsumura, Mamoru Fujii, and Tadao Kasami. On multiple context-free grammars. *Theor. Comput. Sci.*, 88(2):191–229, 1991.

[60] Andreas van Cranenburgh. Efficient parsing with linear context-free rewriting systems. In *EACL*, pages 460–470, 2012.

[61] Jorn Veenstra, Frank Henrik Müller, and Tylman Ule. Topological field chunking for german. In *proceedings of the 6th conference on Natural*

language learning - Volume 20, COLING-02, pages 1–7, Stroudsburg, PA, USA, 2002. Association for Computational Linguistics.

[62] K. Vijay-Shanker, David J. Weir, and Aravind K. Joshi. Characterizing structural descriptions produced by various grammatical formalisms. In *In Proceedings of the 25th Annual Meeting of the Association for Computational Linguistics*, pages 104–111, 1987.

[63] David Weir. *Characterizing Mildly Context-Sensitive Grammar Formalisms*. PhD thesis, Department of Computer and Information Science, University of Pennsylvania, 1988. Available as Technical Report MS-CIS-88-74.

[64] Daniel H. Younger. Recognition and parsing of context-free languages in time n^3. *Information and Control*, 10(2):189–208, 1967.

[65] Yi Zhang and Hans-Ulrich Krieger. Large-scale corpus-driven pcfg approximation of an hpsg. In *Proceedings of the 12th International Conference on Parsing Technologies*, pages 198–208, Dublin, Ireland, October 2011. Association for Computational Linguistics.

[66] Yi Zhang, Rui Wang, and Stephan Oepen. Hybrid multilingual parsing with hpsg for srl. In *Proceedings of the Thirteenth Conference on Computational Natural Language Learning (CoNLL 2009): Shared Task*, pages 31–36, Boulder, Colorado, June 2009. Association for Computational Linguistics.