Technische Universität Dresden
Fakultät Informatik
Professur für Grundlagen der Programmierung

Sommersemester 2015

Diplomarbeit

Training of hybrid grammars for the generation of discontinuous phrase structures and non-projective dependency structures

Revised version: November 16, 2015

Kilian Gebhardt

Eingereicht am 20. August 2015
Bearbeitet vom 16. März 2015
bis zum 19. August 2015

Verantwortlicher Hochschullehrer:
Prof. Dr.-Ing. habil. Heiko Vogler

Abstract

Discontinuous constituent structures and non-projective dependency structures are common in natural language. In this work we consider hybrid trees and hybrid grammars to represent either kind of syntax structure. We formally describe and experimentally evaluate a method for learning a hybrid grammar consisting of a linear context-free rewriting system and a simple definite clause program from a corpus of constituent structures or a corpus of dependency structures.

Selbstständigkeitserklärung

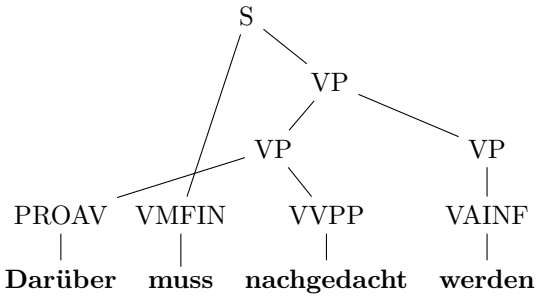
Hiermit versichere ich an Eides statt, die vorliegende Arbeit selbstständig und ohne fremde Hilfe angefertigt zu haben. Alle aus fremden Quellen direkt oder indirekt übernommenen Gedanken sind als solche gekennzeichnet. Die Arbeit wurde noch keiner Prüfungsbehörde in gleicher oder ähnlicher Form vorgelegt.

Dresden, 20. August 2015

Contents

1 Introduction

In linguistics and *natural language processing* (NLP) the syntactic structure of a sentence is often regarded as a tree. It is common in natural language that a sentence's *surface form*, i.e., the words of a sentence in the order they are spoken or written, can deviate from the *yield* of the sentence's syntax tree, i.e., the sequence of leaves taken from left to right [MS08]. As an example we consider the *constituent tree* below, which is taken from the German NEGRA treebank [Sku+98]. Its left-most leaf **muss** is the second word in the surface form.



Each constituent tree of this kind, i.e., each constituent tree whose surface form and yield mismatch, is called *discontinuous*.

For linguistic analysis and automated language processing it is desirable to find a finite formal model that describes every plausible syntax tree of a natural language. Hence, such a model should be capable of representing discontinuous trees. At the same time, the practical use in NLP demands that the model allows for efficient *parsing*, i.e., the generation of a syntax tree given some surface form shall cause low computational costs.

In this work we explore syntax models based on *hybrid trees* and *hybrid grammars* [NV14]. In a hybrid tree the representation of the syntactic structure is separated from the representation of the surface form. This is achieved by describing the former as a tree and the latter as a string where every symbol in the string is injectively linked with a node of the tree. By means of this double perspective discontinuous structures can be represented. A hybrid grammar is a finite device that generates hybrid trees. Given a corpus of constituent structures such as the NEGRA treebank, a hybrid grammar that generalizes this corpus can be learned automatically [NV14]. Our extension of [NV14] is twofold: firstly, we describe in more detail how a hybrid grammar can be learned from a corpus of constituent structures. Secondly, we transfer this learning method to an alternative view on syntax called *dependency structures*. The following outlines how this thesis is organized.

- Machine learning tasks with natural language require domain specific knowledge. Hence, we provide a brief introduction to syntax of natural languages in Chapter 2. We identify requirements a formal model of syntax should fulfill and point out inherent limitations of certain design decisions and of a syntax-based approach to NLP in general.

- In Section 3.1 we explain the idea of hybrid trees and demonstrate why they are suited for an adequate representation of both constituent structures and dependency structures. In the remainder of Chapter 3 we formally define hybrid grammars based on *simple definite clause programs* (sDCPs) and *linear context-free rewriting systems* (LCFRSs) and how they generate languages of hybrid trees.

- Afterwards, in Chapter 4, we explore techniques to *induce* (or learn) a hybrid grammar from a corpus of hybrid trees. Grammar induction is parameterized by a *recursive partitioning strategy* that determines the string grammar and its parsing complexity: a regular grammar, a context-free grammar, or an LCFRS with an adjustable maximal fanout is obtained. Thus, we can learn a hybrid grammar suitable for efficient syntax tree generation. Further, we consider the approaches *strict labeling* and *child labeling* [NV14] to improve the generalization of the induced grammar.

- In Chapter 5 we describe our prototypical implementation of hybrid trees, hybrid grammars, grammar induction, and parsing. We test it by conducting experiments for dependency parsing on the TIGER [Bra+02], NEGRA, METU-Sabanci [Ofl+03], and SDT [De+06] treebank. Also, we report new results for constituent parsing using a larger part of TIGER than in [NV14]. We analyze the results and compare our implementation to previous approaches using the *labeled attachment score* [BM06] and the PARSEVAL [Bla+91] metric.

- We give an overview on the common transition-based [NHN06] and graph-based [McD+05] dependency parsers and the LCFRS-based method of [Kuh13; MK10] in Chapter 6. Also, we list techniques [Cha00; KM03; MMT05; Pet+06] that have turned out beneficial for constituent parsing. How these insights into existing systems might be used to improve the hybrid grammar approach is discussed in Chapter 7.

Note that we focus on studying hybrid grammars from a practical viewpoint. Thus, we specify the grammar formalism, the grammar induction, our implementation, and the evaluation setup in a formally precise manner. However, the formal properties of hybrid grammars as well as proving the correctness of our constructions are outside the scope of this thesis.

2 Syntax of natural languages

Linguists study the language that is spoken by human beings: the sounds in use (phonology), how these sounds combine to syllables and words (morphology), how words are grouped to sentences (syntax), what the meaning of a sentence is (semantics), and how sentences are used to achieve communicative goals (pragmatics). Besides this, linguists compare different languages and examine the change of languages throughout history no less than their interplay with culture and society. In this thesis we analyze a fairly new theoretic model to represent the syntax of human language. Since the author's background is computer science rather than linguistics – the same might hold for the reader – this chapter shall give a brief introduction to the essential notions as well as the potential and limitations of a syntactic view on human language. The presented material is a summary of the first three chapters of [SWB03] and selected sections of [Ben13] that was extended with additional content especially relevant for this work.

Summed up in one sentence, syntax is the difference between an arbitrary sequence of words and an acceptable sequence. Where this line should be drawn has been studied since the antiquity. For a long time the only approach to syntax was a *prescriptive* one, i.e., the objective was to frame rules on how to speak and write well:

> **A preposition is not a good word to end a sentence with.** [Fow26]

Since the late 18th century there were comparative studies of languages and only from the beginning of the 20th century the subject of research became the *description* of languages of the day. In the 1950s Noam Chomsky developed the concept of *generative grammars* [Cho65]: a formal rule-based system should be used to generate all acceptable sentences of a language. In this way precise hypotheses of language structure can be formulated and tested against examples. Chomsky supplemented his ideas with a variety of theoretic results of the expressiveness of such systems [Cho56; Cho59; CS63] and in doing so established the area of formal language theory. His work has not only shaped linguistics to this day but also influenced computer science.

2.1 Why does syntax matter?

Above, we have listed various levels of language – why should the syntactic level be chosen for natural language processing tasks? A key aspect of human languages is its compositional nature which [Sza12] summarizes as follows: The meaning of a complex expression is determined by its structure and the meaning of its constituents. In other words, to understand a sentence we only have to analyze its structure (syntax) and relate this structure to the lexical semantics of the sentence's words. This *principle of compositionality* allows humans to process unknown sentences: they can quickly communicate new topics or known topics with new sentences.

Due to the principle of compositionality, syntax should play a central role in automated processing of natural language. If we modularize the processing, we end up with a chain similar to the one in Figure 2.1. Assuming that our input is text (rather than audio samples) syntactic parsing will be a prerequisite of semantic evaluation. Syntax also restricts text synthesis. Likewise, machine translation can be done at the syntactic level as [Vau68] pointed out.

2.2 Part-of-Speech

A modest way to analyze a language syntactically could be based on categorizing words and finding common building patterns based on these categories. One option for the categorization of words is

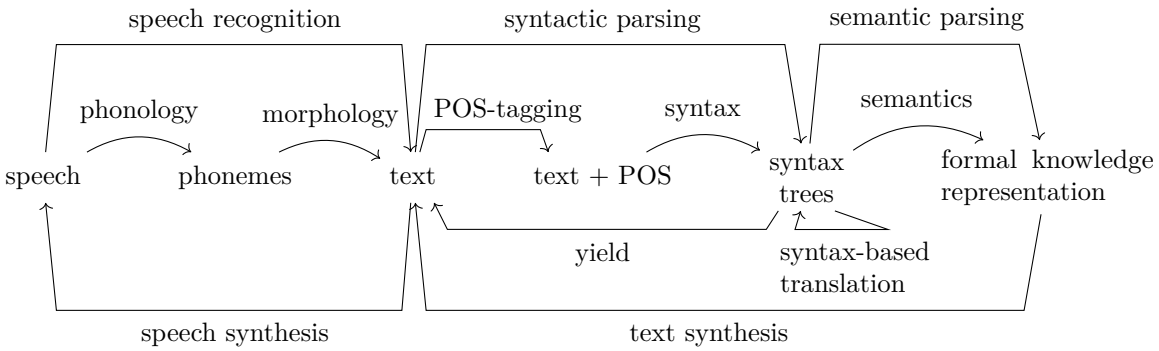


Figure 2.1: Abstract processing chain of a modular NLP system.

according to their function in the sentence which is also called *Part-of-Speech* (POS). There are four major POS that can be found in many languages: *verbs* (which act as predicates), *nouns* (which are arguments of the predicates), *adverbs* (which characterize verbs), and *adjectives* (which characterize nouns). Not every lexical element has a unique POS, e.g., the English word *round* could be a verb, an adjective, an adverb, a preposition or a noun.

Usually it is assumed that different languages or language families require their specific set of POS-tags [Ben13, p. 59]. Still, in recent years there has been effort to establish sets of universal (coarse-grained) POS-tags [PDM12] to foster the development of cross-lingual NLP applications. A frequently used set of POS-tags for German is the Stuttgart-Tübingen-tag set [Sch+99] which distinguishes 54 POS like *finite auxiliary verb* (VAFIN), *infinitive main verb* (VVINF), or *irreflexive personal pronoun* (PPER). One can also include morphological analysis like *genus*, *time*, *casus*, *person*, and *number* into the POS-tag, as it is done by the tool SMOR [SFH04] for German. However, as in many supervised machine learning contexts there is a trade-off between rich analysis and data sparseness: given a finite data set a fine-grained analysis will result in fewer samples per class than a coarse-grained one.

2.3 Constituents

One perception of the structure of some sentence is that of a *constituent tree*. A *constituent* is a group of words that relates as a unit to other words or constituents of the sentence. The constituents form a hierarchical structure, i.e., a tree, whose leaves are the words of the sentence.[1] Each complex constituent, i.e., a constituent consisting of more than one word, belongs to some *syntactic category*. Syntactic categories are often abbreviated with sequences of capital letters. For instance, some identifiers used in the German TIGER treebank [Bra+02] are *sentence* (S), *noun phrase* (NP), *verb phrase* (VP), and *adverbial phrase* (AVP). The constituent structures of a sentence may be obtained from a context-free grammar whose nonterminals are syntactic categories and POS-tags: each parse tree of the sentence is a constituent tree of this sentence.

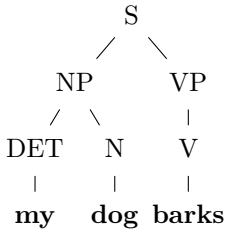
In Figure 2.2 such a context-free grammar is depicted, where we used *noun* (N), *verb* (V), *adverb* (ADV), *determiner* (DET), and the syntactic categories and POS-tags mentioned above as nonterminals. The sentences **my dog barks** and **I see the dog** have the following parse trees in this grammar:

[1] An alternative name for constituent and constituent structure is phrase and phrase structure, respectively. In this work we use the former terms.

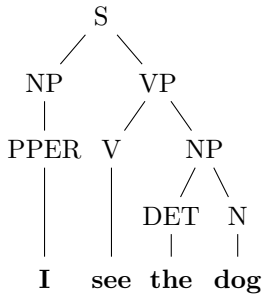
S → NP VP	N → **dog**
NP → DET N | PPER	PPER → **I**
VP → V [ADV] [NP]	V → **see** | **barks** | **disappear**
	DET → **my** | **the**

Figure 2.2: Productions of a context-free grammar where a vertical bar denotes an alternative and square brackets indicate optionality.

(1)

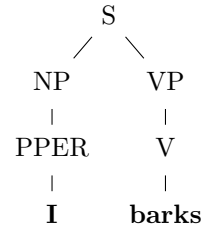


(2)

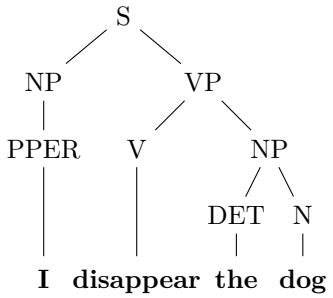


Unfortunately, our grammar is too simple: the following parse trees are also generated by our grammar, but usually they are not considered to be grammatically correct.

(3)



(4)



In (3), the *subject-verb-agreement* is violated: a 1st person singular pronoun requires a different conjugation of the verb than a 3rd person subject. One solution to this problem could be the introduction of a special syntactic category and thus the introduction of a nonterminal and special productions for each combination of person and number. This technique is called *subcategorization* [Cho65]. In (4) we exchanged the *transitive* verb **see** by the *intransitive* verb **disappear**: **see** accepts a *direct object* whereas **disappear** does not. There are even *bitransitive* verbs like **buy**, also accepting an indirect object:

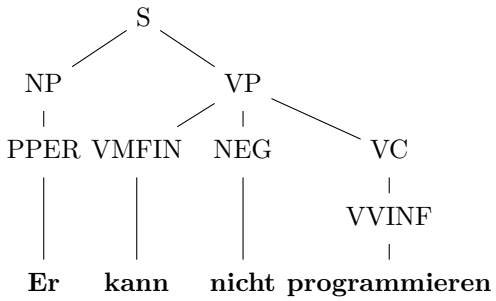
I buy you a beer.

To capture this phenomenon, which is called *valence* of the verb, we could split the V-nonterminals again but this time into three. Since this split is orthogonal to the one for subject-verb-agreement, we have a distinct V-nonterminal for each combination of number, person, and valence. However, it seems desirable to describe orthogonal phenomena isolated from one another since otherwise the grammar is rather large and redundant. Also, if the grammar shall be learned from examples instead of being hand-written by a linguist, then joining orthogonal concepts will require a training example for each possible combination. Thus, for practical application one needs to balance granularity of nonterminals with the size of the training material.[2]

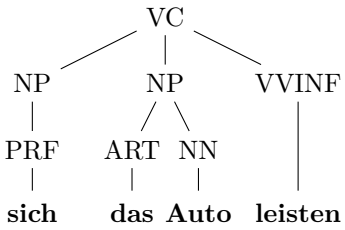
A second phenomenon is of special interest in this thesis. It often occurs in languages with *free word order* like Latin, German, Dutch, and Czech but one can also give examples for English [McC82]. The

[2]To avoid frequent subcategorization [SWB03] describe a device called *headed phrase structure grammars*: nonterminals are equipped with features like number or transitivity on which productions may enforce constraints.

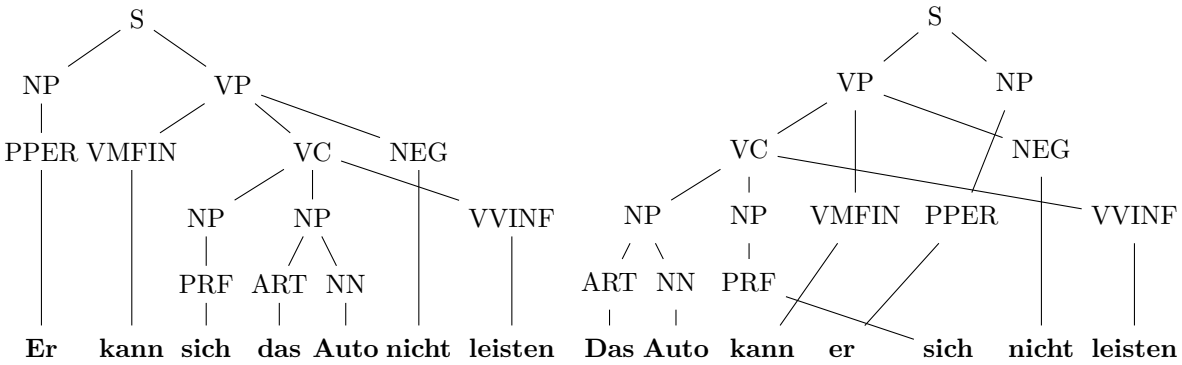
former kind of languages have a high degree of freedom in how words are arranged in the *surface form*, i.e., the words of a sentence in the order they are written or spoken. Consider the German sentence **Er kann nicht programmieren.** (he cannot program) and its constituent structure:



Assume that we exchange the *verb complement* (VC) spanning **programmieren** by **sich das Auto leisten** (afford (himself) the car):



Now the word order in the German sentence needs to be adjusted, resulting in one of the following degenerated trees:



We still have a tree-shaped hierarchy of constituents, however some constituents no longer correspond to a continuous part of the sentence, e.g., VP and VC. Hence, we call such a constituent structure *discontinuous*. Obviously, these structures cannot be parse trees of a context-free grammar.

Note that the children of S in the left tree are NP and VP whereas the children of S in the right tree are VP and NP. This is due to NP containing the first word of the sentence in the left tree and VP containing the first word of the sentence in the right tree. But is this really a structural difference or should one decouple the syntactic hierarchy from word order? At least for languages with free word order the latter might be advantageous [Ofl+03].

We summarize this section by a (non-exhaustive) list of choices that the linguist has to make during the design of a grammar or during the creation of a constituent treebank.

- The granularity of the POS-tags and the number of syntactic categories needs to be chosen.

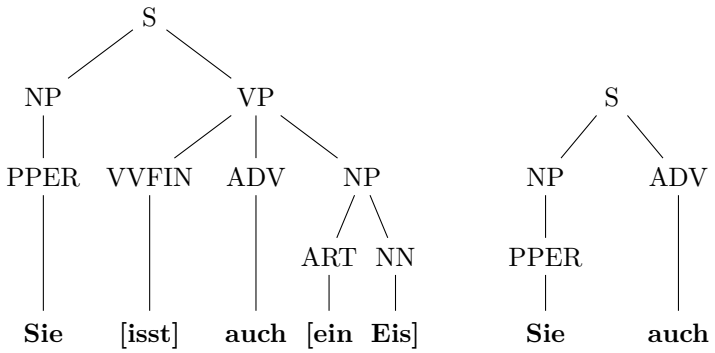


Figure 2.3: **Er isst ein Eis. Sie auch.** (He eats ice cream. She too.) Two possible constituent structures for the second sentence are depicted above.

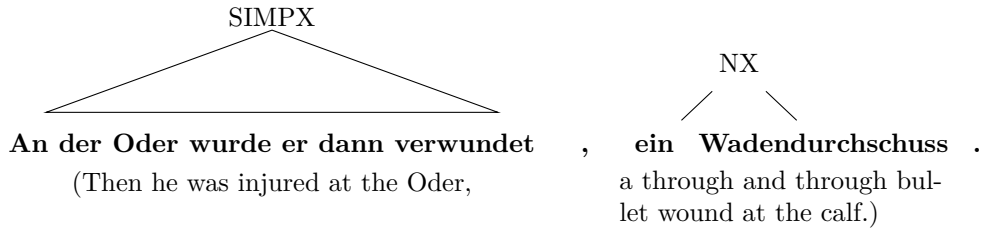


Figure 2.4: A simplified constituent structure from TüBa-D/Z that consists of four disconnected syntax trees: the first and the third tree are nontrivial hierarchies whereas the second and the fourth one consist of the comma and the period, respectively.

- Shall additional information like labeled edges or morphological tags for every word be included?
- What is the correct position for attachment of certain constituents? For instance, should the predicate of a German sentence be attached to the root symbol or to some intermediate verb phrase node?
- Especially in spoken word but also in text, words can be missing (un-)intentionally, a phenomenon called *ellipsis*. Shall the constituent structure include an empty category [Fea01] in this case? Two possible constituent structures for an elliptical sentence are depicted in Figure 2.3.
- Discontinuity can be avoided completely by attaching every child that causes it at a higher node. For instance, this strategy is used in the German TüBa-D/Z [Hin+04].
- What constitutes a syntactic unit? Typically, this is one sentence from the text source. In some annotation schemes certain subclauses are not considered to be syntactically connected to the main clause. In this case we have a sequence of syntax trees instead of a single syntax tree as the example in Figure 2.4 from TüBa-D/Z illustrates.

Typically, each treebank is equipped with *annotation guidelines* that makes the used methodology transparent to the users of the treebank. Since our aim is to learn a grammar from a treebank, we should make sure that our model is capable of representing all the above phenomena.

2.4 Dependencies

An alternative view on the syntactic structure of a sentence is that of dependencies [Tes59]. This theory postulates a *governor-depend*-relation on the words of a sentence which is directed and acyclic.

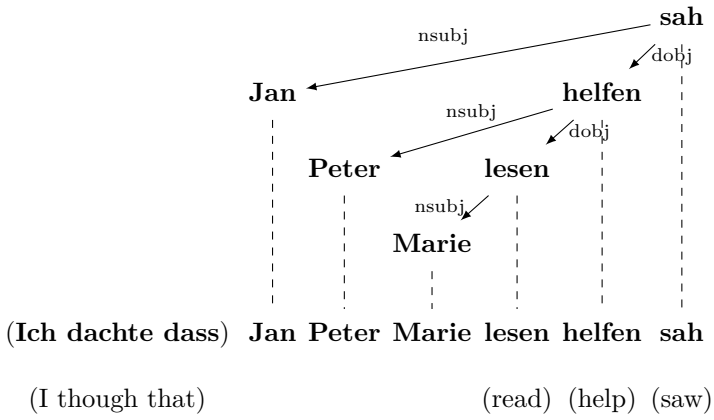


Figure 2.5: A part of the projective dependency structure of the German sentence **Ich dachte dass Jan Peter Marie lesen helfen sah**.

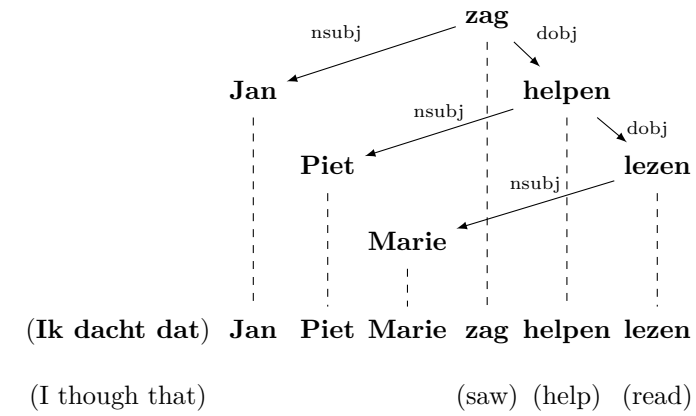


Figure 2.6: A part of the non-projective dependency structure of the Dutch sentence **Ik dacht dat Jan Piet Marie zag helpen lezen**.

Dependency structures have a close relation to constituent structures: in each constituent there is a *syntactic head* (or *head*) which identifies the central word of the constituent, e.g., the head of a constituent consisting of a noun, an adjective, and a denominator is the noun. The remaining words of the constituent are direct or transitive *dependents* of the head. There are two types of direct dependents: *arguments* are required by the head to complete its meaning whereas optional *adjuncts* refine the meaning.

The natural head of a sentence is its predicate, i.e., a verb. The verb can command arguments like subject, direct object, and indirect object. As pointed out in the previous section, this depends on the valence of the verb. The verb may also be equipped with an adverb which would be an adjunct.

The dependency structures of a German and a Dutch sentence (from [Kuh13, Figure 1]) are depicted in Figure 2.5 and Figure 2.6, respectively. Both sentences may be translated with **I thought that Jan saw Peter helping Mary reading**. In both dependencies structures each arc from some head to a dependent is annotated with a label that specifies the syntactic function of the dependent w.r.t. the head. This label is also called *dependency relation* (DEPREL). In both examples we used Universal Standford Dependencies [Mar+14].[3]

[3]The edge labels were annotated by the author and shall only serve as an example. Be aware that these labels are linguistically questionable since an infinite verb form cannot command a subject. Thus, the depicted dependencies are rather semantic than syntactic. The author thanks Thomas GroSS for pointing this out (`http://linguistics.`

Observe that in the German dependency structure the dashed lines do not cross the solid edges of the tree whereas this phenomenon occurs in the Dutch one. The property the German dependency structure has is called *projectivity*: a dependency structure is *projective* if for every word m of the sentence, it holds that if a word n is a descendant of m and a word w occurs between m and n in the surface form, then w is also a descendant of m. Otherwise, the dependency structure is called *non-projective*. Every subtree of a dependency structure can be seen as one constituent of the sentence. In this sense the notions of non-projectivity and discontinuity coincide. Thus, also non-projective dependency structures cannot be represented by context-free grammars, whereas the construction of [Kuh13] yields a context-free grammar for a projective one.

Dependency structures are used in NLP applications such as relation extraction [CS04], machine translation [DP05], and lexical resource augmentation [SJN05]. They contain most of a sentence's predicate-argument information but are flatter than constituent structures since every word in the sentence corresponds only to one tree node. Their smaller size makes them easier to learn and to parse [McD+05]. Also, the order of a word's dependents in the dependency structure is normally not considered. To this end, dependency structures have been argued to be more natural for languages with free word order [Ofl+03] than constituent structures where one typically attaches importance to this order.

Most of the phenomena and annotation decisions listed for constituent structures apply as well for dependency structures. Thus, we should make sure that our model of dependency structures has the flexibility to represent them.

2.5 Limitations of purely syntactic approaches

We argued earlier that the syntax-component should play a key role in our natural language processing chain, cf. Figure 2.1. In practice it is complicated or impossible to decompose the processes since the levels of language are interwoven: Syntax is closely connected to morphology, at least in highly *inflected* languages, i.e., languages with an elaborated declension and conjugation system. Moreover, we saw that the POS but also properties like transitivity of verbs depend strongly on lexical information.

On the other hand, we are faced with ambiguity on syntactic level:

The astronomer saw a student with a telescope.

There are two distinct meanings of this sentence each with its own syntax tree. The first is that the astronomer was looking through a telescope and saw a student. The second is that the student that was seen by the astronomer had a telescope. Given only the sentence both interpretations are equally good. Although many sentences in human communication are syntactically ambiguous, humans are able to use language quite effectively since the brain is good in filtering unlikely meaning. Humans decide for the most likely syntax of a sentence by including frequency biases and semantic aspects like the textual, local, and temporal context in which the sentence appears [SWB03, p.13]. To exemplify the robustness of the human brain we consider the next example.

She found the book on the atom. [SWB03]
She found the book on the table.

Although both sentences are almost identically, in fact both correspond to the same sequence of POS-tags, the human brain easily assigns different syntactic structures. We know that an atom is not a good location to **find a book** and that **a table** is more likely to be the location than the topic of a **book**. One way to deal with this uncertainty in an automated setting is adding weights to the formal grammar that represents the language's syntax. Ideally, one would need to readjust these weights according to the domain or topic of the text. Thereby, one should also consider shifts of the topic and thus weight-adjustments within the processing of a single text.

Furthermore, there are certain phenomena that seem to require semantic parsing. If we translate from a language where tenses have aspect semantics, e.g., the Continuous tenses in English which can

stackexchange.com/questions/12998/).

express displeasure, into languages without it, then we are likely to lose information in a mere syntax-based system. Also, there are exceptions to the principle of compositionality: in case of idioms one must not decompose a phrase in order to understand its meaning. Moreover, if the addressee of a text is supposed to know certain information, then required arguments might be missing. Lastly, irony contradicts compositionality since some opposite of the literal meaning is expressed. Instead of the syntax the semantic context of the expression as well as the tone of the speaker, i.e., phonology, is helpful to discover it.

We conclude that there are clear limitations in what we can expect from a syntax-based model of natural language, i.e., we cannot expect perfect accuracy. Despite these drawbacks good syntax models are very useful.

- Syntax helps to filter out obviously wrong meaning.

  The cat bites the dog.
  The dog bites the cat.

  Both sentences consist of the same words. Since in English the word order prescribes a word's function, both sentences have a different syntactic structure. Also, the semantics of **cat** differs from the semantics of **dog**. Thus, the sentences do not have the same semantics.

- Compositionality remains an important feature of human language that determines the semantic especially in complex expressions.

- Although the interpretation of a sentence or the resolving of its syntactic ambiguities often requires semantic context, it is important to note that one language shares the same syntax across domains. Recently, [Ben+15] argued that one should separate sole syntactic meaning (sentence meaning) from additional layers of interpretation during semantic corpus annotation. One of the advantages is that this distinction allows for reuse of the syntactic part in other domains.

- In the last years syntax-based approaches to statistical machine translation have improved significantly and started to outperform phrase-based methods. We confer to the results in the translation task of the latest *Workshop on Statistical Machine Translation* [Boj+14].

- We may combine a good syntax model with other components of the natural language processing chain. Of course we could pipeline a single result from stage to stage. However, if each stage is weighted, then it might be possible to give a product construction to choose the overall best decisions [Büc15].

3 Hybrid trees and hybrid grammars

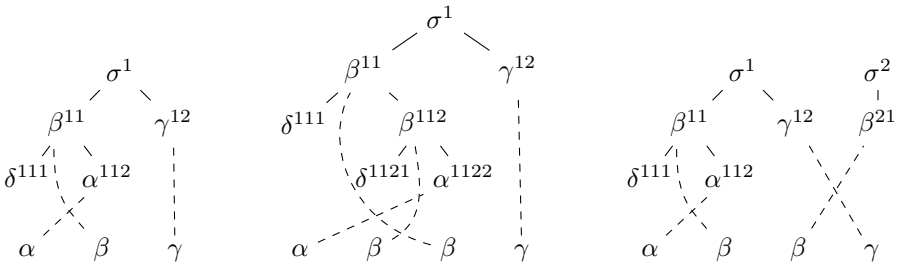
In this chapter we present how discontinuous phrase structures and non-projective dependency structures can be modeled with hybrid trees and hybrid grammars [NV14]. We start by giving an intuition for hybrid trees and exemplify why they are well-suited for the representation of both foregoing views on syntax. Also, we explain how a careful choice of certain parameters allows us to account for other syntactic phenomena outlined in the previous chapter. Afterwards, we recall some mathematical notions and fix a model for trees and strings before giving a formal definition of hybrid trees and LCFRS/sDCP-hybrid grammars.

3.1 Separating syntactic hierarchy from surface form

The problem that arises if a discontinuous constituent structure or a non-projective dependency structure shall be represented with a usual tree is that at least one subtree does not correspond to a continuous part of the surface form. The idea of a hybrid tree is to represent the syntactic hierarchy t and the surface form s of the sentence separately from another. To this end, a hybrid tree has two components: firstly, it consists of a sequence of unranked trees t. We assume that t has a finite domain of positions in the spirit of [Gor67]. The hybrid tree's second component is a linear order $\leq_t$ on a subset of the positions of t: this is the order of the lexical elements in the surface form of the sentence. Obviously, the linear order $\leq_t$ corresponds to a string over positions from t. If each position in this string is replaced with the symbol at this position in t, then the surface form s of the sentence is obtained.

Let Γ and Σ be alphabets of symbols such that $\Sigma \subseteq \Gamma$. To account for some natural properties we require that t is a sequence of unranked trees over Γ and that $\leq_t$ is a linear order over the set of positions of t that are labeled with an element of Σ.

Example 3.1. Let $\Gamma = \{\alpha, \beta, \gamma, \delta, \sigma\}$ and $\Sigma = \{\alpha, \beta, \gamma\}$ be alphabets. Below we depicted three hybrid trees over (Σ, Γ). Each of them is depicted as a pair of a sequence of trees t and a string s. At every node of t the Gorn address of this node is annotated in superscript. Since t is a sequence of trees, the root of the i-th tree in this sequence has the address i. Each dashed line associates a position in t that is labeled by a symbol from Σ with an occurrence of the same symbol in the string s. In this way the linear order $\leq_t$ is specified.



For instance, the linear order is $112 <_t 11 <_t 12$ in the first hybrid tree, $1122 <_t 112 <_t 11 <_t 12$ in the second one, and $112 <_t 11 <_t 21 <_t 12$ in the third one. ■

In most of the subsequent examples we indicate $\leq_t$ with such dashed lines pointing to a string.

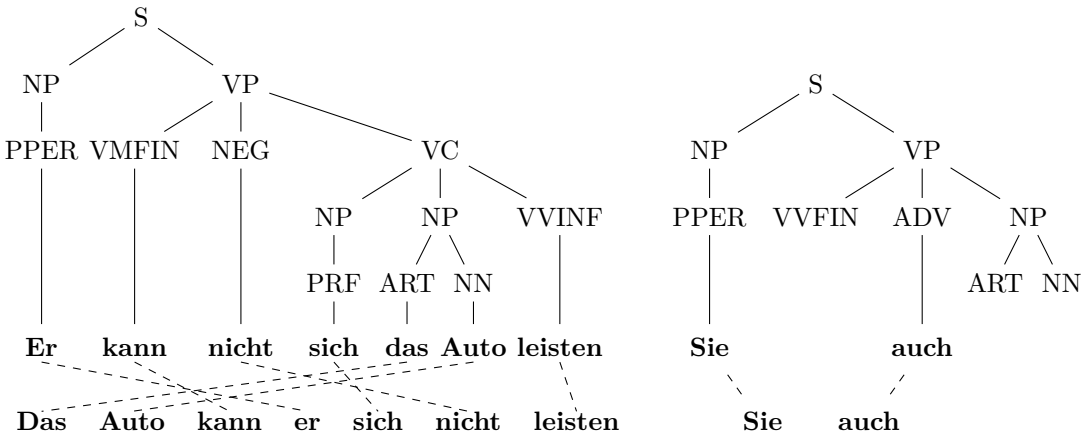


Figure 3.1: Hybrid trees for a discontinuous and an elliptic constituent structure.

3.1.1 Constituent structures

How can a constituent structure be understood as a hybrid tree? Let Γ be the alphabet containing syntactic categories and lexical elements and Σ be the alphabet of lexical elements. Let $(t, \leq_t)$ be a hybrid tree such that every node labeled by an element of Σ is a leaf node. We depicted two constituent structures as hybrid trees in Figure 3.1. The hybrid tree on the left illustrates that we can represent discontinuous constituent structures by amending the order on the leaves labeled with Σ given by the usual tree yield. Also, since we do not require that every leaf node is labeled with an element from Σ we allow for empty categories, however, we cannot specify where (w.r.t. $\leq_t$) the ellipsis is located. The hybrid tree on the right in Figure 3.1 illustrates this. Furthermore, it is not hard to see that we can incorporate constituent structures with disconnected substructures as the one in Figure 2.4 into our model since t is a sequence of trees.

3.1.2 Dependency structures

Hybrid trees are also capable of representing dependency structures. For this we identify both alphabets Γ and Σ, i.e., every node in t is in the linear order. If we want to incorporate dependency labels, then we choose our alphabet to contain pairs of lexical items and dependency labels. In Figure 3.2 we depict a non-projective part of the dependency structure in Figure 2.6 in two ways as hybrid tree, where the upper one does not contain dependency labels but the lower one does. Also, dependency structures with disconnected substructures can be represented. For instance, the dependency structure in Figure 5.1d on page 46 consists of five disconnected substructures.

Since we assume Gorn addresses on the tree structure, the order of subtrees is explicit in a hybrid tree. As we mentioned above, this might be disadvantageous for modeling languages with free word order. Though one could fix a canonical order (e.g., the lexicographical order of the lexical elements) on sibling nodes to harmonize and thus conceal the subtree order, we do not investigate this option in this thesis. Possibly, distinguishing the order of subtrees might have advantages in itself as it allows us to distinguish certain building patterns.

3.1.3 Hybrid grammars

Since natural languages permit an infinite number of sentences [SWB03, Chapter 2], we aim to define finite devices that indicate whether a sentence is acceptable or not. Thus, in the hybrid tree setting we have to find a way to describe languages of hybrid trees.

The tree component of a hybrid tree may be generated by a tree grammar. Recall that the linear order of a hybrid tree defines a string. Thus, we use a string grammar to describe it. In order to establish

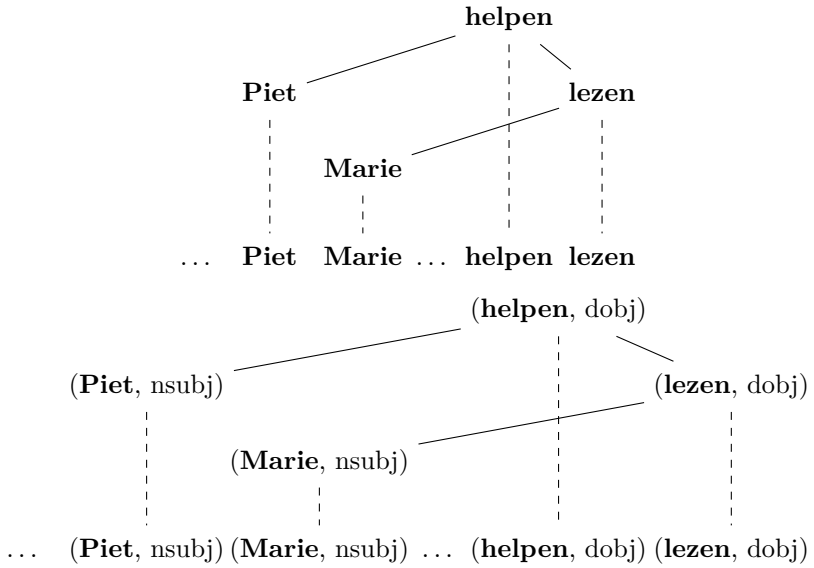
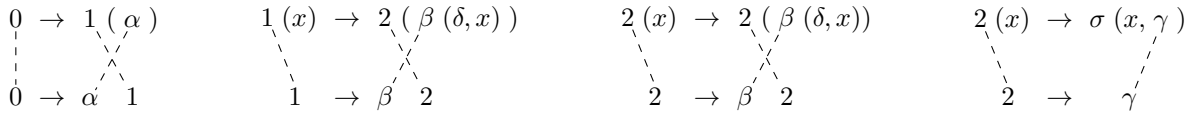


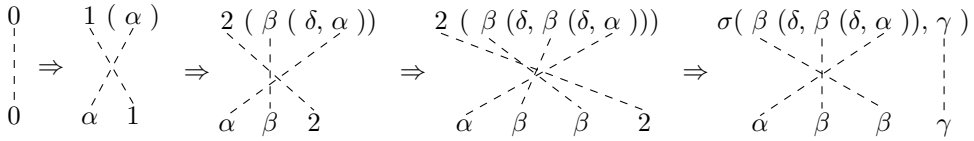
Figure 3.2: A dependency structure with and without dependency labels modeled as a hybrid tree.

a connection between string and tree we combine both grammars to a hybrid grammar. A hybrid grammar contains a finite set of *hybrid productions*. Each hybrid production is a pair of a string grammar production and a tree grammar production whose derivational nonterminals are linked similarly as in synchronous grammars [SS90; SP05]. Additionally, we injectively map each occurrence of a terminal in the string grammar production to a terminal occurrence in the tree grammar production.

Example 3.2. We give a grammar that generates (among others) the first two hybrid trees of Example 3.1. It consists of a regular string grammar with nonterminals 0, 1, and 2, terminal alphabet Σ, and start symbol 0. The tree grammar is a monadic context-free tree grammar [Rou69; ES77]. It has the same nonterminals and start symbol as the string grammar but the terminal alphabet is Γ. The set of hybrid productions is provided as follows, where the tree grammar production is denoted at the top and the string grammar production is denoted at the bottom. A link between two terminals or two nonterminals is indicated by a dashed line.



This is the derivation of the second tree depicted in Example 3.1, where the above productions were applied in the order from left to right.



■

The language of the string grammar in the above example is $\{\alpha\ \beta^n\ \gamma \mid n > 0\}$. Since it is regular, it is very efficient to parse. Synchronously, the tree grammar generates a tree that is richer than the string grammar's parse tree. This illustrates the key advantage of hybrid grammars: by explicitly modeling the discontinuous structure to be part of the grammar's language – instead of being one of its parse trees – we enjoy great freedom in how we generate this structure. We can combine a simple string grammar with an expressive tree grammar and thereby decouple parsing complexity from the expressiveness of the hybrid grammar, i.e., the degree of discontinuity [Kuh13] the grammar allows for.

3.2 Mathematical foundations

Let A and B be sets. We denote the cardinality of A by $|A|$, the cartesian product of A and B by $A \times B$, the power set of A by $\mathfrak{P}(A)$, and the free monoid over A by A^*. We refer to the empty word by ε and denote $A^* \setminus \{\varepsilon\}$ by A^+. We define projections π_1 and π_2 such that, for every $(a, b) \in A \times B$, we have that $\pi_1((a, b)) = a$ and $\pi_2((a, b)) = b$. Let $\varphi\colon A \to B$ be a mapping. We refer by $\varphi(A)$ to the set $\{b \in B \mid \exists a \in A\colon \varphi(a) = b\}$. The set of nonnegative integers including 0 is denoted by $\mathbb{N}$ and the set of positive integers by $\mathbb{N}_+$. Let $n \in \mathbb{N}$. We abbreviate $a_1, \ldots, a_n$ by $a_{1,n}$. For every $n \in \mathbb{N}$, we let $[n] = \{1, 2, \ldots, n\}$ and $[n]_0 = \{0, 1, 2, \ldots, n\}$. Let $X = \{x_j^{(i)} \mid i \in \mathbb{N} \cup \{\spadesuit\}, j \in \mathbb{N}_+\}$ be the set of *variables* and, for every $I \subseteq (\mathbb{N} \cup \{\spadesuit\}) \times \mathbb{N}_+$, let $X_I = \{x_j^{(i)} \in X \mid (i, j) \in I\}$. For every $j \in \mathbb{N}_+$, we abbreviate $x_j^{(\spadesuit)}$ by x_j and, for every $n \in \mathbb{N}$, we let $X_n = X_{\{\spadesuit\} \times [n]}$. Note that $X_0 = \emptyset = X_\emptyset$.

Let A be a set and ϑ be a binary relation on A, i.e., $\vartheta \subseteq A \times A$. We denote the *transitive closure of* ϑ by ϑ^+ and the *reflexive, transitive closure of* ϑ by ϑ^*. We say that ϑ is a *strict order* if ϑ is irreflexive and transitive. Moreover, ϑ is a *total order* if ϑ is a strict order and, for each $a, b \in A$ such that $a \neq b$, we have that $(a, b) \in \vartheta$ or $(b, a) \in \vartheta$. If ϑ is a strict order, then we denote its *reflexive closure* by $\underline{\vartheta}$, i.e., $\underline{\vartheta}$ is the *partial order* (reflexive, transitive, antisymmetric) corresponding to ϑ. Likewise, if $\underline{\vartheta}$ is a partial order, then the corresponding strict order is denoted by ϑ. If ϑ is a binary relation, then we denote $(a, b) \in \vartheta$ also by $a \,\vartheta\, b$.

3.3 Alphabets and s-terms.

A *ranked set* is a tuple $(\Sigma, \mathrm{rk}_\Sigma)$ where Σ is a set of *symbols* and $\mathrm{rk}_\Sigma\colon \Sigma \to \mathbb{N}$ assigns a nonnegative integer, called *rank*, to each symbol. For every $k \in \mathbb{N}$, we let $\Sigma^{(k)} = \{a \in \Sigma \mid \mathrm{rk}_\Sigma(a) = k\}$. Whenever convenient we identify Σ and $(\Sigma, \mathrm{rk}_\Sigma)$. If Σ is finite and nonempty, then we call Σ *ranked alphabet*. If also $\Sigma = \Sigma^{(0)}$, then we say that Σ is an *alphabet* and if $\Sigma = \Sigma^{(1)} \cup \Sigma^{(0)}$, then we say that Σ is a *monadic alphabet*. If $(\Sigma, \mathrm{rk}_\Sigma)$ and $(\Delta, \mathrm{rk}_\Delta)$ are ranked sets such that $\Sigma \cap \Delta = \emptyset$, then we denote the ranked set $\Gamma = (\Sigma \cup \Delta, \mathrm{rk}_\Gamma)$ by $\Sigma \cup \Delta$ where $\mathrm{rk}_\Gamma(\gamma)$ is $\mathrm{rk}_\Sigma(\gamma)$ if $\gamma \in \Sigma$, and $\mathrm{rk}_\Delta(\gamma)$ otherwise. If $(\Sigma, \mathrm{rk}_\Sigma)$ and $(\Gamma, \mathrm{rk}_\Gamma)$ are ranked sets such that $\Sigma \subseteq \Gamma$ and, for every $\sigma \in \Sigma$, it holds that $\mathrm{rk}_\Sigma(\sigma) = \mathrm{rk}_\Gamma(\sigma)$, then we denote the ranked set $(\Sigma, \mathrm{rk}_\Sigma)$ also by $(\Sigma, \mathrm{rk}_\Gamma)$.

In order to represent unranked trees, sequences of trees, and strings we use a notion of *sequence terms* inspired by [SK08]. We constraint the underlying ranked set to be monadic as it suffices for our purpose.

Definition 3.3. Let Σ be a monadic ranked set and U be a set such that $\Sigma \cap U = \emptyset$. We define the set of *terms over Σ indexed by U*, denoted by $\mathrm{T}_\Sigma(U)$, and the set of *sequence terms (s-terms) over Σ indexed by U*, denoted by $\mathrm{T}^*_\Sigma(U)$, simultaneously as follows:

(i) $U \cup \Sigma^{(0)} \subseteq \mathrm{T}_\Sigma(U)$,

(ii) if $\sigma \in \Sigma^{(1)}$ and $s_1 \in \mathrm{T}^*_\Sigma(U)$, then $\sigma(s_1) \in \mathrm{T}_\Sigma(U)$, and

(iii) if $n \in \mathbb{N}$ and $t_1, \ldots, t_n \in \mathrm{T}_\Sigma(U)$, then $(t_1, \ldots, t_n) \in \mathrm{T}^*_\Sigma(U)$. ■

Given a monadic ranked set Σ we conceive of an unranked tree as a term over Σ and regard a sequence of unranked trees as an s-term over Σ. If Σ is an alphabet, then an s-term over Σ can be seen as a string.

Definition 3.4. Let Σ be a monadic ranked set and U be a set disjoint from Σ. Let $s_1 = (t_1, \ldots, t_n)$ and $s_2 = (t_{n+1}, \ldots, t_{n+m})$ be s-terms where $n, m \in \mathbb{N}$ and $t_1, \ldots, t_{n+m} \in T_\Sigma(U)$. The *concatenation of s_1 and s_2*, denoted by $s_1 \diamond s_2$, is the s-term $(t_1, \ldots, t_{n+m})$. Note that $\diamond$ is associative. Let $s \in \mathrm{T}^*_\Sigma(U) \cup \mathrm{T}_\Sigma(U)$ and $\Delta \subseteq \Sigma \cup U$. The set of *$\Delta$-positions of s*, denoted by $\mathrm{pos}_\Delta(s)$, is the subset of $\mathfrak{P}(\mathbb{N}_+{}^*)$ defined

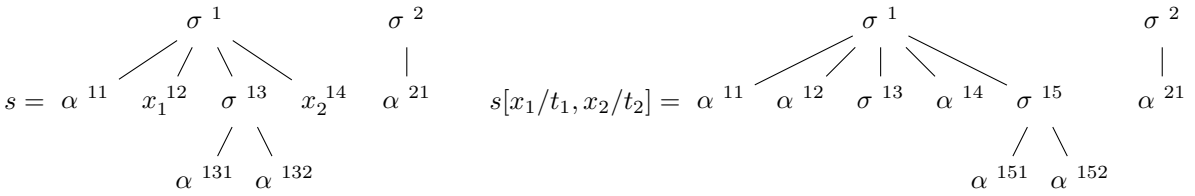


Figure 3.3: The s-term s before and after substitution of variables by s-terms, where $t_1 = (\alpha, \sigma(()), \alpha)$, $t_2 = ()$, and positions are annotated in superscript to each node.

inductively as follows:

$$\mathrm{pos}_\Delta(s) = \begin{cases} \{\varepsilon\} & \text{if } s \in (U \cup \Sigma^{(0)}) \cap \Delta, \\ \emptyset & \text{if } s \in (U \cup \Sigma^{(0)}) \setminus \Delta, \\ \{\varepsilon\} \cup \mathrm{pos}_\Delta(s_1) & \text{if } s = \sigma(s_1) \in \mathrm{T}_\Sigma(U) \text{ and } \sigma \in \Delta, \\ \mathrm{pos}_\Delta(s_1) & \text{if } s = \sigma(s_1) \in \mathrm{T}_\Sigma(U) \text{ and } \sigma \notin \Delta, \\ \{ip \mid i \in [n], p \in \mathrm{pos}_\Delta(t_i)\} & \text{if } s = (t_1, \ldots, t_n) \in \mathrm{T}^*_\Sigma(U). \end{cases}$$

We abbreviate $\mathrm{T}^*_\Sigma(\emptyset)$ by T^*_Σ, $\mathrm{T}_\Sigma(\emptyset)$ by T_Σ, and $\mathrm{pos}_{\Sigma \cup U}(s)$ by $\mathrm{pos}(s)$. Let now $p \in \mathrm{pos}(s)$. The *symbol of s at p*, denoted by $s(p)$, is defined inductively as follows:

$$s(p) = \begin{cases} \sigma & \text{if } p = \varepsilon \text{ and } s = \sigma(s') \in \mathrm{T}_\Sigma(U), \\ s & \text{if } p = \varepsilon \text{ and } s \in \Sigma^{(0)} \cup U, \\ s'(p) & \text{if } p \neq \varepsilon \text{ and } s = \sigma(s') \in \mathrm{T}_\Sigma(U), \\ t_i(p') & \text{if } p = ip' \text{ and } s = (t_1, \ldots, t_n) \in \mathrm{T}^*_\Sigma(U). \end{cases}$$

Let Σ be a monadic alphabet and let I and J be finite subsets of $(\mathbb{N} \cup \{\spadesuit\}) \times \mathbb{N}_+$. For every $(i,j) \in I$, let $t^{(i)}_j \in \mathrm{T}^*_\Sigma(X_J)$. We define inductively the *substitution of $x^{(i)}_j$ by $t^{(i)}_j$ for each $(i,j) \in I$* to be the function $\cdot\,[x^{(i)}_j / t^{(i)}_j \mid (i,j) \in I] \colon \mathrm{T}^*_\Sigma(X_I) \cup \mathrm{T}_\Sigma(X_I) \to \mathrm{T}^*_\Sigma(X_J)$:

$$s \mapsto \begin{cases} t^{(i)}_j & \text{if } s = x^{(i)}_j \in X_I, \\ (s) & \text{if } s \in \Sigma^{(0)}, \\ (\sigma(s_1[x^{(i)}_j / t^{(i)}_j \mid (i,j) \in I])) & \text{if } s = \sigma(s_1) \in \mathrm{T}_\Sigma(X_I), \\ \cdot\,[x^{(i)}_j / t^{(i)}_j \mid (i,j) \in I](s_1) \diamond \cdots \diamond \cdot\,[x^{(i)}_j / t^{(i)}_j \mid (i,j) \in I](s_k) & \text{if } s = (s_1, \ldots, s_k) \in \mathrm{T}^*_\Sigma(X_I). \end{cases}$$

We denote $\cdot\,[x^{(i)}_j / t^{(i)}_j \mid (i,j) \in I](s)$ also by $s[x^{(i)}_j / t^{(i)}_j \mid (i,j) \in I]$. ■

Note that each variable, i.e., a term, is substituted by an s-term. The consequence is that positions in s may *disappear* or get *shifted* after substitution.

Example 3.5. Let Σ be a monadic alphabet where $\Sigma^{(1)} = \{\sigma\}$ and $\Sigma^{(0)} = \{\alpha\}$. Consider the s-terms $s = (\sigma((\alpha, x_1, \sigma((\alpha,\alpha)), x_2)), \sigma((\alpha)))$, $t_1 = (\alpha, \sigma(()), \alpha)$, and $t_2 = ()$ and observe that $\mathrm{pos}(s) = \{1, 11, 12, 13, 131, 132, 14, 2, 21\}$. If $s' = s[x_1/t_1, x_2/t_2]$, then $s' = (\sigma(\alpha, \alpha, \sigma(()), \alpha, \sigma((\alpha,\alpha))), \sigma((\alpha)))$ and $\mathrm{pos}(s') = \{1, 11, 12, 13, 14, 15, 151, 151, 2, 21\}$. A visualization of s and s' is given in Figure 3.3. Note how the positions 13, 131, and 132 in s correspond to 15, 151, and 152 in s', respectively. This is an example for shifted positions. Also, the position 14 in s has no correspondence in s' since it was substituted by an empty s-term. Here a position disappears. ■

Definition 3.6. We denote the *lexicographical order* on $\mathbb{N}^*$ by $<_\ell$. We define the *sibling order* on $\mathbb{N}_+{}^*$, denoted by $<_{\mathrm{sib}}$, as follows. Let $p \in \mathbb{N}_+{}^*$ and $i, j \in \mathbb{N}_+$. Then $pi <_{\mathrm{sib}} pj$ if $i < j$. Also, let the *child relation* (or *prefix relation*) on $\mathbb{N}_+{}^*$ be χ, where $p \,\chi\, q$ if there exists $i \in \mathbb{N}_+$ such that $pi = q$. ■

We may use the reflexive, transitive closure of χ to refer to an arbitrary *descendant* of some position.

Definition 3.7. Let Σ be a monadic alphabet, U a finite set such that $U \cap \Sigma = \emptyset$, $s \in \mathrm{T}^*_\Sigma(U) \cup \mathrm{T}_\Sigma(U)$, $n \in \mathbb{N}_+$, and $p_1, \ldots, p_n \in \mathrm{pos}(s)$. Then $p_1 \cdots p_n$ is a *sequence of consecutive sibling positions* if $n = 1$ or if there exist $p' \in {\mathbb{N}_+}^*$ and $i \in \mathbb{N}$ such that, for every $j \in [n]$, it holds that $p_j = p'(i + j)$. For every sequence of consecutive sibling positions $p_1 \cdots p_n$ where $n \in \mathbb{N}_+$ and $p_1, \ldots, p_n \in \mathrm{pos}(s)$, we define the *sub-s-term of s at the positions $p_1 \cdots p_n$*, denoted by $s|_{p_1 \cdots p_n}$, as follows:

$$s|_{p_1 \cdots p_n} = \begin{cases} (s) & \text{if } n = 1,\ p_1 = \varepsilon \text{ and } s \in \mathrm{T}_\Sigma(U), \\ (s_{i+1}, \ldots, s_{i+n}) & \text{if } s = (s_1, \ldots, s_m) \in \mathrm{T}^*_\Sigma(U) \text{ and } \exists i \in [m-1]_0 \colon \forall j \in [n] \colon p_j = i + j, \\ s_1|_{p_1 \cdots p_n} & \text{if } \forall j \in [n] \colon p_j \neq \varepsilon, \text{ and } s = \sigma(s_1) \in \mathrm{T}_\Sigma(U), \\ s_i|_{p'_1 \cdots p'_n} & \text{if } s = (s_1, \ldots, s_m) \in \mathrm{T}^*_\Sigma(U) \\ & \text{and } \exists i \in [m] \colon \forall j \in [n] \colon \exists p'_j \in {\mathbb{N}_+}^* \colon p_j = i p'_j. \end{cases}$$

We extend χ to sequences of consecutive sibling positions: let $u = p(i+1) \cdots p(i+n)$ and $w = p'(i'+1) \cdots p'(i'+n')$ for some $p, p' \in {\mathbb{N}_+}^*$, $i, i' \in \mathbb{N}$, and $n, n' \in \mathbb{N}_+$. Then $u \,\chi\, w$ if there exists $l \in [n]$ such that $p(i+l) = p'$. Moreover, w is a *strict subsequence of* u, denoted by $u \propto w$, if $p = p'$, $i \leq i'$, $n' < n$, and $n' + i' \leq n + i$.

Let now $s \in \mathrm{T}^*_\Sigma(U)$ and let $u, w \in \mathrm{pos}(s)^+$ be sequences of consecutive sibling positions such that $(u, w) \in \chi^* \cup \propto$ where $u = p(i+1) \cdots p(i+n)$ and $w = p'(i'+1) \cdots p'(i'+n')$ for some $p, p' \in {\mathbb{N}_+}^+$, $i, i' \in \mathbb{N}$, and $n, n' \in \mathbb{N}_+$. We define the *relative position of w to u*, denoted by $w \downarrow u$, as follows. If $u = w$, then $w \downarrow u = 1 \cdots n$. If $u \propto w$, then $w \downarrow u = (i' - i + 1) \cdots (i' - i + n')$. If $u \,\chi^+\, w$ where $p' = p(i+l)v$ for some $l \in [n]$ and $v \in {\mathbb{N}_+}^*$, then $w \downarrow u = lv(i'+1) \cdots lv(i'+n')$. ■

Example 3.8. Recall s' from Example 3.5. For $u = 13\ 14\ 15$ and $w = 14\ 15$ it holds that $u \propto w$. Then $w \downarrow u = 2\ 3$ and we have that

$$s'|_w = \left(\begin{array}{ccccccccccc} & & & & & & \sigma^{\,1} & & & & \sigma^{\,2} \\ & & & & & \swarrow & \downarrow & \searrow & & \downarrow & \\ \alpha^{\,11} & & \alpha^{\,12} & & \sigma^{\,13} & & \alpha^{\,14} & & \sigma^{\,15} & & \alpha^{\,21} \\ & & & & & & & & \swarrow \ \searrow & & \\ & & & & & & & & \alpha^{\,151} \quad \alpha^{\,152} & & \end{array} \right)\Bigg|_{14\ 15} = \begin{array}{ccccc} \alpha^{\,1} & & \sigma^{\,2} & & \\ & \swarrow & \searrow & & \\ & \alpha^{\,21} & & \alpha^{\,22} & \end{array}$$

$$= \left(\begin{array}{ccccc} \sigma^{\,1} & & \alpha^{\,2} & & \sigma^{\,3} \\ & & & \swarrow & \searrow \\ & & & \alpha^{\,31} & \quad \alpha^{\,32} \end{array} \right)\Bigg|_{2\ 3} = (s'|_{13\ 14\ 15})|_{14\ 15 \downarrow 13\ 14\ 15} = (s'|_u)|_{w \downarrow u}$$

■

The above equation holds for every s-term s' and sequences of consecutive sibling positions w and u satisfying a certain precondition. This can be proven by case analysis and induction on the length of the common prefix in u and w.

Lemma 3.9. *Let Σ be a monadic alphabet, U a set disjoint from Σ, $s \in \mathrm{T}^*_\Sigma(U) \cup \mathrm{T}_\Sigma(U)$, and $u, w \in \mathrm{pos}(s)^+$ consecutive sibling positions such that $(u, w) \in \chi^* \cup \propto$. Then*

$$s|_w = (s|_u)|_{w \downarrow u}.$$

Definition 3.10. Let Σ be monadic alphabet, U a set disjoint of Σ, $s \in \mathrm{T}^*_\Sigma(U) \cup \mathrm{T}_\Sigma(U)$, $w \in \mathrm{pos}(s)^+$ a sequence of consecutive sibling positions, and $t \in \mathrm{T}^*_\Sigma(U)$. We define the *substitution of the sub-s-term*

at the positions w *by* t to be the function $\cdot[t]_w \colon \mathrm{T}^*_\Sigma(U) \cup \mathrm{T}_\Sigma(U) \to \mathrm{T}^*_\Sigma(U)$ such that

$$s \mapsto \begin{cases} t & \text{if } w = \varepsilon \text{ and } s \in \mathrm{T}_\Sigma(U), \\ (s_1, \ldots, s_i) \diamond t \diamond (s_{i+n+1}, \ldots, s_m) & \text{if } s = (s_1, \ldots, s_m) \in \mathrm{T}^*_\Sigma(U) \text{ and } \exists i \in [m-1]_0 \colon \\ & \exists n \in [m-i] \colon w = (i+1) \cdots (i+n), \\ (\sigma([t]_{w_1 \cdots w_n}(s_1))) & \text{if } s = \sigma(s_1) \in \mathrm{T}_\Sigma(U) \text{ and } \exists n \in \mathbb{N}_+ \colon \\ & \exists w_1, \ldots, w_n \in \mathrm{pos}(s_1) \colon w = w_1 \cdots w_n \\ (s_1, \ldots, s_{i-1}) \diamond [t]_{w_1 \cdots w_n}(s_i) \diamond (s_{i+1}, \ldots, s_m) & \text{if } s = (s_1, \ldots, s_m) \in \mathrm{T}^*_\Sigma(U) \text{ and } \exists i \in [m] \colon \\ & \exists n \in \mathbb{N}_+ \colon \exists w_1, \ldots, w_n \in \mathrm{pos}(s_i) \colon w = i w_1 \cdots i w_n. \end{cases}$$

We denote $\cdot[t]_w(s)$ also by $s[t]_w$. ■

Similarly to substitution of variables, substitution at a position sequence w may lead to shifts or disappearances of positions. However, only positions that are descendants of some position in w or descendants of some sibling position to the right of w can be affected. Thus, assuming sequences of consecutive sibling position $w^{(1)}, \ldots, w^{(k)}$ that are pairwise not in χ^+ and non-overlapping, one can substitute at $w^{(1)}$ by $t^{(1)}$, …, at $w^{(k)}$ by $t^{(k)}$ *in parallel* by proceeding stepwise in lexicographically descending order of the $w^{(i)}$s, substituting at $w^{(i)}$ by $t^{(i)}$ in each step.

3.4 Hybrid trees and relevant restrictions for linguistics

Now that we have defined s-term for the representation of trees and strings, we can supplement Section 3.1 by a formal definition of hybrid trees, constituent structures, and (labeled) dependency structures.

Definition 3.11. Let $(\Sigma, \mathrm{rk}_\Sigma)$ be an alphabet and $(\Gamma, \mathrm{rk}_\Gamma)$ be a monadic alphabet such $\Sigma \subseteq \Gamma$. A *hybrid tree* h *over* (Σ, Γ) is a pair $(s, \leq_s)$ such that $s \in T^*_\Gamma$ and $<_s$ is a total order on $\mathrm{pos}_\Sigma(s)$. Now $\leq_s$ defines an s-term in T^*_Σ, denoted by $\mathrm{str}(h)$, as follows: let $\mathrm{pos}_\Sigma(s) = \{p_1, \ldots, p_n\}$ where $p_i \leq p_{i+1}$ holds for each $i \in [n-1]$. Then $\mathrm{str}(h) = (s(p_1), \ldots, s(p_n))$. We denote the set of all hybrid trees over (Σ, Γ) by $\mathbf{HT}(\Sigma, \Gamma)$. ■

Definition 3.12. Let $(\Gamma, \mathrm{rk}_\Gamma)$ be a monadic alphabet. A *phrase structure* or *constituent structure* is a hybrid tree over $((\Gamma^{(0)}, \mathrm{rk}_\Gamma), \Gamma)$. Naturally, we assume $\Gamma^{(1)}$ to contain the syntactic categories and $\Gamma^{(0)}$ to contain the word forms. ■

Definition 3.13. Let $\Gamma = \Gamma^{(1)}$ be a ranked alphabet and $(\Sigma, \mathrm{rk}_\Sigma)$ an alphabet such that $\Sigma = \Gamma$. A *dependency structure* is a hybrid tree over (Σ, Γ). Here Γ contains word forms. ■

Definition 3.14. Let $\Gamma = \Gamma^{(1)}$ be a ranked alphabet, where $\Gamma = \Gamma_1 \times \Gamma_2$, Γ_1 is set containing word forms and Γ_2 is a set containing dependency relations. Let $(\Sigma, \mathrm{rk}_\Sigma)$ be an alphabet such that $\Sigma = \Gamma$. A *labeled dependency structure* is a hybrid tree over (Σ, Γ). ■

3.5 Linear context-free rewriting systems

In this section we present a grammar formalism that is capable of generating mildly context-sensitive string languages [Kal10]. It was first defined as *linear context-free rewriting system* (LCFRS) by [VWJ87], but equivalent formalism like *multiple context-free grammars* [Sek+91; SK08] or *range concatenation grammars* [Bou05] have been used and studied as well. Our notion is close to [VWJ87]. A probabilistic variant of LCFRSs was addressed by [KSK06].

The central idea of LCFRSs is to generalize context-free grammars such that each nonterminal may generate multiple strings, i.e., tuples of strings. The arity of this tuple is called *fanout* of the nonterminal. Each LCFRS production consists of the context-free nonterminal backbone $A_0 \to A_1 \cdots A_n$ and a word-tuple function. The *word-tuple function* prescribes how the strings in the input tuples, i.e., the tuples

that are generated by the nonterminals on the right-hand side, combine to a tuple of strings for the nonterminal on the left-hand side. Not every function with this signature is a word-tuple function: the result may be obtained only by concatenation of terminal symbols and the components of the input tuples. Also, it is required that each component of an input tuple is used exactly once, a property also called *single syntactic use requirement* [Gie88]. Formally, we define a word-tuple function to be a tuple of s-terms over an alphabet indexed with variables. The latter represent the components of the input tuples.

Definition 3.15. Let Σ be an alphabet, $n \in \mathbb{N}$, and $k_0, \ldots, k_n \in \mathbb{N}_+$. The set of *word-tuple functions over Σ with signature $k_1 \cdots k_n, k_0$*, denoted by $\mathcal{F}^{\mathrm{LCFRS}(\Sigma)}_{k_1 \cdots k_n, k_0}$, is defined as follows. For every $i \in [n]$, we let $m_i = \sum_{j=1}^{i} k_j$. Also, we let $s_1, \ldots, s_{k_0} \in \mathrm{T}^*_\Sigma(X_{m_n})$. Then $\langle s_1, \ldots, s_{k_0} \rangle \in \mathcal{F}^{\mathrm{LCFRS}(\Sigma)}_{k_1 \cdots k_n, k_0}$ if for every $j \in [m_n]$, it holds that x_j occurs exactly once in $s_1 \diamond \cdots \diamond s_{k_0}$. ■

Definition 3.16. A *linear context-free rewriting system* (LCFRS) is a tuple $G = (N, S, \Sigma, \varphi, P)$ where

- N is a finite and nonempty set of *nonterminals*,
- $\varphi \colon N \to \mathbb{N}_+$ specifies the *fanout* of a nonterminal,
- Σ is an alphabet of *terminals* such that $N \cap \Sigma = \emptyset$,
- $S \in N$ is the *start symbol* and $\varphi(S) = 1$, and
- P is a finite set of productions p of the form

$$A_0(s_1, \ldots, s_{\varphi(A_0)}) \to A_1(x_1, \ldots, x_{m_1}) \cdots A_n(x_{m_{n-1}+1}, \ldots, x_{m_n}), \tag{1}$$

  where $n \in \mathbb{N}$, $A_0, \ldots, A_n \in N$, $m_i = \sum_{j=1}^{i} \varphi(A_j)$ for every $i \in [n]$,
  and $\langle s_1, \ldots, s_{\varphi(A_0)} \rangle \in \mathcal{F}^{\mathrm{LCFRS}(\Sigma)}_{\varphi(A_1) \cdots \varphi(A_n), \varphi(A_0)}$. The *rank of p*, denoted by $\mathrm{rank}(p)$, is n.

The *derivation relation of G*, denoted by $\Rightarrow_G$, is defined as follows. Let $u, v \in (N \cup \Sigma \cup \{(,), \mathrm{comma}\})^*$, where comma stands for ",". Then $u \Rightarrow_G v$ if there are

- $w, w' \in (N \cup \Sigma \cup \{(,), \mathrm{comma}\})^*$,
- a production $p \in P$ of form (1), and
- $\hat{s}_1, \ldots, \hat{s}_{m_n} \in \mathrm{T}^*_\Sigma$

such that

$$u = w\ A_0(s_1[x_i/\hat{s}_i \mid i \in [m_n]], \ldots, s_{\varphi(A_0)}[x_i/\hat{s}_i \mid i \in [m_n]])\ w' \qquad \text{and}$$
$$v = w\ A_1(\hat{s}_1, \ldots, \hat{s}_{m_1}) \cdots A_n(\hat{s}_{m_{n-1}+1}, \ldots, \hat{s}_{m_n})\ w'.$$

The *language of G* is defined as the set $\mathcal{L}(G) = \{w \in \mathrm{T}^*_\Sigma \mid S(w) \Rightarrow^*_G \varepsilon\}$. ■

Example 3.17. Let $G = (N, S, \Sigma, \varphi, S)$ be an LCFRS, where $N = \{S, A, B, C, D\}$, $\Sigma = \{\mathbf{P}, \mathbf{M}, \mathbf{h}, \mathbf{l}\}$, φ is such that

$$\varphi(S) = \varphi(B) = \varphi(D) = 1 \qquad \text{and} \qquad \varphi(A) = \varphi(C) = 2,$$

and P contains the following productions:

$$\begin{array}{lcl}
S\ ((x_1, x_3, x_2, x_4)) & \to & A\ (x_1, x_2)\ \ C\ (x_3, x_4) \\
A\ ((x_1), (\mathbf{h})) & \to & B\ (x_1) \\
B\ ((\mathbf{P})) & \to & \varepsilon \\
C\ ((x_1), (\mathbf{l})) & \to & D\ (x_1) \\
D\ ((\mathbf{M})) & \to & \varepsilon.
\end{array}$$

Observe that $\langle (x_1, x_3, x_2, x_4) \rangle$ is in $\mathcal{F}^{\mathrm{LCFRS}(\Sigma)}_{2\,2,1}$, $\langle (x_1), (\mathbf{h}) \rangle$ and $\langle (x_1), (\mathbf{l}) \rangle$ are in $\mathcal{F}^{\mathrm{LCFRS}(\Sigma)}_{1,2}$, and $\langle (\mathbf{P}) \rangle$ and $\langle (\mathbf{M}) \rangle$ are in $\mathcal{F}^{\mathrm{LCFRS}(\Sigma)}_{\varepsilon,0}$. A successful derivation of G is

$$\begin{aligned} S((\mathbf{P}, \mathbf{M}, \mathbf{h}, \mathbf{l})) &\Rightarrow_G A((\mathbf{P}), (\mathbf{h}))\ \ C((\mathbf{M}), (\mathbf{l})) \\ &\Rightarrow_G B((\mathbf{P}))\qquad C((\mathbf{M}), (\mathbf{l})) \\ &\Rightarrow_G \qquad\qquad C((\mathbf{M}), (\mathbf{l})) \\ &\Rightarrow_G \qquad\qquad D((\mathbf{M})) \\ &\Rightarrow_G \varepsilon, \end{aligned}$$

where the above productions were applied in the order from top to bottom. Note that the accepted s-term corresponds to the surface form of the upper dependency structure in Figure 3.2 where each word is abbreviated by its first letter. ■

Let $G = (N, S, \Sigma, \varphi, P)$ be an LCFRS. The *fanout of* G, denoted by $\varphi(G)$, is $\max_{A \in N} \varphi(N)$. The *rank of* G is $\max_{p \in P} \mathrm{rank}(p)$. Given a word w of length n the time-complexity of solving $w \in \mathcal{L}(G)$? is in $\mathcal{O}(n^{(\mathtt{rank}(G)+1)\cdot\varphi(G)} \cdot |G|)$ [Sek+91; Gil10]. Note that an LCFRS with fanout 1 and rank 2 is a binary context-free grammar. Its parsing complexity according to the foregoing formula is $\mathcal{O}(n^3 \cdot |G|)$ and coincides with the time-complexity of the [CS70; You67; Kas65]-algorithm.

3.6 Simple definite clause programs

We consider a class of tree grammars called *simple definite clause programs* (sDCPs) [NV14] that generalizes context-free grammars by associating each nonterminal with a fixed number of inherited and synthesized arguments similar as attribute grammars [Knu68]. In its inherited attributes a nonterminal receives input data from its parents and siblings in the derivation tree. Likewise, a nonterminal provides data in its synthesized attributes to its parent and siblings that was computed bottom up using the input data. In sDCPs the domain of both types of attributes are s-terms. In each production the flow and the computation of information is specified with *semantic functions* that are annotated to the context-free nonterminal backbone $A_0 \to A_1 \cdots A_n$. To this end, the inherited attribute of A_0 and the synthesized attributes of $A_1, \ldots, A_n$ are represented by variables. The synthesized arguments of A_0 and the inherited arguments of $A_1, \ldots, A_n$ are supplied with s-terms over a monadic alphabet indexed by the former variables. Similarly as in LCFRSs we impose a single syntactic use requirement on all variables in one production.

Definition 3.18. Let Δ be a monadic alphabet, $n \in \mathbb{N}$, $\iota_0, \ldots, \iota_n \in \mathbb{N}$, $\sigma_0, \ldots, \sigma_n \in \mathbb{N}_+$, and $\tilde{\mathrm{s}} = (\iota_1, \sigma_1) \cdots (\iota_n, \sigma_n), (\iota_0, \sigma_0)$. We define the *set of sDCP functions over* Δ *with signature* $\tilde{\mathrm{s}}$, denoted by $\mathcal{F}^{\mathrm{sDCP}(\Delta)}_{\tilde{\mathrm{s}}}$, as follows.

1. We set the *variable indices of* $\tilde{\mathrm{s}}$ to

$$\mathcal{V}(\tilde{\mathrm{s}}) = \{(0, j) \mid j \in [\iota_0]\} \cup \{(i, j) \mid i \in [n],\ j \in [\sigma_i]\}$$

   and the *s-term indices of* $\tilde{\mathrm{s}}$ to

$$\mathcal{S}(\tilde{\mathrm{s}}) = \{(0, j) \mid j \in [\sigma_0]\} \cup \{(i, j) \mid i \in [n],\ j \in [\iota_i]\}.$$

2. For every $(i, j) \in \mathcal{S}(\tilde{\mathrm{s}})$, we let $s^{(i)}_j \in \mathrm{T}^*_\Delta(X_{\mathcal{V}(\tilde{\mathrm{s}})})$ such that every variable in $X_{\mathcal{V}(\tilde{\mathrm{s}})}$ occurs exactly once in $s^{(0)}_{1,\sigma_0}, s^{(1)}_{1,\iota_1}, \ldots, s^{(n)}_{1,\iota_n}$ together.
3. Then $\langle s^{(0)}_{1,\sigma_0}, s^{(1)}_{1,\iota_1}, \ldots, s^{(n)}_{1,\iota_n} \rangle \in \mathcal{F}^{\mathrm{sDCP}(\Delta)}_{(\iota_1,\sigma_1)\cdots(\iota_n,\sigma_n),(\iota_0,\sigma_0)}$. ■

Example 3.19. Consider the nonterminal backbone $A_0 \to A_1 A_2 A_3$ and let the number of inherited arguments ι_i and synthesized arguments σ_i of A_i for every $i \in [3]_0$ be as follows:

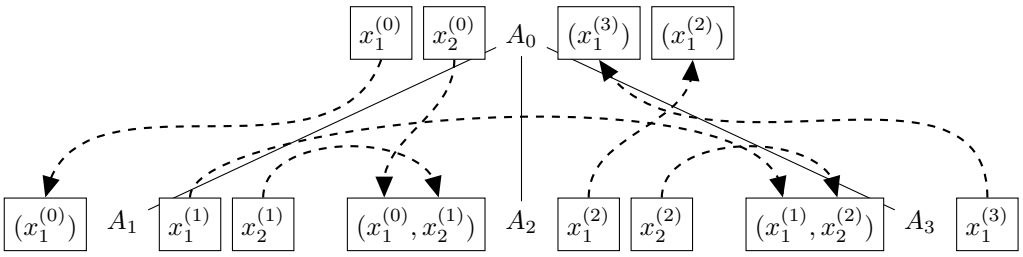


Figure 3.4: Information flow of the sDCP function in Example 3.19.

A_i	A_0	A_1	A_2	A_3
ι_i	2	1	1	1
σ_i	2	2	2	1

The signature that fits this nonterminal-backbone is $\tilde{s} = (1,2)(1,2)(1,1),(2,2)$. For every $(i,j) \in \mathcal{S}(\tilde{s}) = \{(0,1),(0,2),(1,1),(2,1),(3,1)\}$, we have to define an s-term $s_j^{(i)}$, i.e., there is an s-term for every synthesized argument of A_0 and every inherited argument of some A_i where $0 < i \leq 3$. Likewise, for every $(i,j) \in \mathcal{V}(\tilde{s}) = \{(0,1),(0,2),(1,1),(1,2),(2,1),(2,2),(3,1)\}$, i.e., every inherited attribute of A_0 and every synthesized attribute of A_i where $i > 0$, the variable $x_j^{(i)}$ has to occur in exactly one of the above s-terms. An sDCP function in $\mathcal{F}_{\tilde{s}}^{\mathrm{sDCP}(\Sigma)}$ that does not contain any additional terminal symbols is

$$\langle s_1^{(0)}, s_2^{(0)}, s_1^{(1)}, s_1^{(2)}, s_1^{(3)} \rangle = \langle (x_1^{(3)}), (x_1^{(2)}), (x_1^{(0)}), (x_1^{(0)}, x_2^{(1)}), (x_1^{(1)}, x_2^{(2)}) \rangle.$$

It corresponds to the graph in Figure 3.4, where the nonterminal backbone is depicted as a tree. The inherited and synthesized attributes are denoted in boxes to the left and the right of a nonterminal, respectively. The dashed arrows represent the flow of information. ■

Definition 3.20. A *simple definite clause program* (sDCP) is a tuple $G = (N, Z, \Delta, \iota, \sigma, C)$ where

- N is a finite set of nonterminals,
- Δ is a monadic alphabet such that $\Delta \cap N = \emptyset$,
- $\iota\colon N \to \mathbb{N}$ and $\sigma\colon N \to \mathbb{N}_+$ are functions assigning the *number of inherited attributes* and the *number of synthesized attributes*, respectively, to each nonterminal,
- $Z \in N$ such that $\iota(Z) = 0$ and $\sigma(Z) = 1$, and
- C is a finite set of clauses of the form

$$A_0(x_{1,\iota_0}^{(0)}, s_{1,\sigma_0}^{(0)}) \to A_1(s_{1,\iota_1}^{(1)}, x_{1,\sigma_1}^{(1)}) \cdots A_n(s_{1,\iota_n}^{(n)}, x_{1,\sigma_n}^{(n)}) \tag{2}$$

  where $n \in \mathbb{N}$, $A_0, \ldots, A_n \in N$, $\iota_i = \iota(A_i)$ and $\sigma_i = \sigma(A_i)$ for every $i \in [n]_0$, and $\langle s_{1,\sigma_0}^{(0)}, s_{1,\iota_1}^{(1)}, \ldots, s_{1,\iota_n}^{(n)} \rangle \in \mathcal{F}_{\tilde{s}}^{\mathrm{sDCP}(\Delta)}$ for $\tilde{s} = (\iota_1, \sigma_1) \cdots (\iota_n, \sigma_n), (\iota_0, \sigma_0)$.

The *derivation relation of* G, denoted by $\Rightarrow_G$, is defined as follows. Let $u, v \in (N \cup \Delta \cup \{(,), \mathrm{comma}\})^*$. Then $u \Rightarrow_G v$ if there are

- $w, w' \in (N \cup \Delta \cup \{(,), \mathrm{comma}\})^*$,
- a clause $c \in C$ of the form (2), and
- s-terms $t_{1,\iota_0}^{(0)}, t_{1,\sigma_1}^{(1)}, \ldots, t_{1,\sigma_n}^{(n)}$ in T_Δ^*

such that

$$u = w\ A_0(t^{(0)}_{1,\iota_0}, s^{(0)}_{1,\sigma_0}[x^{(i)}_j/t^{(i)}_j \mid (i,j) \in \mathcal{V}(\tilde{s})])\ w' \qquad \text{and}$$
$$v = w\ A_1(s^{(1)}_{1,\iota_1)}[x^{(i)}_j/t^{(i)}_j \mid (i,j) \in \mathcal{V}(\tilde{s})], t^{(1)}_{1,\sigma_1}) \cdots A_n(s^{(n)}_{1,\iota_n}[x^{(i)}_j/t^{(i)}_j \mid (i,j) \in \mathcal{V}(\tilde{s})], t^{(n)}_{1,\sigma_n})\ w'.$$

The *language generated by* G is the set $\mathcal{L}(G) = \{s \in \mathrm{T}^*_\Delta \mid Z(s) \Rightarrow^*_G \varepsilon\}$. ■

Example 3.21. Let $G = (N, S, \Delta, \iota, \sigma, C)$ be an sDCP, where $N = \{S, A, B, C, D\}$, $\Delta = \Delta^{(1)} = \{\mathbf{P}, \mathbf{M}, \mathbf{h}, \mathbf{l}\}$, ι is such that

$$\iota(S) = \iota(B) = \iota(C) = \iota(D) = 0 \qquad \text{and} \qquad \iota(A) = 1,$$

σ is such that

$$\sigma(S) = \sigma(A) = \sigma(B) = \sigma(C) = \sigma(D) = 1,$$

and C contains the following clauses:

$$\begin{aligned}
S\big((x^{(1)}_1)\big) &\to A((x^{(2)}_1), x^{(1)}_1)\ C(x^{(2)}_1) \\
A\big(x^{(0)}_1, (\mathbf{h}((x^{(1)}_1, x^{(0)}_1)))\big) &\to B(x^{(1)}_1) \\
B\big((\mathbf{P}(()))\big) &\to \varepsilon \\
C\big((\mathbf{l}((x^{(1)}_1)))\big) &\to D(x^{(1)}_1) \\
D\big((\mathbf{M}(()))\big) &\to \varepsilon.
\end{aligned}$$

We depicted the information flow of the first, second and fourth clause in Figure 3.5, where for ease of readability s-terms are visualized as graphs. Below we give a successful derivation of G, where the above clauses were applied in the order from top to bottom. Observe how the assignment for the variable $x^{(2)}_1$ is guessed in the first step. Later it is checked that this guess was correct.

$$\begin{aligned}
S\left(\left(\begin{array}{ccc} & \mathbf{h} & \\ \mathbf{P} & & \mathbf{l} \\ & & \mathbf{M} \end{array}\right)\right) &\Rightarrow_G A\left(\left(\begin{array}{c} \mathbf{l} \\ \mathbf{M} \end{array}\right), \left(\begin{array}{ccc} & \mathbf{h} & \\ \mathbf{P} & & \mathbf{l} \\ & & \mathbf{M} \end{array}\right)\right)\ C\left(\left(\begin{array}{c} \mathbf{l} \\ \mathbf{M} \end{array}\right)\right) \\
&\Rightarrow_G \quad B\left(\left(\mathbf{P}(())\right)\right) \qquad\qquad C\left(\left(\begin{array}{c} \mathbf{l} \\ \mathbf{M} \end{array}\right)\right) \\
&\Rightarrow_G \qquad\qquad\qquad\quad C\left(\left(\begin{array}{c} \mathbf{l} \\ \mathbf{M} \end{array}\right)\right) \\
&\Rightarrow_G \qquad\qquad\qquad\quad D\big((\mathbf{M}(()))\big) \\
&\Rightarrow_G \qquad \varepsilon
\end{aligned}$$

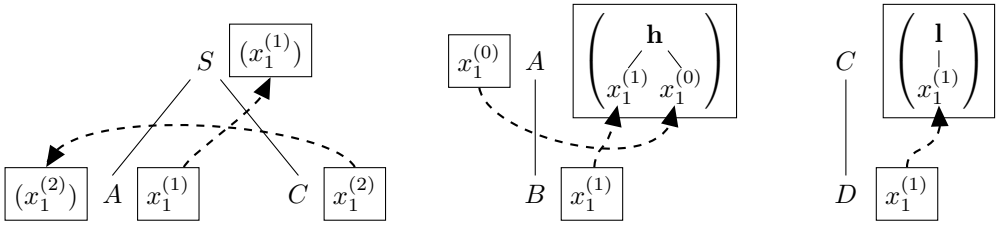


Figure 3.5: Information flow in three of the sDCP clauses of Example 3.21.

Note that G accepts the s-term t of the upper dependency structure $h = (t, \leq_t)$ depicted in Figure 3.2 where each word is abbreviated by its first letter. ■

The main difference between sDCPs and attribute grammars with s-terms as value domain is that in an attribute grammar there is a unique set of semantic functions for each nonterminal backbone $A \to A_1 \cdots A_n$, whereas in sDCPs there can be several clauses with the same the nonterminal backbone but different sDCP functions.

Note also that each LCFRS $(N, S, \Delta, \varphi, P)$ generates the same language as the sDCP $(N, S, \Delta, \iota, \varphi, C)$ where we have $\iota(A) = 0$ for every $A \in N$. Furthermore, C is obtained from P by appropriate renaming of the variables in each production of form (1): for every $j \in [n]$ and $i \in \{m_{j-1}+1, \ldots, m_j\}$, the variable x_i is replaced by $x^{(j)}_{i-m_{j-1}}$.

3.7 Hybrid grammars

We will combine an LCFRS and an sDCP to a hybrid grammar that generates hybrid trees over (Σ, Γ). As mentioned earlier, the sDCP generates the s-terms t and the LCFRS accounts for the linear order $\leq_t$ on $\mathrm{pos}_\Sigma(t)$. Obviously, an LCFRS cannot directly generate a string over positions of some s-term. Instead, we annotate indices to the symbols in Σ where the same index at two terminal occurrences is regarded as a semantic link. Formally, we introduce the notion of an *indexed-extension* of ranked sets.

Definition 3.22. Let Ω be a ranked alphabet and A a set. We define the *A-indexed extension of Ω* to be the ranked set $\mathcal{I}(A, \Omega) = \{\omega^{\boxed{a}} \mid a \in A, \omega \in \Omega\}$ where $\mathrm{rk}(\omega^{\boxed{a}}) = \mathrm{rk}(\omega)$.

Let Γ be a ranked alphabet disjoint from $\mathcal{I}(A, \Omega)$. We define $\mathcal{I}^*_{A,\Omega,\Gamma}(X)$ such that, for every $s \in \mathrm{T}^*_{\mathcal{I}(A,\Omega)\cup\Gamma}(X)$, it holds that $s \in \mathcal{I}^*_{A,\Omega,\Gamma}(X)$ if every index $a \in A$ occurs at most once in s.

Let A and B be sets of indices. A *reindexing function* $\mathcal{R}$ extends some injective mapping $r\colon A \to B$ to $\mathcal{I}(A, \Omega) \to \mathcal{I}(B, \Omega)$ such that, for every $a \in A$ and $w \in \Omega$, we have that $w^{\boxed{a}} \mapsto w^{\boxed{r(a)}}$. If A is finite, $|A| = n$, $A = \{a_1, \ldots, a_n\}$, and, for every $i \in [n]$, it holds that $r(a_i) = b_i$, then we denote $\mathcal{R}$ also by $\{a_1 \mapsto b_1, \ldots, a_n \mapsto b_n\}$ and say that $\mathcal{R}$ is a *finite reindexing*. The *deindexing function* $\mathcal{D}\colon \mathcal{I}(A, \Omega) \to A\colon w^{\boxed{a}} \mapsto w$ removes indices. We lift $\mathcal{R}$ and $\mathcal{D}$ to s-terms, lists of s-terms, LCFRS productions, and sDCP productions in the obvious way, where $\mathcal{R}$ relabels only those symbols in $\mathcal{I}(A, \Omega)$ and $\mathcal{D}$ removes the indices from any indexed sub-alphabet the s-term was build over. ■

We can regard a hybrid tree $h = (\xi, \leq_\xi)$ over (Σ, Γ) also as a pair $[s, t]$ of s-terms over adequate indexed-extensions of Σ and Γ.

Definition 3.23. Let $(\Gamma, \mathrm{rk}_\Gamma)$ be a monadic alphabet, $(\Sigma, \mathrm{rk}_\Sigma)$ an alphabet such that $\Sigma \subseteq \Gamma$, $\Delta = \Gamma \setminus \Sigma$, and $n \in \mathbb{N}$. Let $s \in \mathcal{I}^*_{[n],\Sigma,\emptyset}$ and $t \in \mathcal{I}^*_{[n],(\Sigma,\mathrm{rk}_\Gamma),(\Delta,\mathrm{rk}_\Gamma)}$ such that

- $|\mathrm{pos}(s)| = n = |\mathrm{pos}_\Sigma(t)|$,
- for every $p \in \mathrm{pos}(s)$, there exist $i \in \mathbb{N}_+$ and $\sigma \in \Sigma$ such that $s(p) = \sigma^{\boxed{i}}$ and $p = i \cdot \varepsilon$,[1] and
- for every $\sigma \in \Sigma$ and $i \in [n]$, it holds that $\sigma^{\boxed{i}}$ occurs in s iff $\sigma^{\boxed{i}}$ occurs in t.

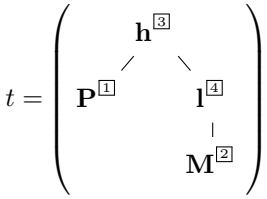
Then $[s, t]$ is called *canonically indexed over* (Σ, Γ). There is a one-to-one correspondence between the canonically indexed pairs of terms over (Σ, Γ) and the hybrid trees over (Σ, Γ). A pair $[s, t]$ *corresponds* to the hybrid tree h constructed as follows. We define the total order $\leq_t$ on $\mathrm{pos}_{\mathcal{I}(\mathbb{N}_+,\Sigma)}(t)$ such that for every $u_1, u_2 \in \mathrm{pos}_{\mathcal{I}(\mathbb{N}_+,\Sigma)}(t)$ it holds that

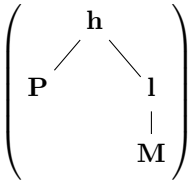
$$u_1 \leq_t u_2 \text{ iff } \exists \sigma_1, \sigma_2 \in \Sigma\colon \exists i_1, i_2 \in [n]\colon t(u_1) = \sigma_1^{\boxed{i_1}},\ t(u_2) = \sigma_2^{\boxed{i_2}},\ \text{and } i_1 \leq i_2.$$

Then $h = (\mathcal{D}(t), \leq_t)$. ■

[1] The appending of ε to i expresses a type conversion from $\mathbb{N}_+$ to ${\mathbb{N}_+}^*$.

Example 3.24. Below we depict s-terms s and t and a hybrid tree $h = (t', \leq_{t'})$ where $[s, t]$ is canonically indexed and corresponds to h.

$t =$ 

$t' =$ 

$s = (\mathbf{P}^{\boxed{1}}, \mathbf{M}^{\boxed{2}}, \mathbf{h}^{\boxed{3}}, \mathbf{l}^{\boxed{4}})$ $\quad 11 <_{t'} 121 <_{t'} 1 <_{t'} 12$ ■

Now we combine an LCFRS and an sDCP to a hybrid grammar which accepts canonically indexed pairs of s-terms over (Σ, Γ). Thus, we can use a hybrid grammar to specify a language of hybrid trees over (Σ, Γ). An LCFRS/sDCP-hybrid grammar synchronizes the derivation of an LCFRS and an sDCP by coupling an LCFRS production with an sDCP clause to a hybrid production. This coupling is twofold: firstly, there is a one-to-one correspondence between nonterminals in the LCFRS component and those in the sDCP component of a hybrid production. Secondly, each terminal in the LCFRS component of a hybrid production is indexed and there must be a unique corresponding occurrence of the same symbol in the sDCP component of the hybrid production.

Since both grammar devices only consume terminals during a derivation, we do not need to introduce additional indices for terminals once the derivation has started. However, we need to reindex the indexed terminals in a hybrid production to make it applicable to a sentential form.

The nonterminals in the derivation of a hybrid grammar are indexed by element from $\mathbb{N}_+^*$ where the start symbol is indexed by ε. In each production suffixes are annotated at the nonterminals where the left-hand side nonterminal has the suffix ε and the right-hand side nonterminals have suffixes 1 to n in ascending order. Consequently, we impose the restriction that the i-th nonterminal on the right-hand side of the LCFRS component is synchronized with the i-th nonterminal on the right-hand side of the sDCP component.[2] If a production is applied, then the new nonterminals inherit the prefix of the replaced one and append their respective suffix. Thus, in a derivation every nonterminal is indexed by its Gorn address in the derivation tree.

Definition 3.25. An *LCFRS/sDCP-hybrid grammar* is a triple $H = ((N_1, S_1, \Sigma, \varphi), (N_2, S_2, \Gamma, \iota, \sigma), P)$ where

- $(N_1, S_1, \Sigma, \varphi, \emptyset)$ is an LCFRS,
- $(N_2, S_2, \Gamma, \iota, \sigma, \emptyset)$ is an sDCP,
- $\Sigma \subseteq \Gamma$ and $\Delta = \Gamma \setminus \Sigma$, and
- P is a finite set of *hybrid productions* $[r_1, r_2]$ of the form

$$\begin{aligned}&\big[\ A_0^{\boxed{\varepsilon}}(s_{1,k_0}) \to A_1^{\boxed{1}}(x_{1,m_1}) \cdots A_n^{\boxed{n}}(x_{m_{n-1}+1,m_n}) \\ &\ , B_0^{\boxed{\varepsilon}}(x_{1,\iota_0}^{(0)}, \xi_{1,\sigma_0}^{(0)}) \to B_1^{\boxed{1}}(\xi_{1,\iota_1}^{(1)}, x_{1,\sigma_1}^{(1)}) \cdots B_n^{\boxed{n}}(\xi_{1,\iota_n}^{(n)}, x_{1,\sigma_n}^{(n)})\ \big] \end{aligned} \tag{3}$$

  where $n \in \mathbb{N}$ and, for every $i \in [n]_0$, it holds that $A_i \in N_1$, $B_i \in N_2$, $k_i = \varphi(A_i)$, $\iota_i = \iota(B_i)$, and $\sigma_i = \sigma(B_i)$. Also, there exists $q \in \mathbb{N}$ such that $s_{1,k_0} \in \mathcal{I}^*_{[q],\Sigma,\emptyset}(X)$, $m_i = \sum_{j=1}^{i} k_j$ for every $i \in [n]$, and $\xi_{1,\sigma_1}^{(0)}, \xi_{1,\iota_1}^{(1)}, \ldots, \xi_{1,\iota_n}^{(n)} \in \mathcal{I}^*_{[q],(\Sigma,\mathrm{rk}_\Gamma),(\Delta,\mathrm{rk}_\Gamma)}(X)$. Further, for every $p \in [q]$, there is exactly one symbol $\sigma \in \Sigma$ such that $\sigma^{\boxed{p}}$ occurs in the production.[3] We require that $\sigma^{\boxed{p}}$ occurs exactly once in r_1 and exactly once in r_2.

[2] It can be seen that this syntactic restriction does not affect the expressiveness of LCFRS/sDCP-hybrid grammars.

[3] For labeled dependency parsing we will loosen this restriction and allow for different symbols in the LCFRS component and sDCP component of a production to be synchronized.

We demand that $H_1 = (N_1, S_1, \Sigma, \varphi, \mathcal{D}(\pi_1(P)))$ is an LCFRS and that $H_2 = (N_2, S_2, \Gamma, \iota, \sigma, \mathcal{D}(\pi_2(P)))$ is an sDCP.[4] We refer to H_1 as *string component* and to H_2 as *tree component* of H.

Given a hybrid production $[r_1, r_2]$ of form (3), $w \in {\mathbb{N}_+}^*$, and a reindexing function $\mathcal{R}\colon \mathcal{I}([q], \Sigma) \to \mathcal{I}([p], \Sigma)$, where $q \in \mathbb{N}$, we define the *derivation relation of H given $[r_1, r_2]$, w, and $\mathcal{R}$*, denoted by $\xRightarrow{\mathcal{R}[r_1, r_2],\, w}_H$, as follows. Let $u_1, v_1 \in (\mathcal{I}({\mathbb{N}_+}^*, N_1) \cup \mathcal{I}([p], \Sigma) \cup \{(,), \mathrm{comma}\})^*$ and $u_2, v_2 \in (\mathcal{I}({\mathbb{N}_+}^*, N_2) \cup \mathcal{I}([p], \Sigma) \cup \Delta \cup \{(,), \mathrm{comma}\})^*$ where comma stands for ,. Then

$$[u_1, v_1] \xRightarrow{\mathcal{R}[r_1, r_2],\, w}_H [u_2, v_2]$$

if there are

- $w_1, w_1' \in (\mathcal{I}({\mathbb{N}_+}^*, N_1) \cup \mathcal{I}([p], \Sigma) \cup \{(,), \mathrm{comma}\})^*$,
- $w_2, w_2' \in (\mathcal{I}({\mathbb{N}_+}^*, N_2) \cup \mathcal{I}([p], \Sigma) \cup \Delta \cup \{(,), \mathrm{comma}\})^*$,
- $\hat{s}_1, \ldots, \hat{s}_{m_n} \in \mathcal{I}^*_{[p], \Sigma, \emptyset}$, and
- s-terms $t^{(0)}_{1,\iota_0}, t^{(1)}_{1,\sigma_1}, \ldots, t^{(n)}_{1,\sigma_n}$ in $\mathcal{I}^*_{[p], (\Sigma, \mathrm{rk}_\Gamma), (\Delta, \mathrm{rk}_\Gamma)}$

such that

$$\begin{aligned} u_1 &= w_1\ A_0^{\boxed{w}}(\mathcal{R}(s_1)[x_i/\hat{s}_i \mid i \in [m_n]], \ldots, \mathcal{R}(s_{\varphi(A_0)})[x_i/\hat{s}_i \mid i \in [m_n]])\ w_1' \\ u_2 &= w_1\ A_1^{\boxed{w1}}(\hat{s}_1, \ldots, \hat{s}_{m_1}) \cdots A_n^{\boxed{wn}}(\hat{s}_{m_{n-1}+1}, \ldots, \hat{s}_{m_n})\ w_1' \end{aligned}$$

and, abbreviating $((\iota_1, \sigma_1) \cdots (\iota_n, \sigma_n), (\iota_0, \sigma_0))$ by $\tilde{\mathrm{s}}$,

$$\begin{aligned} v_1 &= w_2\ B_0^{\boxed{w}}(t^{(0)}_{1,\iota_0}, \mathcal{R}(\xi^{(0)}_{1,\sigma_0})[x_j^{(i)}/t_j^{(i)} \mid (i,j) \in \mathcal{V}(\tilde{\mathrm{s}})])\ w_2' \\ v_2 &= w_2\ B_1^{\boxed{w1}}(\mathcal{R}(\xi^{(1)}_{1,\iota_1})[x_j^{(i)}/t_j^{(i)} \mid (i,j) \in \mathcal{V}(\tilde{\mathrm{s}})], t^{(1)}_{1,\sigma_1}) \\ &\quad \cdots B_n^{\boxed{wn}}(\mathcal{R}(\xi^{(n)}_{1,\iota_n})[x_j^{(i)}/t_j^{(i)} \mid (i,j) \in \mathcal{V}(\tilde{\mathrm{s}})], t^{(n)}_{1,\sigma_n})\ w_2'. \end{aligned}$$

We set

$$\Longrightarrow_H\ = \bigcup_{[r_1,r_2] \in P,\ w \in {\mathbb{N}_+}^*,\ \mathcal{R} \text{ reindexing function}} \xRightarrow{\mathcal{R}[r_1, r_2],\, w}_H$$

and define the *hybrid language generated by H* to be

$$[H] = \{[s,t] \text{ canonically indexed over } (\Sigma, \Gamma) \mid [S_1^{\boxed{\varepsilon}}(s), S_2^{\boxed{\varepsilon}}(t)] \Longrightarrow_H^* [\varepsilon, \varepsilon]\}.$$

The set of *hybrid trees generated by H*, denoted by $\mathcal{L}(H)$, is the set of all hybrid trees over (Σ, Γ) that correspond to some $[s,t] \in [H]$. We define $P_\mathcal{R} = P_\mathcal{R}{}^{(1)} = \{([r, r'], \mathcal{R}') \mid [r, r'] \in P,\ \mathcal{R}' \text{ finite reindexing}\}$ to be a monadic indexed set. Let h be a hybrid tree over (Σ, Γ). We define the set of *derivation trees of h*, denoted by $D_H(h)$, to be the smallest subset of $\mathrm{T}_{P_\mathcal{R}}$ satisfying the following condition. Let $[s,t] \in [H]$ correspond to h and let $n \in \mathbb{N}_+$, $\mathcal{R}_1, \ldots, \mathcal{R}_n$ reindexing functions, $w_1, \ldots, w_n \in {\mathbb{N}_+}^*$, and $[r_1, r_1'], \ldots, [r_n, r_n'] \in P$ such that

$$[S_1^{\boxed{\varepsilon}}(s), S_2^{\boxed{\varepsilon}}(t)] \xRightarrow{\mathcal{R}_1[r_1, r_1'],\, w_1}_H \cdots \xRightarrow{\mathcal{R}_n[r_n, r_n'],\, w_n}_H [\varepsilon, \varepsilon].$$

Then $\xi \in D_H(h)$, if $\xi \in \mathrm{T}_{P_\mathcal{R}}$, $\mathrm{pos}(\xi) = \{w_1, \ldots, w_n\}$, and $\xi(w_i) = ([r_i, r_i'], \mathcal{R}_i)$ for every $i \in [n]$. ■

[4] Thus, we impose restrictions on the variable domain and the single syntactic use requirement of LCFRS productions and sDCP productions also for a hybrid production.

Example 3.26. Let $G = ((N, S, \Sigma, \varphi), (N, S, \Gamma, \iota, \sigma), P)$ be an LCFRS/sDCP-hybrid grammar, where $N = \{S, A, B, C, D\}$, σ, ι, and σ are such that

x	S	A	B	C	D
$\varphi(x)$	1	2	1	2	1
$\iota(x)$	0	1	0	0	0
$\sigma(x)$	1	1	1	1	1

and P contains the hybrid productions depicted in Figure 3.6. Observe that the restriction of G to the first and second component results in the grammars given in Example 3.17 and Example 3.21. Note also that in a hybrid production the same index can occur at a pair of terminals and a pair of nonterminals. Still, by the definition of the derivation semantics of G this does not imply a semantic link between a terminal and a nonterminal. A derivation of G and the corresponding derivation tree is depicted in Figure 3.7. ■

In order to represent the ambiguity of natural language we also define a probabilistic version of LCFRS/sDCP-hybrid grammars.

Definition 3.27. A *probabilistic LCFRS/sDCP-hybrid grammar* is given by a quadruple

$$H = ((N_1, S_1, \Sigma, \varphi), (N_2, S_2, \Gamma, \iota, \sigma), P, p)$$

such that $H' = ((N_1, S_1, \Sigma, \varphi), (N_2, S_2, \Gamma, \iota, \sigma), P)$ is a hybrid grammar and $p\colon P \to [0, 1]$. We require that H is *proper*, i.e., for every $(A, B) \in N_1 \times N_2$ where there exist $[r_1, r_2] \in P$ and $w \in \mathbb{N}_+^*$ such that $A^{\boxed{w}}$ occurs in r_1 and $B^{\boxed{w}}$ occurs in r_2, it holds that

$$1 = \sum_{[r_1, r_2] \in P\colon A^{\boxed{\varepsilon}} \text{ occurs in } r_1,\ B^{\boxed{\varepsilon}} \text{ occurs in } r_2} p([r_1, r_2]).$$

Let h be a hybrid tree over (Σ, Γ). The *probability of h w.r.t. H* is

$$[\![H]\!](h) = \max_{\xi \in D_{H'}(h)} \prod_{u \in \mathrm{pos}(\xi)} p(\pi_1(\xi(u))).$$

■

Strictly speaking, one should not call H probabilistic but weighted with the Viterbi semiring, since in general it does not induce a probability distribution over $\mathbf{HT}(\Sigma, \Gamma)$. We do not require H to be *consistent*, i.e., the weights of all derivation trees do not need to sum up to one. Moreover, we use the weight of the best derivation tree of some hybrid tree h as an approximation for the sum of all derivation trees of h.

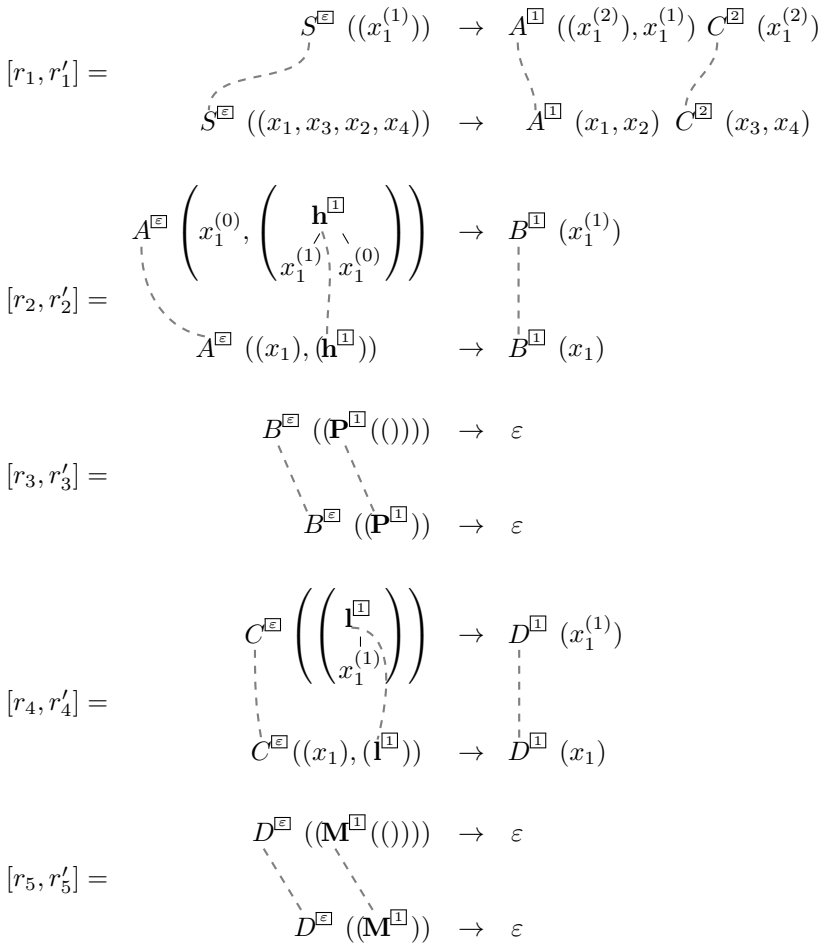


Figure 3.6: Five LCFRS/sDCP-hybrid productions where the string component of each production is depicted at the bottom and the tree component of each production is depicted at the top. The dashed lines additionally highlight linked nonterminals and terminals.

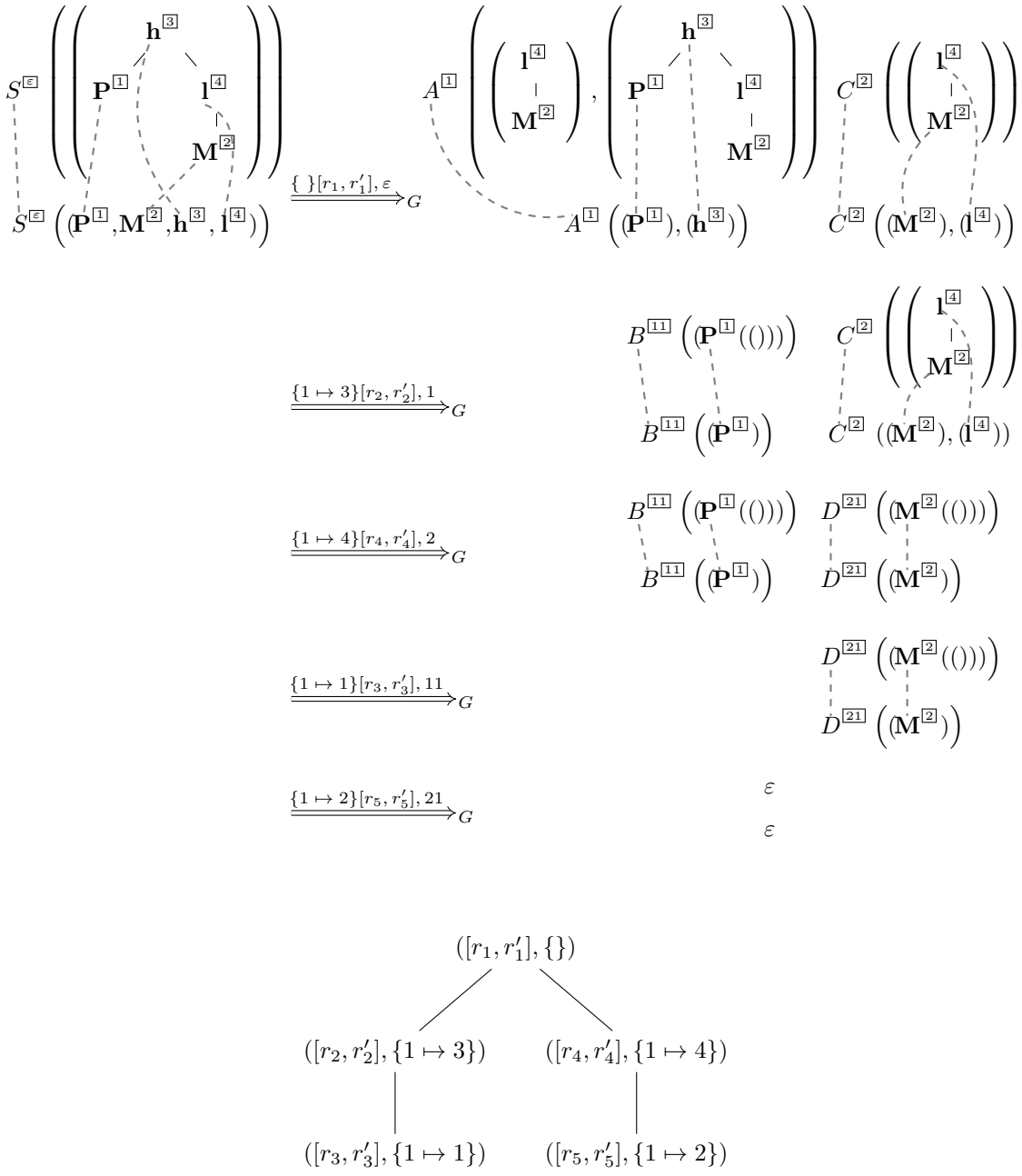


Figure 3.7: A derivation of the LCFRS/sDCP-hybrid grammar in Example 3.26 and the corresponding derivation tree.

4 Induction of hybrid grammars

In this chapter we describe how a probabilistic LCFRS/sDCP-hybrid grammars can be induced from a corpus of constituent structures or a corpus of (labeled) dependency structures. In each scenario first a hybrid grammar $G_{h,\pi}$ is induced from a single hybrid tree h such that $\mathcal{L}(G_{h,\pi}) = \{h\}$. To construct $G_{h,\pi}$ we partition the hybrid tree h according to a *recursive partitioning* π. For each resulting part, we will define a word-tuple function and an sDCP function that generate this part and combine both functions to a hybrid production. The hybrid productions of $G_{h,\pi}$ will license one derivation that disassembles the hybrid tree in exactly these parts. Afterwards, we discuss methods to combine the hybrid grammars induced for each hybrid tree of some corpus to a single probabilistic LCFRS/sDCP-hybrid grammar.

4.1 Recursive partitionings

In this section we present a method to partition a hybrid tree $h = (s, \leq_s) \in \mathbf{HT}(\Sigma, \Gamma)$ by recursively dividing $\mathrm{pos}_\Sigma(s)$. The resulting *recursive partitioning of* $\mathrm{pos}_\Sigma(s)$ is a term π over $\mathfrak{P}(\mathrm{pos}_\Sigma(s))$. Later π will also determine the fanout and the rank of the LCFRS component of the induced hybrid grammar. Since both parameters affect the parsing complexity of the LCFRS, we present a method to transform π in order to adjust these parameters.

Definition 4.1. Let A be a finite set. We define the ranked alphabet $\Sigma = \Sigma^{(1)} = \mathfrak{P}(A)$. A term $\pi \in \mathrm{T}_\Sigma$ is a *recursive partitioning of* A, if

i) $\pi(\varepsilon) = A$,

ii) for every $p \in \mathrm{pos}(\pi)$, it holds that
   a) if $|\pi(p)| \leq 1$, then $p1 \notin \mathrm{pos}(\pi)$, and
   b) if $|\pi(p)| > 1$, then there exists $n \geq 2$ such that
      - $pn \in \mathrm{pos}(\pi)$ and $p(n+1) \notin \mathrm{pos}(\pi)$,
      - for every $i \in [n]$, it holds that $\pi(pi) \neq \emptyset$,
      - $\pi(p) = \bigcup_{i=1}^{n} \pi(pi)$, and
      - for every $i, j \in [n]$ where $i \neq j$, it holds that $\pi(pi) \cap \pi(pj) = \emptyset$. ■

For every finite, totally ordered set we define two simple recursive partitionings.

Definition 4.2. Let $n \in \mathbb{N}$ and $A = \{a_1, \ldots, a_n\}$ be a totally ordered set such that $a_1 < \cdots < a_n$. The *left-branching* recursive partitioning of A is the recursive partitioning π of A such that, for every $p \in \mathrm{pos}(\pi)$, there exists $i \in [n]$ such that

- $\pi(p) = \{a_i\}$, or
- $\pi(p) = \{a_1, \ldots, a_i\}$, $\pi(p1) = \{a_1, \ldots, a_{i-1}\}$, $\pi(p2) = \{a_i\}$, and $p3 \notin \mathrm{pos}(\pi)$.

Likewise, the *right-branching* recursive partitioning of A is the recursive partitioning π of A where, for every $p \in \mathrm{pos}(\pi)$, there exists $i \in [n]$ such that

- $\pi(p) = \{a_i\}$, or
- $\pi(p) = \{a_i, \ldots, a_n\}$, $\pi(p1) = \{a_{i+1}, \ldots, a_n\}$, $\pi(p2) = \{a_i\}$, and $p3 \notin \mathrm{pos}(\pi)$. ■

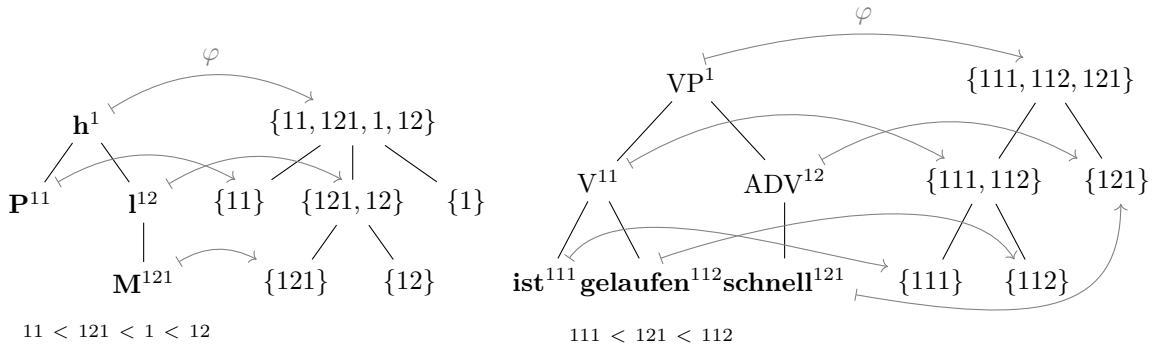
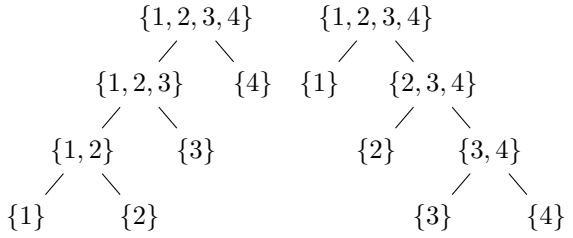


Figure 4.1: The recursive partitioning directly extracted from a dependency structure and from a constituent structure, respectively.

Example 4.3. Let $A = [4]$ be ordered with the usual order $<$ on $\mathbb{N}$. The left-branching and right-branching recursive partitioning of A are depicted below:



■

We may directly extract a recursive partitioning of $\mathrm{pos}_\Sigma(s)$ from a hybrid tree $h = (s, \leq_s) \in \mathbf{HT}(\Sigma, \Gamma)$.

Definition 4.4. Let $h = (s, \leq_s) \in \mathbf{HT}(\Sigma, \Gamma)$. The *recursive partitioning directly extracted from* h is the smallest recursive partitioning π of $\mathrm{pos}_\Sigma(s)$ (small w.r.t. the cardinality of $\mathrm{pos}(\pi)$) satisfying the following condition. For every $p \in \mathrm{pos}(s)$, let $U(p) = \{q \in \mathrm{pos}_\Sigma(s) \mid p \chi^* q\}$. We require a mapping $\varphi\colon \mathrm{pos}(s) \to \mathrm{pos}(\pi)$ such that, for each $p \in \mathrm{pos}(s)$ where $U(p) \neq \emptyset$, it holds that $\varphi(p) = U(p)$. Further, if $p, q \in \mathrm{pos}(\pi)$, then $\min_{<_s}(\pi(p)) <_s \min_{<_s}(\pi(q))$ implies $p <_\ell q$. ■

The above definition is sound. We construct π such that it contains

- $\mathrm{pos}_\Sigma(s)$ as root,
- the node $U(p)$ for every $p \in \mathrm{pos}(s)$ such that $U(p) \neq \emptyset$, and
- the node $\{q\}$ for every $q \in \mathrm{pos}_\Sigma(s)$.

Observe that these nodes can be arranged to meet the requirements of a recursive partitioning, and that the last sentence of Definition 4.4 specifies the order of siblings uniquely. None of the above nodes can be left out, i.e., π is the smallest recursive partitioning that satisfies these conditions. We depicted the recursive partitionings directly extracted from a dependency structure and from a constituent structure in Figure 4.1. In the former case we need the nodes $\{1\}$ and $\{12\}$ in addition to those contained in the image of φ. In the latter case every node in π is reached by φ. Observe the structural similarities between s and π, if π is restricted to the image of φ.

Transformation of recursive partitionings

We conclude this section with presenting a method for the transformation of a recursive partitioning of a finite, totally ordered set. Let A be a set, $<$ a total order on A, and π a recursive partitioning of

A. Also, let J be a node of π, i.e., a subset of A. We can determine a sequence over elements from J that is *consecutive* in the sense that there is no element in A that can be inserted at an intermediate position. We also want this sequence to be *maximal*, i.e., it cannot be extended with an element from J. There is a finite number k of maximal consecutive sequences of J. The transformation of the recursive partitioning shall alter π into π' such that, for each node J in π', we have that the number of maximal consecutive sequences of J is below some target value ℓ. The method shall be such that π' is binarized but still structurally similar to π in a certain sense.

Definition 4.5. Let A be a finite set, $J \subseteq A$, and $<$ a total order on A. Furthermore, let $k \in \mathbb{N}_+$ and $b_1, \ldots, b_k \in J$. We say that $b_1 \cdots b_k$ is a *maximal $<$-consecutive sequence of J*, if

- $b_1 < b_2 < \cdots < b_k$,
- for every $i \in [k-1]$, there does not exist $a \in A$ such that $b_i < a < b_{i+1}$,
- for every $b \in J$ such that $b < b_1$, there exists $a \in A$ such that $b < a < b_1$, and
- for every $b \in J$ such that $b_k < b$, there exists $a \in A$ such that $b_k < a < b$.

We denote the set of all maximal $<$-consecutive sequences of J by $\mathrm{mseq}_<(J)$. ■

Next, we construct a vector that contains all maximal $<$-consecutive sequences of J in a canonical order.

Definition 4.6. Let A be a finite set and $<$ a total order on A. For each $J \subseteq A$, we define the *vector of maximal $<$-consecutive sequences of J*, denoted by $\mathrm{vmseq}_<(J)$, as

$$\langle b_1^{(1)} \cdots b_{k_1}^{(1)}, \ldots, b_1^{(n)} \cdots b_{k_n}^{(n)} \rangle,$$

where $n = |\,\mathrm{mseq}_<(J)|$, $b_1^{(i)} \cdots b_{k_i}^{(i)} \in \mathrm{mseq}_<(J)$ for every $i \in [n]$, and $b_{k_i}^{(i)} < b_1^{(i+1)}$ holds for every $i \in [n-1]$. ■

Example 4.7. Consider the set $[7]$, the set $J = \{1, 2, 5, 7\}$, and the usual order $<$ on the integers. Then $\mathrm{mseq}_<(J) = \{5, 1\,2, 7\}$ and $\mathrm{vmseq}_<(J) = \langle 1\,2, 5, 7 \rangle$. ■

Now we provide an algorithm for the transformation of a recursive partitioning.

$\texttt{transform}(\ell, B((B_1(s_1), \ldots, B_n(s_n))))$ $\qquad$ $[\texttt{assertion}\ |\,\mathrm{mseq}_<(B)| \leq \ell]$

1. `if` $n = 0$ `then return` $B(())$
2. for every $i \in [n]$, let $t_i = B_i(s_i)$.
3. choose a sub-term $C(s')$ of $(t_1, \ldots, t_n)$ such that $|\,\mathrm{mseq}_<(C)| \leq \ell$ and $|\,\mathrm{mseq}_<(B \setminus C)| \leq \ell$
4. let $(t') = \texttt{filter}(C, B((t_1, \ldots, t_n)))$
5. `return` $B((\texttt{transform}(\ell, C(s')), \texttt{transform}(\ell, t')))$

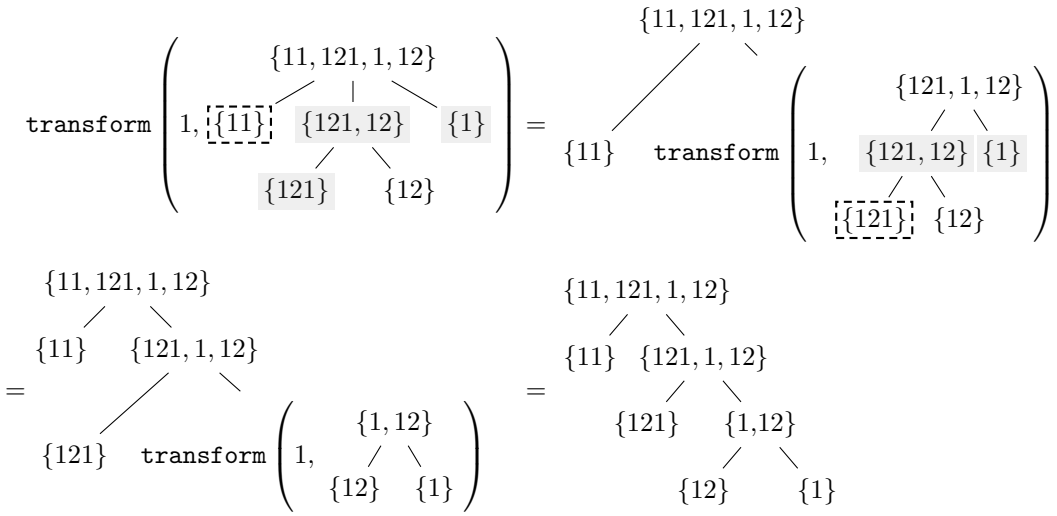
where

$$\texttt{filter}(C, B((t_1, \ldots, t_n)) = \begin{cases} (B((t_1, \ldots, t_n))) & \text{if } B \cap C = \emptyset, \\ ((B \setminus C)(\texttt{filter}(C, t_1) \diamond \cdots \diamond \texttt{filter}(C, t_n))) & \text{if } |B \setminus C| > 1, \\ ((B \setminus C)()) & \text{if } |B \setminus C| = 1, \text{ and} \\ () & \text{if } |B \setminus C| = 0. \end{cases}$$

Note that, for every recursive partitioning π, it holds that $|\,\mathrm{mseq}_<(\pi(\varepsilon))| \leq 1$. Thus, the assertion for the initial call of `transform` is fulfilled. Pertaining to 3, observe that one can always choose C to be the leaf that just contains the smallest or largest element of some sequence in $\mathrm{mseq}_<(B)$. Such a node must

exist since π is a recursive partitioning. Often, there is some degree of freedom in the choice of C which influences how the resulting recursive partitioning differs from π. For instance, one can use a strategy that selects the largest or smallest set possible or perform a simple breath-first or depth-first search on $B((t_1, \ldots, t_n))$. For our implementation (cf. Chapter 5) we use breath-first search where nodes are visited from right to left at each level. This favors left-branching structures which turned out to be beneficial, cf. Section 5.5.

Example 4.8. We depict the transformation of the recursive partitioning directly extracted from the dependency structure in Figure 4.1 with threshold $\ell = 1$. We highlight each node in gray whose choice would violate the condition imposed in step 3 of the algorithm. The chosen sub-term $C(s')$ is indicated by a dashed box.



Incidentally, the result is the right-branching recursive of $\mathrm{pos}_\Sigma(s)$. ■

4.2 Induction of a hybrid grammar from a single hybrid tree

Throughout this section let $h = (s, \leq_s)$ be a hybrid tree over (Σ, Γ) and π a recursive partitioning of $\mathrm{pos}_\Sigma(s)$. We induce an LCFRS/sDCP-hybrid grammar $G_{h,\pi}$ whose language is $\{h\}$. Each node of π is a nonterminal of $G_{h,\pi}$. Moreover, π will be isomorphic to the only derivation tree of $G_{h,\pi}$ for h. Thus, each node J in π and its children $J_1, \ldots, J_n$ constitute the nonterminal backbone of some hybrid production, where J is the nonterminal on the left-hand side and $J_1, \ldots, J_n$ occur on the right-hand side in this order. We have the same nonterminals in the string grammar and the tree grammar and also synchronize only the same nonterminals. We dedicate single subsections to the construction of word-tuple functions, which is the same in the constituent and the dependency scenario, and the construction of sDCP functions, where both scenarios differ. We specify the values of the functions fanout (φ), number of inherited arguments (ι), and number of synthesized arguments (σ) for the nonterminals J and $J_1, \ldots, J_n$ in the applicable subsection. In a final step the nonterminal backbone and both kinds of functions are assembled to $G_{h,\pi}$.

4.2.1 Construction of the word-tuple functions

In the LCFRS component of the hybrid grammar every nonterminal J in π shall generate maximal substrings of $\mathrm{str}(h)$ that correspond to the positions in J. Technically speaking, J generates in its i-th component the string corresponding to the i-th component of $\mathrm{vmseq}_{<_s}(J)$. Now assume that J's children $J_1, \ldots, J_n$ generate substrings $s^{(1)}_{1,k_1}, \ldots, s^{(n)}_{1,k_n}$ of $\mathrm{str}(h)$. Since π is a recursive partition, each

string $s_j^{(i)}$ is a substring of one of the strings generated by J. Also, given distinct pairs (i,j) and (i',j'), the sequences of positions corresponding to $s_j^{(i)}$ and $s_{j'}^{(i')}$ are non-overlapping. Thus, we can construct a word-tuple function that composes $s_{1,k_1}^{(1)},\dots,s_{1,k_n}^{(n)}$ to the strings generated by J.

Formally, for every $q \in \mathrm{pos}_\Sigma(\pi)$ and $J = \pi(q)$, we construct a word-tuple function f_J^{S} as follows.

1. If $|J| = 0$, then we set $f_J^{\mathrm{S}} = \langle() \rangle = \mathrm{str}(h)$.[1] Observe that $f_J^{\mathrm{S}} \in \mathcal{F}_{\varepsilon,1}^{\mathrm{LCFRS}(\mathcal{I}([0],\Sigma))}$.
2. If $|J| = 1$, i.e., $J = \{p\}$ for some $p \in \mathrm{pos}(s)$, then we set $f_J^{\mathrm{S}} = \langle (s(p)^{\boxed{1}}) \rangle$. Observe that $f_J^{\mathrm{S}} \in \mathcal{F}_{\varepsilon,1}^{\mathrm{LCFRS}(\mathcal{I}([1],\Sigma))}$.
3. Otherwise, if $|J| > 1$, then there exists $n \geq 2$ such that $qn \in \mathrm{pos}(\pi)$ and $q(n+1) \notin \mathrm{pos}(\pi)$. Let now $\langle \tau_1^{(0)},\dots,\tau_{k_0}^{(0)} \rangle = \mathrm{vmseq}_{<_s}(J)$. Furthermore, for each $i \in [n]$, we let $J_i = \pi(qi)$ and $\langle \tau_1^{(i)},\dots,\tau_{k_i}^{(i)} \rangle = \mathrm{vmseq}_{<_s}(J_i)$. We define $m_i = \sum_{j=1}^{i} k_i$ for each $i \in [n]_0$. We let

$$\langle \tau_1,\dots,\tau_{m_n} \rangle = \langle \tau_{1,k_1}^{(1)},\dots,\tau_{1,k_n}^{(n)} \rangle$$

and set

$$f_J^{\mathrm{S}} = \langle s_1,\dots,s_{k_0} \rangle$$

where, for every $i \in [k_0]$, we set $s_i = (x_{j_1},\dots,x_{j_k})$ if $\tau_i^{(0)} = \tau_{j_1} \cdots \tau_{j_k}$ for some $k \in \mathbb{N}_+$ and $j_1,\dots,j_k \in \{1,\dots,m_n\}$. Observe that k and $j_1,\dots,j_k$ are unique and that $f_J^{\mathrm{S}} \in \mathcal{F}_{k_1\cdots k_n,k_0}^{\mathrm{LCFRS}(\mathcal{I}([0],\Sigma))}$.

We set $\varphi(J) = |\,\mathrm{mseq}_{<_s}(J)|$, i.e., $\varphi(J) = 1$ in cases 1 and 2, and $\varphi(J) = k_0$ in case 3.

Example 4.9. Recall the dependency structure $(s,\leq)$ in Figure 4.1 (on the left) with its directly extracted recursive partitioning π. By the above method we extract the following word-tuple functions.

1. For the singleton sets in π we apply case 2 and obtain the following word-tuple functions:

J	$\{1\}$	$\{11\}$	$\{12\}$	$\{121\}$
f_J^{S}	$\langle (\mathrm{h}^{\boxed{1}}) \rangle$	$\langle (\mathrm{P}^{\boxed{1}}) \rangle$	$\langle (\mathrm{l}^{\boxed{1}}) \rangle$	$\langle (\mathrm{M}^{\boxed{1}}) \rangle$

2. For $J = \{121, 12\}$ we construct a word-tuple function according to case 3. Then $J_1 = \{121\}$, $J_2 = \{12\}$, $\mathrm{vmseq}_<(J_1) = \langle 121 \rangle = \langle \tau_1 \rangle$, and $\mathrm{vmseq}_<(J_2) = \langle 12 \rangle = \langle \tau_2 \rangle$. Since $\mathrm{vmseq}_<(J) = \langle 121, 12 \rangle = \langle \tau_1^{(0)}, \tau_2^{(0)} \rangle$, we have that $s_1 = (x_1)$ and $s_2 = (x_2)$. Thus, $f_J^{\mathrm{S}} = \langle (x_1), (x_2) \rangle$.
3. Also, for $J = \{11, 121, 1, 12\}$ we apply case 3. The following table lists the assignments for $\tau_1,\dots,\tau_4$ and, for each $i \in [3]$, the values of J_i and $\mathrm{vmseq}_<^{\leq}(J_i)$.

i	J_i	$\mathrm{vmseq}_<(J_i)$	$=$	$\langle \tau_{m_{i-1}+1},\dots,\tau_{m_i} \rangle$
1	$\{11\}$	$\langle 11 \rangle$		$\langle \tau_1 \rangle$
2	$\{121, 12\}$	$\langle 121, 12 \rangle$		$\langle \tau_2, \tau_3 \rangle$
3	$\{1\}$	$\langle 1 \rangle$		$\langle \tau_4 \rangle$

Note that $\mathrm{vmseq}_<(J) = \langle 11\,121\,1\,12 \rangle$ which corresponds to $\langle \tau_1\,\tau_2\,\tau_4\,\tau_3 \rangle$. Thus, we have that $f_J^{\mathrm{S}} = \langle (x_1, x_2, x_4, x_3) \rangle$.

Observe that the dimension of $\mathrm{vmseq}_<(J)$ determines the fanout of f_J^{T}. In Figure 4.2 it is depicted how the induced word-tuple functions (and those that we would obtain for the right-branching recursive partitioning of $\mathrm{pos}_\Sigma(s)$) interact. ■

[1] This case applies only if $\mathrm{pos}_\Sigma(s) = \emptyset$.

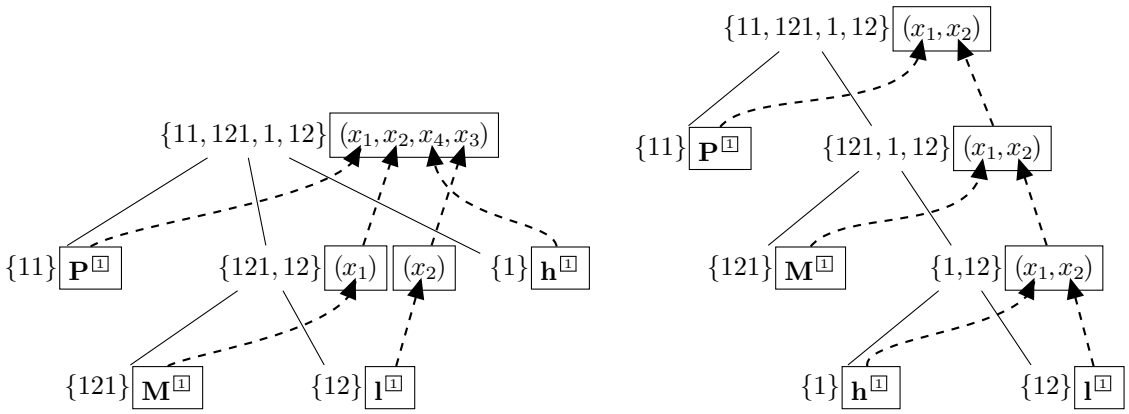


Figure 4.2: Information flow of the induced word-tuple functions for the directly extracted and the right-branching recursive partitioning.

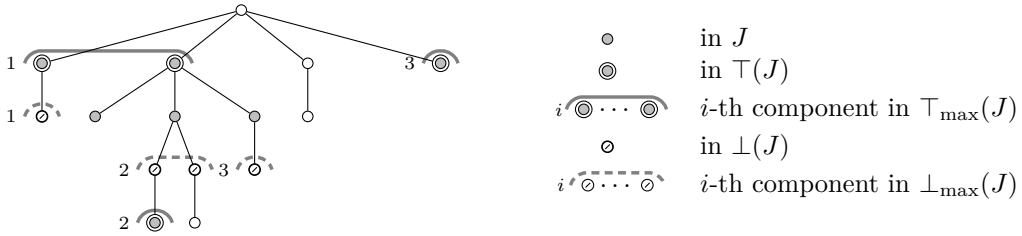


Figure 4.3: The schematic representation of an s-term, a set of nodes J, $\top(J)$, $\top_{\max}(J)$, $\bot(J)$, and $\bot_{\max}(J)$.

Recall that the maximal fanout of some nonterminal and the maximal rank of a production determine the parsing complexity of an LCFRS. Thus, if π is a binarized structure and, for every node J of π, we have that $|\operatorname{vmseq}_{<_s}(J)|$ is below some threshold ℓ, then parsing with the string component of the induced hybrid grammar will be more efficient. Moreover, choosing the left-branching or the right-branching recursive partitioning of $\operatorname{pos}_\Sigma(s)$ constraints the LCFRS such that its language is regular, i.e., a sentence can even be parsed in linear time.

4.2.2 Top and bottom positions

For the induction of sDCP functions, we first have to partition the s-terms s and later recompose the resulting parts. This process is slightly more involved than the construction of word-tuple functions since not only substrings are concatenated but a hierarchical structure is built. For this composition we need to refer to the *top-positions* of some subset J of $\operatorname{pos}(s)$, i.e., the positions in J whose parents are not in J. It is only logical to define also *bottom-positions* that have their parent in J but are not J by themselves: these are the positions where an sDCP function will later insert the sub-s-term at the top-positions of another set J'. Next, we formally define both concepts as well as vectors of maximal consecutive sequences of top-positions and bottom-positions.

Let s be an s-term. For every set $J \subseteq \operatorname{pos}(s)$, we define

$$\top(J) = \{p \in J \mid \nexists p' \in J \colon p' \chi p\}$$

to be the set of *top-positions of* J. Likewise, we let

$$\bot(J) = \{p \in \operatorname{pos}(s) \setminus J \mid \exists p' \in J \colon p' \chi p\}$$

be the set of *bottom-positions of* J. Now we form maximal sequences of consecutive sibling positions in $\top(J)$ and $\bot(J)$ and order these sequences lexicographically by their first member. Let $k \in \mathbb{N}$ and

$w_1, \ldots, w_k \in (\top(J))^+$ such that, for every $i \in [k]$, we have that w_i is a sequence of consecutive sibling positions and there exists no $p \in \top(J)$ such that $w_i p$ or $p w_i$ is a sequence of consecutive sibling positions. Also, for every $p \in \top(J)$, there shall exist $i \in [k]$ such that p occurs in w_i. For every $i \in [k]$, let p_i be the first element in w_i. We require that, for every $i \in [k-1]$, it holds that $p_i <_\ell p_{i+1}$. Then we set $\top_{\max}(J)$ to $\langle w_1, \ldots, w_k \rangle$. We define $\bot_{\max}(J)$ analogously. A visualization of this concept is given in Figure 4.3.

4.2.3 Construction of sDCP functions for dependency structures

For a dependency structure it holds that $\Sigma^{(0)} = \Sigma = \Gamma = \Gamma^{(1)}$, i.e., each position of s is in the total order $<_s$. Every nonterminal J shall generate the s-terms $s_1^{(0)}, \ldots, s_{\sigma_0}^{(0)}$ at the positions in $\top_{\max}(J)$. To this end, we construct an sDCP function that composes the s-terms generated by J's children $J_1, \ldots, J_n$. However, there might be a position p which is not in J (and thus also not in $J_1, \ldots, J_n$) but below some position in J. There will be another nonterminal that contains p which has to account for this sub-term of s. Note that there is a position q such that q's parent is in J, $q \, \chi^* \, p$, and every position on the path from q to p is not in J. Obviously, q must be in $\bot(J)$ and the sub-term of s at position q contains the sub-term at position p. We equip J with an inherited attribute that stands for q. More precise, J has an inherited attribute for each component of $\bot_{\max}(J)$ and we rely on J's parent or siblings to provide the missing sub-s-terms.

Formally, for every $q \in \mathrm{pos}(\pi)$ and $J = \pi(q)$, let

$$\langle \tau_{1,\iota_0} \rangle = \bot_{\max}(J) \quad \text{and } \langle \tau_{\iota_0+1,\iota_0+\sigma_0} \rangle = \top_{\max}(J)$$

Also, set $\iota(J) = \iota_0$ and $\sigma(J) = \sigma_0$, i.e., to the dimension of $\bot_{\max}(J)$ and $\top_{\max}(J)$, respectively. We construct the sDCP function f_J^{T} as follows.

1. If $|J| = \emptyset$, then we set $f_J^{\mathrm{T}} = \langle () \rangle = \langle s \rangle$. Observe that $f_J^{\mathrm{T}} \in \mathcal{F}_{\varepsilon,(0,1)}^{\mathrm{sDCP}(\mathcal{I}([0],\Gamma))}$.
2. If $|J| = 1$, i.e., $J = \{p\}$ for some $p \in \mathrm{pos}(s)$, there are two cases.
   i) If $\bot_{\max}(J) = \langle \rangle$, i.e., p is a leaf in s, then we set $f_J^{\mathrm{T}} = \langle (s(p)^{\boxed{1}}(())) \rangle$. Observe that $f_J^{\mathrm{T}} \in \mathcal{F}_{\varepsilon,(0,1)}^{\mathrm{sDCP}(\mathcal{I}([1],\Gamma))}$.
   ii) If $\bot_{\max}(J) = \langle \tau_1 \rangle$, i.e., p has the child positions τ_1 in s, then we set $f_J^{\mathrm{T}} = \langle (s(p)^{\boxed{1}}((x_1^{(0)}))) \rangle$. Observe that $f_J^{\mathrm{T}} \in \mathcal{F}_{\varepsilon,(1,1)}^{\mathrm{sDCP}(\mathcal{I}([1],\Gamma))}$.
3. Otherwise, if $|J| > 1$, then there exists $n \geq 2$ such that $qn \in \mathrm{pos}(\pi)$ and $q(n+1) \notin \mathrm{pos}(\pi)$. We let

   $$\langle \varrho_{1,\sigma_0}^{(0)} \rangle = \top_{\max}(J) \quad \text{and} \quad \langle \tau_{1,\iota_0}^{(0)} \rangle = \bot_{\max}(J).$$

   For every $i \in [n]$, we let $J_i = \pi(qi)$,

   $$\langle \varrho_{1,\iota_0}^{(i)} \rangle = \bot_{\max}(J_i), \quad \text{and} \quad \langle \tau_{1,\sigma_0}^{(i)} \rangle = \top_{\max}(J_i).$$

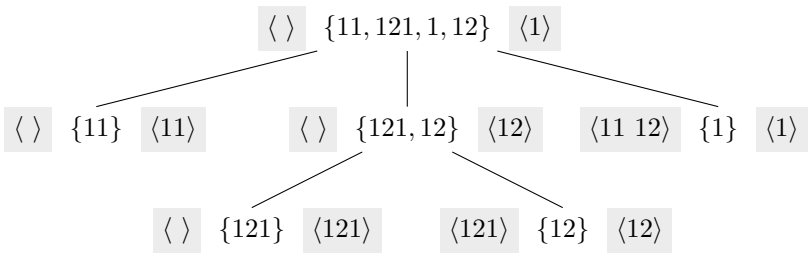
   We define

   $$f_J^{\mathrm{T}} = \langle s_{1,\sigma_0}^{(0)}, s_{1,\iota_1}^{(1)}, \ldots, s_{1,\iota_n}^{(n)} \rangle$$

   where, for every $(i,j) \in \mathcal{S}((\iota_1,\sigma_1)\cdots(\iota_n,\sigma_n),(\iota_0,\sigma_0))$, we set $s_j^{(i)} = (x_{j_1}^{(i_1)}, \ldots, x_{j_k}^{(i_k)})$ if $\varrho_j^{(i)} = \tau_{j_1}^{(i_1)} \ldots \tau_{j_k}^{(i_k)}$.

It can be proven by case analysis that in case 3 it holds that $f_J^{\mathrm{T}} \in \mathcal{F}_{(\iota_1,\sigma_1)\cdots(\iota_n,\sigma_n),(\iota_0,\sigma_0)}^{\mathrm{sDCP}(\mathcal{I}([0],\Gamma))}$.

Example 4.10. Recall the dependency structure in Figure 4.1 and the recursive partitioning π directly extracted from it. Below, we annotated $\bot_{\max}(J)$ and $\top_{\max}(J)$ in gray boxes to the left and to the right of each node J in π, respectively.



We construct sDCP functions as follows.

1. For each singleton set J, we construct an sDCP function according to case 2.i or 2.ii depending on the dimension of $\bot_{\max}(J)$:

J	$\{11\}$	$\{121\}$	$\{12\}$	$\{1\}$
f_J^{T}	$\langle(\mathbf{P}^{\boxed{1}}(()))\rangle$	$\langle(\mathbf{M}^{\boxed{1}}(()))\rangle$	$\langle(\mathbf{l}^{\boxed{1}}((x_1^{(0)})))\rangle$	$\langle\mathbf{h}^{\boxed{1}}((x_1^{(0)}))\rangle$

2. Consider $J = \{121, 12\}$, $J_1 = \{121\}$ and $J_2 = \{12\}$. Since $\top_{\max}(J)$ has dimension one, J has only one synthesized attribute, i.e., only $s_1^{(0)}$ needs to be specified. Likewise, $\bot_{\max}(J_1)$ has dimension 0, i.e., J_1 has no inherited attribute. Contrarily, $\bot_{\max}(J_2)$ has dimension 1 and we need specify $s_1^{(2)}$. Observe that

$$\top_{\max}(J) = \langle\varrho_1^{(0)}\rangle = \langle 12\rangle = \langle\tau_1^{(2)}\rangle = \top_{\max}(J_2), \qquad \text{i.e., } s_1^{(0)} = (x_1^{(2)}),$$

and that

$$\bot_{\max}(J_2) = \langle\varrho_1^{(2)}\rangle = \langle 121\rangle = \langle\tau_1^{(1)}\rangle = \top_{\max}(J_1), \qquad \text{i.e., } s_1^{(2)} = (x_1^{(1)}).$$

Thus, we obtain that

$$f_J^{\mathrm{T}} = \langle s_1^{(0)}, s_1^{(2)}\rangle = \langle(x_1^{(2)}), (x_1^{(1)})\rangle.$$

3. For $J = \{11, 121, 1, 12\}$, $J_1 = \{11\}$, $J_2 = \{121, 12\}$, and $J_3 = \{1\}$, we have to construct $s_1^{(0)}$ and $s_1^{(3)}$.

$$\top_{\max}(J) = \langle\varrho_1^{(0)}\rangle = \langle 1\rangle = \langle\tau_1^{(3)}\rangle = \top_{\max}(J_2), \qquad \text{i.e., } s_1^{(0)} = (x_1^{(3)}),$$

and that

$$\bot_{\max}(J_3) = \langle\varrho_1^{(3)}\rangle = \langle 11\,12\rangle = \langle\tau_1^{(1)}\tau_2^{(2)}\rangle \qquad \text{i.e., } s_1^{(3)} = (x_1^{(1)}, x_1^{(2)}).$$

The interaction of the constructed sDCP functions is illustrated in Figure 4.4. ■

4.2.4 Construction of sDCP functions for constituent structures

If h is a constituent structure, then $\Sigma = \Gamma^{(0)}$, i.e., every node in the linear order $\leq_s$ is a leaf and s might contain nodes that are not in $\leq_s$. Thus, in contrary to dependency structures, each nonterminal J generates not only the symbols at the positions in J but also the symbols (constituents) at internal positions *over* those leaf positions in J. More precisely, J accounts for an internal position p if every descendant of p labeled with a symbol from Σ is in J and if there exists at least one such descendant. The nodes for which J accounts are grouped to maximal disjoint sub-s-terms. These sub-s-terms are composed from those generated by J's children. Since the nodes in J are leaves, we do not need to equip J with inherited attributes.

Formally, for every $p \in \mathrm{pos}(\pi)$ and $J = \pi(p)$, we define the set of *constituents of* J, denoted by $\mathcal{C}(J)$, as follows:

$$\mathcal{C}(J) = \{p \in \mathrm{pos}(s) \mid \forall q \in \mathrm{pos}_{\Sigma}(s)\colon p\,\chi^*\,q \text{ implies } q \in J \text{ and } \exists q \in J\colon p\,\chi^*\,q\}.$$

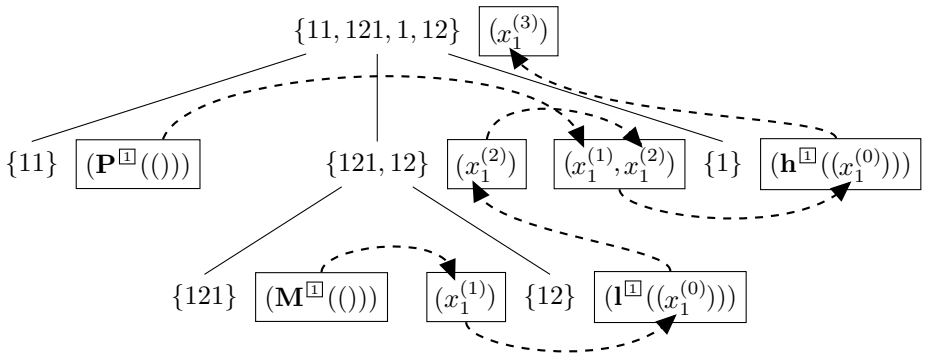


Figure 4.4: The interaction of the sDCP functions constructed in Example 4.10.

If s contains ε-constituents, i.e., nodes that are not above any position in $\leq_s$, then we have to enrich this set. Let $\mathcal{E} = \{p \in \mathrm{pos}(s) \mid \nexists q \in \mathrm{pos}_\Sigma(s) \colon p \,\chi^*\, q\}$. Let $\mathcal{C} = \mathrm{pos}(s) \setminus \mathcal{E}$. We define the set of *$\varepsilon$-siblings of constituents of J* by

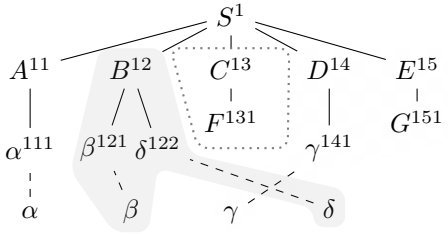
$$\mathcal{E}^S(J) = \left\{ e \in \mathcal{E} \;\middle|\; \begin{array}{ll} \exists p \in \mathcal{C}(J) \colon & p <_{\mathrm{sib}} e \vee e <_{\mathrm{sib}} p \text{ and} \\ \forall p \in \mathcal{C} \colon & (p <_{\mathrm{sib}} e \wedge (\nexists q \in \mathcal{C} \colon p <_{\mathrm{sib}} q <_{\mathrm{sib}} e) \implies p \in \mathcal{C}(J)) \text{ and} \\ & (p >_{\mathrm{sib}} e \wedge (\nexists q \in \mathcal{C} \colon p >_{\mathrm{sib}} q >_{\mathrm{sib}} e) \implies p \in \mathcal{C}(J)) \end{array} \right\}$$

and the set of *ε-constituents of J* by

$$\mathcal{E}(J) = \{e \in \mathcal{E} \mid \exists p \in \mathcal{E}^S(J) \colon p \,\chi^*\, e\}.$$

Intuitively, $\mathcal{E}(J)$ contains every ε-constituent, where the closest non-ε-siblings in either direction are in $\mathcal{C}(J)$, or, if no non-ε-sibling exists, there is an ancestor that satisfies this criterion. The *nodes over J* are $O(J) = \mathcal{C}(J) \cup \mathcal{E}(J)$. Obviously, if $\mathcal{E} = \emptyset$, then $O(J) = \mathcal{C}(J)$.

Example 4.11. Consider the following constituent structure $h = (s, \leq_s)$ over the monadic alphabet Γ, where $\Gamma^{(0)} = \{\alpha, \beta, \gamma, \delta\}$, $\Gamma^{(1)} = \{A, B, \ldots, F\}$ and the Gorn address is annotated in superscript at each node in s.



Note that $\mathcal{C} = \{1, 11, 12, 121, 122, 14, 141\}$ and $\mathcal{E} = \{13, 131, 15, 151\}$. In the following table we list the sets $\mathcal{C}(J)$, $\mathcal{E}^S(J)$, $\mathcal{E}(J)$, and $O(J)$ for several subsets J of $\mathrm{pos}_\Sigma(s)$. In row (a) only 121 is contained in

	J	$\mathcal{C}(J)$	$\mathcal{E}^S(J)$	$\mathcal{E}(J)$	$O(J)$
(a)	$\{121\}$	$\{121\}$	$\emptyset$	$\emptyset$	$\{121\}$
(b)	$\{141\}$	$\{141, 14\}$	$\{15\}$	$\{15, 151\}$	$\{141, 14, 15, 151\}$
(c)	$\{121, 122\}$	$\{121, 122, 12\}$	$\emptyset$	$\emptyset$	$\{121, 122, 12\}$
(d)	$\{111, 141\}$	$\{111, 141, 11, 14\}$	$\{15\}$	$\{15, 151\}$	$\mathcal{C}(J) \cup \mathcal{E}(J)$
(e)	$\{121, 122, 141\}$	$\{121, 122, 12, 141, 14\}$	$\{13, 15\}$	$\{13, 131, 15, 151\}$	$\mathcal{C}(J) \cup \mathcal{E}(J)$
(f)	$\mathrm{pos}_\Sigma(s)$	$\mathcal{C}$	$\{13, 15\}$	$\mathcal{E}$	$\mathrm{pos}(s)$

$\mathcal{C}(J)$ whereas 12 is not since $12 \,\chi\, 122$ and $122 \notin J$. The nodes highlighted with the checkerboard pattern

correspond to $O(J)$ in (b) and those highlighted in gray correspond to $O(J)$ in (c). Row (d) indicates that $O(J)$ does not need to consist only of the positions of a single sub-s-term. Further, for the dotted positions to be $\mathcal{E}(J)$, it is required that both nearest constituent nodes of 13 are in $\mathcal{C}(J)$, i.e., 12 and 14. Thus, the positions surrounded by dotes are included in $O(J)$ of (e) but not in $O(J)$ of (b) or (c). If all positions in pos_Σ are in J, then all positions of s are in $O(J)$ as (f) illustrates. ■

Let now $p \in \mathrm{pos}(\pi)$ and $J = \pi(p)$.

1. If $|J| = 0$, then $p = \varepsilon$. We define $f_J^{\mathrm{T}} = \langle s \rangle$. Observe that $f_J^{\mathrm{T}} \in \mathcal{F}^{\mathrm{sDCP}(\mathcal{I}([0],(\Sigma,\mathrm{rk}_\Gamma)\cup(\Gamma\setminus\Sigma,\mathrm{rk}_\Gamma))}_{\varepsilon,(0,1)}$.
2. If $|J| = 1$, then there is exactly one position q in $O(J) \cap \mathrm{pos}_\Sigma(s)$. We let $\langle \tau \rangle = \top_{\max}(O(J))$ and $f_J^{\mathrm{T}} = \langle (s[s(q)^{\boxed{1}}]_q)|_\tau \rangle$. Observe that $f_J^{\mathrm{T}} \in \mathcal{F}^{\mathrm{sDCP}(\mathcal{I}([1],(\Sigma,\mathrm{rk}_\Gamma)\cup(\Gamma\setminus\Sigma,\mathrm{rk}_\Gamma))}_{\varepsilon,(0,1)}$.
3. If $|J| \geq 2$, then there exists $n \geq 2$ such that $pn \in \mathrm{pos}(\pi)$ and $p(n+1) \notin \mathrm{pos}(\pi)$. For $i \in [n]$ we let $J_i = \pi(pi)$. Also, we let $O'(J) = \bigcup_{i=1}^n O(J_i)$. We set
$$\langle \tau_1^{(0)}, \ldots, \tau_{\sigma_0}^{(0)} \rangle = \top_{\max}(O(J))$$
and, for every $i \in [n]$, we set
$$\langle \tau_1^{(i)}, \ldots, \tau_{\sigma_i}^{(i)} \rangle = \top_{\max}(O(J_i)).$$
For each $j \in [\sigma_0]$, we define ζ_j as follows.
   - Let $\langle \beta_1, \ldots, \beta_\kappa \rangle = \top_{\max}(\{p \in O'(J) \mid (\tau_j^{(0)}, p\,\varepsilon) \in (\chi^* \cup \infty)\})$.[2]
   - For every $k \in [\kappa]$, we let $\varrho_k = (x_{j_1}^{(i_1)}, \ldots, x_{j_l}^{(i_l)})$ if $\beta_k = \tau_{j_1}^{(i_1)} \cdots \tau_{j_l}^{(i_l)}$ where $l \in \mathbb{N}_+$ and, for every $m \in [l]$, we have that $i_m \in [n]$ and $j_m \in [\sigma_{i_m}]$. Note that l, $i_1, \ldots, i_l$, and $j_1, \ldots, j_l$ exist and are unique.
   - For each $k \in [\kappa]$, we set β'_k to $\beta_k \downarrow \tau_j^{(0)}$.
   - Now we define ζ_j as $\left(\cdots \left((s|_{\tau_j^{(0)}})[\varrho_\kappa]_{\beta'_\kappa} \right) [\varrho_{\kappa-1}]_{\beta'_{\kappa-1}} \cdots \right) [\varrho_1]_{\beta'_1}$.

   We set
$$f_J^{\mathrm{T}} = \langle \zeta_1, \ldots, \zeta_{\sigma_0} \rangle.$$
   Observe that $f_J^{\mathrm{T}} \in \mathcal{F}^{\mathrm{sDCP}(\mathcal{I}([0],(\Sigma,\mathrm{rk}_\Gamma))\cup(\Gamma\setminus\Sigma,\mathrm{rk}_\Gamma))}_{(0,\sigma_1)\cdots(0,\sigma_n),(0,\sigma_0)}$.

We set $\iota(J)$ to 0 and $\sigma(J)$ to the arity of $\top_{\max}(J)$.

Example 4.12. We show some sDCP functions that can be extracted from the constituent structure $h = (s, \leq_s)$ in example 4.11.

1. Since $\mathrm{pos}_\Sigma(s)$ is nonempty, case 1 does not apply.
2. For case 2 consider $J = \{141\}$. The only position in $O(J) \cap \mathrm{pos}_\Sigma(s)$ is 141. Also, $\top_{\max}(O(J)) = \langle 14\,15 \rangle$. Thus,
$$f_J^{\mathrm{T}} = \langle (D((\gamma^{\boxed{1}})), E((G))) \rangle,$$
   which corresponds to the checkerboard highlighted sub-s-term where γ was indexed.
3. For case 3 let $J = \{121, 122\}$, $J_1 = \{121\}$, and $J_n = J_2 = \{122\}$. This production accounts for the sub-s-term of s that is highlighted in gray where the sub-s-terms at the position 121 and the position 122 are provided by J_1 and J_2, respectively. Now $O'(J) = \{121, 122\}$, $\top_{\max}(O(J)) = \langle 12 \rangle$, and $\top_{\max}(\{p \in O'(J) \mid (12, p\,\varepsilon) \in (\chi^* \cup \infty)\}) = \top_{\max}(\{121, 122\}) = \langle 121\,122 \rangle$. Obviously, $\langle 121 \rangle = \top_{\max}(O(J_1))$ and $\langle 122 \rangle = \top_{\max}(O(J_2))$. Thus, for $j = 1 = \sigma_0$ we have that $\beta_1 = 121\,122 = \tau_1^{(1)}\,\tau_1^{(2)}$ and $\beta'_1 = \beta_1 \downarrow 12 = 11\,12$. Hence, $\varrho_1 = (x_1^{(1)}, x_1^{(2)})$. Then $\zeta_1 = (s|_{12})[\varrho_1]_{\beta'_1} = (B((\beta, \delta)))[(x_1^{(1)}, x_1^{(2)})]_{11\,12} = (B((x_1^{(1)}, x_1^{(2)})))$ and thus
$$f_J^{\mathrm{T}} = \langle (B((x_1^{(1)}, x_1^{(2)}))) \rangle.$$

[2] Here we write $p\,\varepsilon$ to convert p from $\mathbb{N}_+{}^*$ to a sequence p of length one in $(\mathbb{N}_+{}^*)^+$.

4. The induced sDCP function of case 3 may account also for multiple sub-s-terms. Let $J = \{121, 141\}$, $J_1 = \{121\}$ and $J_2 = \{141\}$. We have that $O(J) = \{121, 141, 14, 15, 151\}$, $\top_{\max}(O(J)) = \langle 121, 14\,15\rangle$, and $\top_{\max}(O'(J)) = \langle 121, 14\,15\rangle = \top_{\max}(O(J_1)) \cdot \top_{\max}(O(J_2))$. Thus

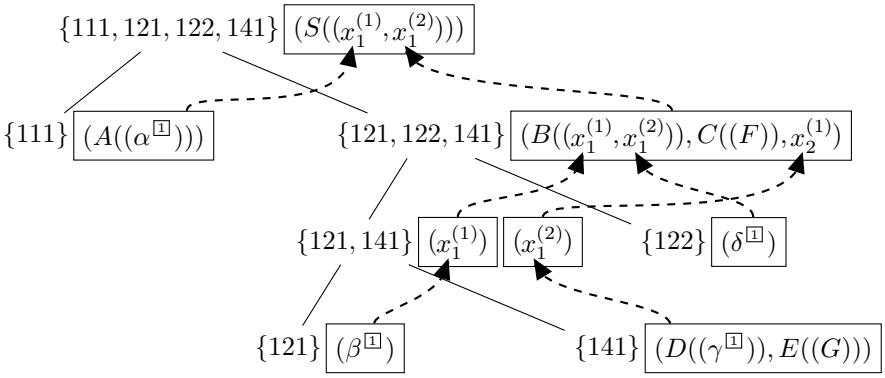
$$f_J^{\mathrm{T}} = \langle (x_1^{(1)}), (x_1^{(2)})\rangle,$$

i.e., this function just passes on the values from the left-hand side nonterminals.

5. The next sDCP function is responsible for the sub-s-term in the gray, dotted, and checkerboard highlighted positions and uses the result from the previous function. Let $J = \{121, 122, 141\}$, $J_1 = \{121, 141\}$ and $J_2 = \{122\}$. Note that $\top_{\max}(O(J)) = \langle 12\,13\,14\,15\rangle$, $\top_{\max}(O'(J)) = \langle \tau_1^{(0)}, \tau_2^{(0)}\rangle = \langle 121\,122, 14\,15\rangle$, $\top_{\max}(O(J_1)) = \langle \tau_1^{(1)}, \tau_2^{(1)}\rangle = \langle 121, 14\,15\rangle$, and $\top_{\max}(O(J_2)) = \langle \tau_1^{(2)}\rangle = \langle 122\rangle$. Thus,

$$f_J^{\mathrm{T}} = \langle (\underbrace{B}_{12}((\underbrace{x_1^{(1)}}_{121}\ \underbrace{x_1^{(2)}}_{122})), \underbrace{C}_{13}((\underbrace{F}_{131})), \underbrace{x_2^{(1)}}_{14\ 15})\rangle.$$

Below, we depict the complete information flow obtained for h and some recursive partitioning. Note that the sDCP functions constructed in 2, 4, and 5 are contained in it.



■

4.2.5 Assembly of the nonterminal backbone, the word-tuple functions and the sDCP functions to an LCFRS/sDCP-hybrid grammar

Let $N = \{J \subseteq \mathrm{pos}_\Sigma(s) \mid \exists p \in \mathrm{pos}(\pi)\colon \pi(p) = J\}$. For every $J \in N$, let $\varphi(J)$, $\iota(J)$, and $\sigma(J)$ as defined in the foregoing sections. For every $p \in \mathrm{pos}(\pi)$, we let $J = \pi(p)$ and $J_1 = \pi(p1), \ldots, J_n = \pi(pn)$. Also, let P contain the hybrid production

$$\begin{aligned}\varrho_J = [&J^{\boxed{\varepsilon}}(s_{1,k_0}) \to J_1^{\boxed{1}}(x_{1,m_1}) \cdots J_n^{\boxed{n}}(x_{m_{n-1}+1,m_n})\\ &, J^{\boxed{\varepsilon}}(x_{1,\iota_0}^{(0)}, \xi_{1,\sigma_0}^{(0)}) \to J_1^{\boxed{1}}(\xi_{1,\iota_1}^{(1)}, x_{1,\sigma_1}^{(1)}) \ldots J_n^{\boxed{n}}(\xi_{1,\iota_n}^{(n)}, x_{1,\sigma_n}^{(n)})]\end{aligned} \tag{4}$$

such that $\langle s_{1,k_0}\rangle = f_J^S$, $\langle \xi_{1,\sigma_0}^{(0)}, \xi_{1,\iota_0}^{(1)}, \ldots \xi_{1,\iota_n}^{(n)}\rangle = f_J^T$, $\iota_0 = \iota(J)$, and, for every $i \in [n]$, it holds that $m_i = \sum_{j=1}^{i} \varphi(J_i)$ and $\sigma_i = \sigma(J_i)$. This constitutes the hybrid grammar

$$G_{h,\pi} = ((N, \mathrm{pos}_\Sigma(s), \Sigma, \varphi), (N, \mathrm{pos}_\Sigma(s), \Gamma, \iota, \sigma), P).$$

Observation 4.13. $\mathcal{L}(G_{h,\pi}) = \{h\}$.

4.3 Induction of a hybrid grammar form a corpus of hybrid trees

4.3.1 Labeling nonterminals

The language of the grammar $G_{h,\pi}$ constructed so far contains exactly one hybrid tree. Given a corpus H of hybrid trees over (Σ, Γ), i.e., $H \subseteq \mathbf{HT}(\Sigma, \Gamma)$, and a recursive partitioning π for each $h \in H$, we

can employ the above procedure to induce $G_{h,\pi} = ((N^h, S^h, \Sigma, \varphi_h), (N^h, S^h, \Gamma, \iota_h, \sigma_h), P^h)$ for each $h = (s^h, \leq_{s^h}) \in H$. Let $N = \{A^h \mid h \in H, A \in N^h\}$ be the disjoint union of nonterminals of all grammars $G_{h,\pi}$. Define the functions $\varphi_N \colon N \to \mathbb{N}_+$, $\iota_N \colon N \to \mathbb{N}$, and $\sigma_N \colon N \to \mathbb{N}_+$ such that, for every $\psi \in \{\varphi, \iota, \sigma\}$, it holds that $\psi_N \colon A^h \mapsto \psi_h(A)$. Moreover, let M be a finite set and assume functions $\varphi_M \colon M \to \mathbb{N}_+$, $\iota_M \colon M \to \mathbb{N}$, and $\sigma_M \colon M \to \mathbb{N}_+$. A *nonterminal labeling strategy* is a function $\ell \colon N \to M$ such that, for every $A \in N$ and $\psi \in \{\varphi, \iota, \sigma\}$, it holds that $\psi_N(A) = \psi_M(A)$.

We define the LCFRS/sDCP-hybrid grammar

$$G = ((M, S, \Sigma, \varphi_M), (M, S, \Gamma, \iota_M, \sigma_M), P)$$

where $S \in M \setminus \ell(N)$ and P contains every production obtained as follows.

- Let $h \in H$ and let π be the chosen recursive partitioning of $\mathrm{pos}_\Sigma(s_h)$.
- For each hybrid production ϱ_J in P^h of the form (4), we let P contain the hybrid production

$$\begin{aligned}\ell(\varrho_J) = [&\ell(J^h)^{\boxed{\varepsilon}}(s_{1,k_0}) \to \ell(J_1^h)^{\boxed{1}}(x_{1,m_1}) \cdots \ell(J_n^h)^{\boxed{n}}(x_{m_{n-1}+1,m_n}) \\ &, \ell(J^h)^{\boxed{\varepsilon}}(x^{(0)}_{1,\iota_0}, \xi^{(0)}_{1,\sigma_0}) \to \ell(J_1^h)^{\boxed{1}}(\xi^{(1)}_{1,\iota_1}, x^{(1)}_{1,\sigma_1}) \ldots \ell(J_n^h)^{\boxed{n}}(\xi^{(n)}_{1,\iota_n}, x^{(n)}_{1,\sigma_n})].\end{aligned} \tag{5}$$

- Also, P contains the production

$$[S(x_1) \to \ell(S^h)(x_1), S(x_1^{(1)}) \to \ell(S^h)(x_1^{(1)})]. \tag{6}$$

Observe that $\mathcal{L}(G) \supseteq \bigcup_{h \in H} \mathcal{L}(G_{h,\pi})$.

4.3.2 The disjoint union labeling

In the easiest scenario we choose $M = N \cup \{S\}$ and define ℓ to be the identity mapping, i.e., we set G to be the *disjoint union* of the individual grammars $G_{h,\pi}$. Then $\mathcal{L}(G) = \bigcup_{h \in H} \mathcal{L}(G_{h,\pi}) = H$. The objective of machine learning is generalization, i.e., to learn a grammar that also generates the dependency structures of sentences that are not in the training corpus. This is obviously not achieved by disjoint union of the grammars $G_{h,\pi}$. The problem is that every nonterminal in N gets mapped injectively to some element of M. Thus, each nonterminal (except for the start symbol) occurs in exactly one successful derivation of G. Next, we explore methods to define labeling strategies that are hopefully not injective and thus yield a grammar G that allows for versatile combinations of productions. Obviously, mapping arbitrary nonterminals to the same element is unlikely to yield a grammar that is sensible from a linguistic viewpoint. Hence, we map each nonterminal A in N to some string that encodes which terminals are generated by A and similarly to *Markovization* described in [KM03] in which context these terminals occur.

4.3.3 Strict nonterminal labeling

For every $\mathrm{s} = (k, \iota, \sigma) \in \mathbb{N}_+ \times \mathbb{N} \times \mathbb{N}_+$, we let M' contain each string B of the form $\langle w_1, \ldots, w_{\iota+\sigma} \rangle$ s, where $w_i \in \Gamma^+$ for every $i \in [\iota + \sigma]$ and $\varphi_M(B) = k$, $\iota_M(B) = \iota$, and $\sigma_M(B) = \sigma$. We define $\ell \colon N \to M'$ such that

$$A^h \mapsto \begin{cases} \ell'(\bot_{\max}(A) \cdot \top_{\max}(A)) \ (\varphi_N(A^h), \iota_N(A^h), \sigma_N(A^h)) & \text{if } h \text{ is a dependency structure and} \\ \ell'(\top_{\max}(O(A))) \ (\varphi_N(A^h), \iota_N(A^h), \sigma_N(A^h)) & \text{if } h \text{ is a constituent structure,} \end{cases}$$

where $\top_{\max}$ and $\bot_{\max}$ are w.r.t. the hybrid tree $h = (t, \leq_t)$ and ℓ' is defined as follows:

$$\ell'(\tau) = \begin{cases} \langle \ell'(\tau_1), \ldots, \ell'(\tau_n) \rangle & \text{if } \tau = \langle \tau_1, \ldots, \tau_n \rangle, \\ t(pi) \cdots t(p(i+k)) & \text{if } \tau = pi \ p(i+1) \ \cdots \ p(i+k) \in \mathrm{pos}(t)^+. \end{cases}$$

Afterwards, we set $M = \{S\} \cup \{B \in M' \mid \exists A \in N \colon \ell(A) = B\}$ in order to obtain a finite set of nonterminals. We call ℓ the *strict nonterminal labeling strategy.*

4.3.4 Child nonterminal labeling

With *child nonterminal labeling* [NV14] proposed a modification of strict nonterminal labeling with the intention of decreasing the number of nonterminals even more. Each sequence of consecutive sibling positions in $\bot_{\max}(A) \cdot \top_{\max}(A)$ that is of length greater than 1 is collapsed and replaced with the common parent. Formally, for every $\mathrm{s} = (\varphi, \iota, \sigma) \in \mathbb{N}_+ \times \mathbb{N} \times \mathbb{N}_+$, we let M' contain each string B of the form $\langle w_1, \ldots, w_{\iota+\sigma} \rangle$ s, where $w_i \in \Gamma \cup \{\text{children-of}(\gamma) \mid \gamma \in \Gamma\} \cup \{\text{children-of(ROOT)}\}$ for every $i \in [\iota + \sigma]$ and ROOT is a new symbol not occurring in Γ. Also, for each B of this form, we set $\varphi_M(B) = k$, $\iota_M(B) = \iota$, and $\sigma_M(B) = \sigma$. We define $\ell \colon N \to M'$ such that

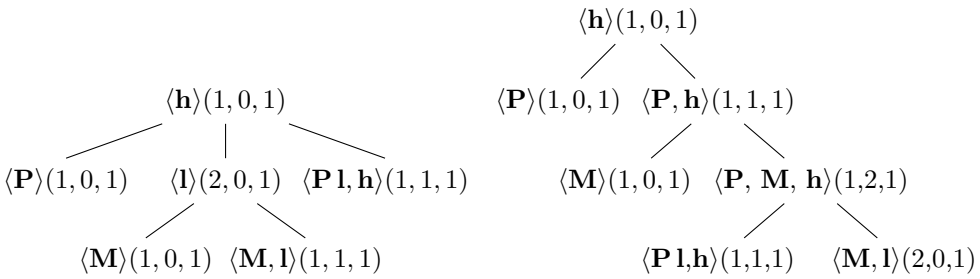
$$A^h \mapsto \begin{cases} \ell'(\bot_{\max}(A) \cdot \top_{\max}(A)) \ (\varphi_N(A^h), \iota_N(A^h), \sigma_N(A^h)) & \text{if } h \text{ is a dependency structure and} \\ \ell'(\top_{\max}(O(A))) \ (\varphi_N(A^h), \iota_N(A^h), \sigma_N(A^h)) & \text{if } h \text{ is a constituent structure,} \end{cases}$$

where $\top_{\max}$ and $\bot_{\max}$ are w.r.t. the hybrid tree $h = (t, \leq_t)$ and ℓ' is defined as follows:

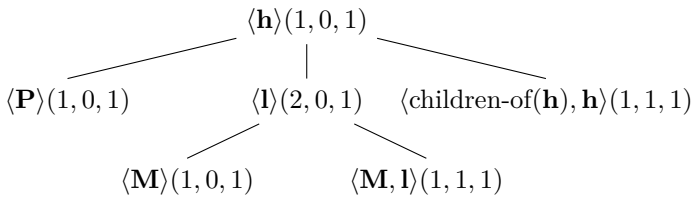
$$\ell'(\tau) = \begin{cases} \langle \ell'(\tau_1), \ldots, \ell'(\tau_n) \rangle & \text{if } \tau = \langle \tau_1, \ldots, \tau_n \rangle, \\ t(pi) & \text{if } \tau = pi \text{ where } pi \in \mathrm{pos}(t), \\ \text{children-of}(t(p)) & \text{if } \tau = pi \ p(i+1) \ \cdots \ p(i+k), \ k > 1, \text{ and } p \in \mathrm{pos}(t), \\ \text{children-of(ROOT)} & \text{if } \tau = i \ (i+1) \ \cdots \ (i+k) \text{ and } k > 1. \end{cases}$$

Note that the last case applies only if h has disconnected substructures. Again, let $M = \{S\} \cup \{B \in M' \mid \exists A \in N \colon \ell(A) = B\}$.

Example 4.14. We applied strict labeling to the nonterminals of the directly extracted and right-branching recursive partitioning of the dependency structure in Figure 4.1:



With child labeling **P l** is contracted to children-of(**h**):

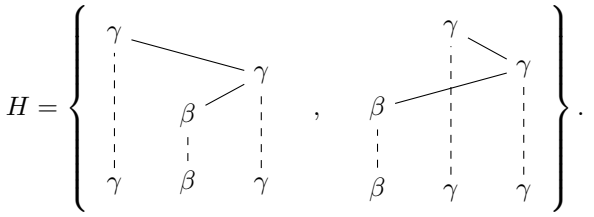


■

4.3.5 Enforcing non-circularity

The use of sDCPs causes a complication in the dependency scenario: since we make use of inherited attributes we may combine productions such that information flows in a circular way.

Example 4.15. Consider the corpus of dependency structures H and the left-branching recursive partitioning π:

$H =$ 

Grammar induction in combination with an appropriate labeling strategy (e.g., every nonterminal is labeled with the multi set of terminals it generates) yields a hybrid grammar with the following productions, where ϱ_1, ϱ_3, ϱ_5, and ϱ_6 were obtained from the first depicted hybrid tree and ϱ_2, ϱ_4, ϱ_5, and ϱ_6 were obtained from the second one.

$$\varrho_1 = \begin{array}{l} S\ ((x_1^{(1)})) \rightarrow A\ ((x_1^{(2)}), x_1^{(1)}, x_2^{(1)})\ C\ ((x_2^{(1)}), x_1^{(2)}) \\ S\ ((x_1, x_2)) \rightarrow A\ (x_1)\ C\ (x_2) \end{array}$$

$$\varrho_2 = \begin{array}{l} S\ ((x_2^{(1)})) \rightarrow A\ ((x_1^{(2)}), x_1^{(1)}, x_2^{(1)})\ C\ ((x_1^{(1)}), x_1^{(2)}) \\ S\ ((x_1, x_2)) \rightarrow A\ (x_1)\ C\ (x_2) \end{array}$$

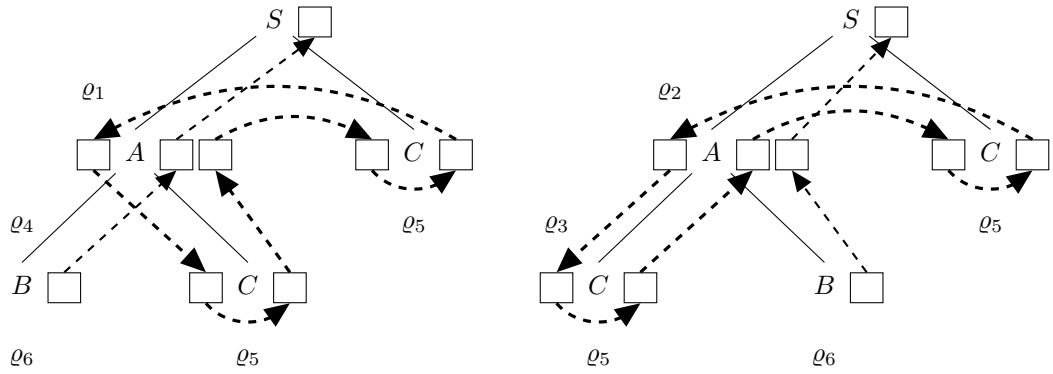
$$\varrho_3 = \begin{array}{l} A\ (x_1^{(0)}, (x_1^{(1)}), (x_1^{(2)})) \rightarrow C\ ((x_1^{(0)}), x_1^{(1)})\ B\ (x_1^{(2)}) \\ A\ ((x_1, x_2)) \rightarrow C\ (x_1)\ B\ (x_2) \end{array}$$

$$\varrho_4 = \begin{array}{l} A\ (x_1^{(0)}, (x_1^{(1)}), (x_1^{(2)})) \rightarrow B\ (x_1^{(1)})\ C\ ((x_1^{(0)}), x_1^{(2)}) \\ A\ ((x_1, x_2)) \rightarrow B\ (x_2)\ C\ (x_1) \end{array}$$

$$\varrho_5 = \begin{array}{l} C\ (x_1^{(0)}, (\ \gamma\ ((x_1^{(0)})))) \rightarrow \varepsilon \\ C\ (\ (\ \gamma\ )\ ) \rightarrow \varepsilon \end{array}$$

$$\varrho_6 = \begin{array}{l} B\ ((\ \beta\ ((\ )))) \rightarrow \varepsilon \\ B\ ((\ \beta\ )\ ) \rightarrow \varepsilon \end{array}$$

Combining ϱ_1 with ϱ_4 or ϱ_2 with ϱ_3 leads to circular flow of information:



■

The way we defined the derivation semantics of sDCPs implies that such constellations do normally not occur in successful derivations.[3] However, we want to use the induced hybrid grammar to determine the constituent structure or dependency structure of some sentence. Thus, we want to reconstruct a hybrid tree for every successful parse of the string component of the hybrid grammar. If the grammar is non-circular, then we can always evaluate the sDCP derivation that corresponds to the successful LCFRS derivation.[4]

To enforce non-circularity we have to make the dependencies between inherited and synthesized attributes explicit. Every nonterminal of the sDCP is assigned some *is-dependency signature* that constraints how its attributes are used in a production. Although such notions can be formally incorporated into the definition of sDCPs,[5] we will only append this signature to the strings generated by the labeling strategies.

The is-dependency signature is basically a strict order $\prec$ on $[n]$ where n is the number of inherited and synthesized arguments. Here, $i \prec j$ shall express that the i-th argument contributes to the j-th one which is the case if the top-positions or bottom-positions that correspond to the i-th argument are below those of the j-th one. Thus, we do not only take local dependencies, i.e., those in one production, into account, but also consider information flow through the whole derivation tree. If in every production the sDCP function adheres the is-dependency signatures of the nonterminal backbone, then the strictness of $\prec$ prevents cycles. Since we enforce the single syntactic use requirement in sDCPs, we know that every argument has at most one direct $\prec$-successor. Thus, we can find a compact representation of $\prec$ as an s-term of size linear in the number of inherited and synthesized attributes.

Definition 4.16. Let $n \in \mathbb{N}_+$. We define the set of *n-ary is-dependencies*, denoted by $\mathrm{DEP}(n)$, to contain every strict order $\prec$ on $[n]$ that satisfies the following. Let $i, j, k \in [n]$. We require that

$$(i \prec j \text{ and } i \prec k) \quad \text{implies} \quad (j \preceq k \text{ or } k \preceq j).$$

We represent $\prec$ as an s-term $s_\prec$ over the monadic alphabet $[n] = [n]^{(1)}$ such that

- every $i \in [n]$ occurs exactly once in $s_\prec$,
- $i \prec j$ iff the position of j in $s_\prec$ is a strict prefix of the position of i in $s_\prec$, and
- for every $p \in \mathbb{N}_+^*$ and $i \in \mathbb{N}_+$ such that $pi, p(i+1) \in \mathrm{pos}(s)$, we have that $s(pi) < s(p(i+1))$, where $<$ is the usual order on $\mathbb{N}$.

In the following we identify $\prec$ and $s_\prec$. ■

Let J be the node of a recursive partitioning and $\langle \tau_{1,\iota}, \tau_{\iota+1,\iota+\sigma} \rangle = \bot_{\max}(J) \cdot \top_{\max}(J)$. We set the *is-dependency signature of J* to $\delta(J) = d \in \mathrm{DEP}(\iota + \sigma)$ such that, for every $i, j \in [\iota + \sigma]$, it holds that $(i, j) \in d$ iff $\tau_j \, \chi^+ \, \tau_i$.

[3] There are marginal instances of sDCPs where cyclic information flow occurs in successful derivations.
[4] We describe an evaluation algorithm in Section 5.4.8 that is only correct if the derivation does not contain cycles.
[5] This would amount to defining a subclass of sDCPs similar to strongly non-circular attribute grammars [Jou84].

Example 4.17. The nonterminal A of example 4.15 has different is-dependency signature in the productions ϱ_1 and ϱ_3 than in ϱ_2 and ϱ_4. In the former case $\delta(A) = 2(1(3))$ since the first synthesized argument 2 is built using the first inherited argument 1. Also, 1 depends on the second synthesized argument 3. In the latter case $\delta(A) = 3(1(2))$ since the roles of the two synthesized arguments of A are swapped. Annotating the label A with $\delta(A)$ prohibits both circular combinations of productions depicted in example 4.15. ■

Observation 4.18. The grammar $G_{h,\pi}$ induced from a hybrid tree h and a recursive partitioning π is always non-circular. Thus, also grammar induction from a corpus of hybrid trees yields a non-circular grammar if is-dependencies are annotated to nonterminals.

4.3.6 Weighting the hybrid grammar

We weight the LCFRS/sDCP-hybrid grammar $G = ((M, S, \Sigma, \varphi_M), (M, S, \Gamma, \iota_M, \sigma_M), P)$ induced from H by *relative frequency estimation*. Let the *count* of some hybrid production $\varrho \in P$ denote how often ϱ was induced. Formally, let $G_{h,\pi} = ((N^h, S^h, \Sigma, \varphi_h), (N^h, S^h, \Gamma, \iota_h, \sigma_h), P^h)$ the induced grammar for every $h \in H$ where π is the chosen recursive partitioning for h. For every $\varrho \in P$ of type (5), we let

$$\mathrm{count}(\varrho) = \sum_{h \in H} \sum_{q \in \mathrm{pos}(\pi)} \delta(\varrho, \ell(\varrho_J)) \qquad \text{where } J = \pi(q),\ \delta(\varrho, \varrho') = \begin{cases} 1 & \text{if } \varrho = \varrho' \\ 0 & \text{otherwise} \end{cases}$$

and ϱ_J and $\ell(\varrho_J)$ are as in (4) and (5), respectively. For productions $\varrho \in P$ of type (6) with the form $[S^{\boxed{\varepsilon}}(x_1) \to A^{\boxed{1}}(x_1), S^{\boxed{\varepsilon}}(x_1^{(1)}) \to A^{\boxed{1}}(x_1^{(1)})]$ we let

$$\mathrm{count}(\varrho) = \sum_{h \in H} \delta(\ell(S^h), A).$$

Further, for every $A \in M$, let P_A be the subset of P in which every hybrid production has the left-hand side nonterminal $A^{\boxed{\varepsilon}}$. Then, for every $A \in M$ and $\varrho \in P_A$, we set

$$p(\varrho) = \frac{\mathrm{count}(\varrho)}{\sum_{\varrho' \in P_A} \mathrm{count}(\varrho')}.$$

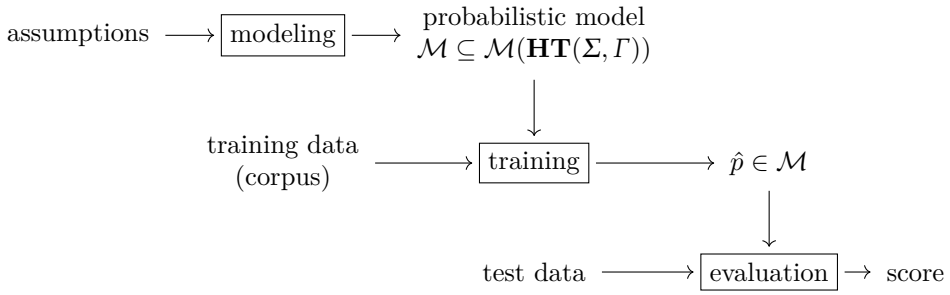
The induced probabilistic LCFRS/sDCP-hybrid grammar is $G' = ((M, S, \Sigma, \varphi_M), (M, S, \Gamma, \iota_M, \sigma_M), P, p)$. Note that relative frequency estimation guarantees the properness of G'.

5 Experimental evaluation of hybrid grammars

In this chapter we describe the implementation and evaluation of hybrid grammars and grammar induction. We give a high-level description of the experimental setup before explaining the used corpora and metrics. Afterwards, we describe the algorithms for LCFRS parsing and sDCP evaluation and the implementation of our system in Python. Finally, the results of applying this implementation to the corpora are reported, interpreted, and compared to reference scores.

5.1 Experiment setup

We follow a common three step experiment setup from statistical machine translation [Lop08] to test the suitability of the hybrid grammar approach to represent non-projective dependency structures and discontinuous constituent structures: modeling, training, and evaluation. Note that we employ the related terminology from statistics although our definition of probabilistic hybrid grammars is not probabilistic in the strict sense, cf. Definition 3.27.



In the above graphic $\mathcal{M}(\mathbf{HT}(\Sigma, \Gamma))$ is the set of all probability distributions over $\mathbf{HT}(\Sigma, \Gamma)$. Our assumptions to modeling restrict this set to the set of probability distributions $\mathcal{M}$ that are induced by probabilistic LCFRS/sDCP-hybrid grammars. During training, we induce an (unweighted) LCFRS/sDCP-hybrid grammar G' from the corpus and perform relative frequency estimation to obtain probabilities for its productions. This probabilistic LCFRS/sDCP-hybrid grammar G induces a probability distribution $\hat{p} \in \mathcal{M}$. Finally, we evaluate the system by comparing each hybrid tree h_g from a test corpus with the hybrid tree h_p, where

$$\begin{aligned} h_\mathrm{p} &= \operatorname{argmax}_{h \in \mathbf{HT}(\Sigma,\Gamma):\ \mathrm{str}(h)=\mathrm{str}(h_\mathrm{g})} \hat{p}(h) \\ &= \operatorname{argmax}_{h \in \mathbf{HT}(\Sigma,\Gamma):\ \mathrm{str}(h)=\mathrm{str}(h_\mathrm{g})} [\![G]\!](h). \end{aligned}$$

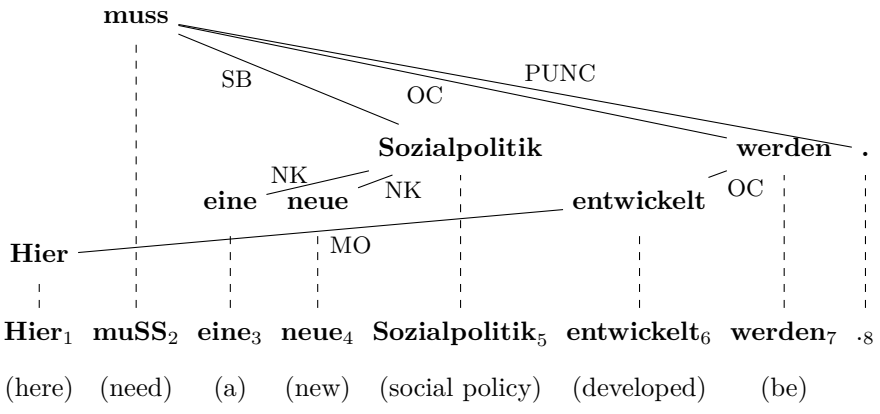
In other words, h_p is obtained by probabilistic parsing of $\mathrm{str}(h_\mathrm{g})$ with the string component of G and a subsequent evaluation of the sDCP productions of the best parse. To evaluate our system we score the similarity of h_g and h_p according to an established metric.

5.2 Corpora and metrics for dependency parsing

The annual *Conference on Natural Language Learning* (CoNLL) is accompanied by a shared task on a machine learning problem from the natural language domain. In 2006 and 2007 this task has been multilingual dependency parsing [BM06; Niv+07]. In order to allow many teams to run their parsers on

```
1   Hier           _   ADV     ADV     _   6   MO     2   MO
2   mu             _   VMFIN   VMFIN   _   0   ROOT   0   ROOT
3   eine           _   ART     ART     _   5   NK     5   NK
4   neue           _   ADJA    ADJA    _   5   NK     5   NK
5   Sozialpolitik  _   NN      NN      _   2   SB     2   SB
6   entwickelt     _   VVPP    VVPP    _   7   OC     7   OC
7   werden         _   VAINF   VAINF   _   2   OC     2   OC
8   .              _   $.      $.      _   2   PUNC   2   PUNC
```

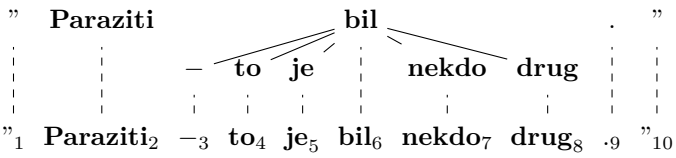
(a) A sentence from TIGER in the format of the CoNLL-X shared task, and



(b) a graphical representation of its (labeled) dependency structure.

```
1   "          "        PUNC    PUNC                    _                                                                    0   ExD    _   _
2   Paraziti   parazit  Noun    Noun-common             Gender=masculine|Number=plural|Case=nominative                        0   ExD    _   _
3   –          –        PUNC    PUNC                    _                                                                    6   AuxG   _   _
4   to         ta       Pronoun Pronoun-demonstrative   Gender=neuter|Number=singular|Case=nominative|Syntactic-Type=nominal  6   Sb     _   _
5   je         biti     Verb    Verb-copula             VForm=indicative|Tense=present|Person=third|Number=singular|Negative=no  6   AuxV   _   _
6   bil        biti     Verb    Verb-copula             VForm=participle|Tense=past|Number=singular|Gender=masculine|Voice=active  0   ExD    _   _
7   nekdo      nekdo    Pronoun Pronoun-indefinite      Case=nominative|Syntactic-Type=nominal                               6   Pnom   _   _
8   drug       drug     Pronoun Pronoun-indefinite      Gender=masculine|Number=singular|Case=nominative|Syntactic-Type=nominal  7   Atr    _   _
9   .          .        PUNC    PUNC                    _                                                                    0   ExD    _   _
10  "          "        PUNC    PUNC                    _                                                                    0   ExD    _   _
```

(c) A sentence from SDT in the format of the CoNLL-X shared task, and



(d) a graphical representation of its (unlabeled) dependency structure.

Figure 5.1: Dependency structures in CoNLL-X format and their graphical representations.

Corpus	Language	LAS	LAS (avr)	UAS	UAS (avr)	LA	LA (avr)
TIGER	German	87.3	78.6	90.4	82.6	92.1	86.3
METU-Sabanci	Turkish	65.7	56.0	75.8	69.4	78.5	69.6
SDT	Slovene	73.4	65.2	83.2	76.5	82.5	76.3

Table 5.1: Top and average (avr) scores in the CoNLL-X shared task.

corpora from different languages, a homogeneous format was introduced and a common metric for success defined. A German and a Slovene sentence in what we henceforth call *CoNLL format* are depicted in Figure 5.1. In this format sentences are finite sequences of tokens with ten fields of which the first eight are as follows: the position of the token in the sentence, the form, the lemma, the coarse POS-tag (CPOS), the POS-tag, optional (morphological) features (FEATS) like case or number, the position of the token's head (HEAD) in the sentence, and the dependency relation (DEPREL) the token has to its head. In addition, there are two fields for the projectivized head and the projectivized DEPREL which indicate a projective dependency structure for the same sentence.

Existing dependency and constituent corpora, varying in size and annotated information, were transformed into the CoNLL format. Each corpus was split into a training section of varying size and a test section with approximately 5,000 scoring tokens. A token might have empty fields if some information is not available in the original treebank. An empty field is indicated by `_`.

The CoNLL-X shared task was the following: a sentence is given as a sequence of tokens, for which only the first six fields are available. For each token, the HEAD and the DEPREL, i.e., the seventh and eighth field, shall be predicted correctly. To evaluate the performance of a dependency parser its output on the test section of the corpus is compared to the gold standard annotation, i.e., the manual annotation in the corpus, and three metrics are computed. The *unlabeled attachment score* (UAS) is the percentage of scoring tokens for which the HEAD was found correctly. The *label accuracy* (LA) is the percentage of scoring tokens for which the DEPREL is correct. The *labeled attachment score* (LAS) describes the percentage of scoring tokens with correct DEPREL and HEAD. In each of the above metrics a scoring token is one where lemma or form do not consist completely of punctuation symbols, e.g., punctuation and %-signs are not scored. The LAS is the metric relevant for ranking in the shared task.

Although all corpora share the common format, they have different characteristics influencing the difficulty of parsing. Next, we briefly sketch those of the corpora used in this work and report reference scores and difficulties that arose in the CoNLL-X shared task.

The TIGER treebank [Bra+02] is an annotated German corpus based on articles from the newspaper *Frankfurter Rundschau*. POS-tags are specified using a modified version of the *Stuttgart-Tübingen-Tagset* (STTS) [Sch+99]. Also, lemmata and morphology are annotated. The syntactic structure is given as a (discontinuous) constituent tree whose edges are labeled with syntactic functions. For the CoNLL-X shared task a dependency corpus was automatically extracted and split into a training set of ca. 700,000 tokens and a test set of ca. 5,000 tokens. After this conversion lemma and FEATS are no longer present and POS and CPOS are always similar. There are 46 DEPRELs and each tree's root has the DEPREL ROOT. The size of the training set is very large compared to the other corpora in the CoNLL-X shared task. This might be one reason for the high reference scores which we list in Table 5.1.

A second slightly older German constituent treebank that was not featured in the CoNLL-X shared task is NEGRA [Sku+98]. It shares many of the characteristics of TIGER: the STTS was used for POS-tagging and discontinuities are represented. We included NEGRA to compare our results against the only existing LCFRS-based implementation of a dependency parser by [MK10] that we are aware of. We used their implementation `rparse`[1] for automatic conversion of NEGRA into CoNLL format. In the output no lemma and FEATS information is contained, CPOS and POS are equal for each word, and 760 different DEPRELs occur. 25 DEPRELs occur only in the test set which is not significant as only 0.12% of the scoring tokens are affected. We use the experiment setup of [MK10]: from the sentences of

[1] `rparse` is available under GPL2.0 at `https://github.com/wmaier/rparse`.

length smaller than 25 words the first 14,858 sentences were used for training and the remaining 1,651 for testing. Reference scores include punctuation and are 78.98% in case of UAS and 71.84% in case of LAS for the LCFRS-based system of [MK10]. The MSTParser [McD+05] achieves a UAS of 87.96% and a LAS of 82.62% on this corpus.

The *METU-Sabanci Turkish Treebank* [Ofl+03] is a native dependency corpus. Different text sources were annotated using lemma, 14 CPOS, 30 POS, and 82 binary grammatical features at word level. In total 700 different feature combinations occur in the training corpus and 40 unseen feature combinations occur in the test data. Since Turkish has a rich morphology, there may be multiple tokens for a single word each with its own POS-tag. There is one main token per word to whom the other tokens are attached as children with the DEPREL DERIV. The latter tokens are not scored in the CoNLL-X shared task. Syntactic functions are otherwise annotated with 25 DEPRELs. A sentence may have multiple root nodes. Each of them has the DEPREL ROOT. The relatively low scores from 2006 are depicted in Table 5.1 They were reckoned to be caused by the inherent difficulties of the language, a rather small training corpus based on multiple text sources, and the annotation scheme [Niv+07]. In the 2007 shared task a modified annotation scheme was used with which better results were obtained but we did not use this new version.

The *Slovene Dependency Treebank* (SDT) [De+06] is a small corpus with only 29,000 training tokens. For each word, lemma, CPOS, POS, and grammatical features are annotated. To this end, 11 CPOS, 28 POS and 51 feature categories were used. In the training set occur 479 different combination of features and additional 22 ones occur only in the test set. Syntactic annotation may lead to multiple nodes at root level and utilizes 25 DEPRELs. Reference scores, cf. Table 5.1, are again relatively low, potentially due to the small corpus size.

5.3 A corpus and a metric for constituent parsing

The focus of this work lies on dependency parsing. Thus, we provide only one new experiment for constituent parsing. However, we use a much larger portion of the TIGER corpus than in [NV14]. Now the first 40,000 sentences are used for training omitting 10 where a single tree did not span the entire sentence[2] and from the remaining 10,474 sentences we removed the ones of length greater than 20 leaving 7,597 sentences for testing.

The usual metric for constituent parsing is PARSEVAL [Bla+91]. Often this metric is also referred to as `evalb` which is an implementation by [Col97]. We describe how the PARSEVAL metric translates to hybrid trees. Let $[s,t]$ be a pair of canonically indexed s-terms over (Σ,Γ) where $|s|=n$. For each $p\in\mathrm{pos}(t)$, let $\beta_t(p)=\{i\in[n]\mid\exists q\in\mathbb{N}_+{}^*\colon\exists\sigma\in\Sigma\colon t(pq)=\sigma^{\boxed{i}}\}$. Let now $[s_g,t_g]$ be the gold standard constituent tree and $[s,t]$ the system output where $s=s_g$. We define the sets Relevant, Found, and FoundCorrectly such that

$$\begin{aligned}\text{Relevant}&=\{(\mathcal{D}(t_g(p)),\beta_{t_g}(p))\mid p\in\mathrm{pos}_{\mathcal{I}([n],\Gamma)^{(1)}}(t_g)\},\\ \text{Found}&=\{(\mathcal{D}(t(p)),\beta_t(p))\mid p\in\mathrm{pos}_{\mathcal{I}([n],\Gamma)^{(1)}}(t)\},\text{ and}\\ \text{FoundCorrectly}&=\text{Relevant}\cap\text{Found}.\end{aligned}$$

Then we define *(Labeled) Precision*, *(Labeled) Recall* and *F-measure* as follows:

$$\text{Precision}=\frac{|\text{FoundCorrectly}|}{|\text{Found}|},\qquad\text{Recall}=\frac{|\text{FoundCorrectly}|}{|\text{Relevant}|},\text{ and}$$
$$\text{F-measure}=\frac{2\cdot\text{Precision}\cdot\text{Recall}}{\text{Precision}+\text{Recall}}.$$

Actually, PARSEVAL additionally restricts Relevant and Found to contain only tuples $(\gamma,B)\in\Gamma^{(1)}\times\mathfrak{P}([n])$ where $|B|>1$. However, we did not apply this restriction in our experiments.

[2] The implementation of hybrid trees with multiple nodes at root level of the s-term toke place after these time-consuming experiments.

package	description
`/hybridtree`	Hybrid trees and a legacy-wrapper for constituent trees are defined here.
`/corpora`	This package contains modules for reading and writing tree corpora from and to text files.
`/grammar`	The implementation of LCFRS/sDCP-hybrid grammars.
`/constituent`	Grammar induction for the constituent scenario.
`/dependency`	Grammar induction for the dependency scenario.
`/parser`	Interfaces for LCFRS parser and derivation-trees.
`/active`	An experimental implementation of the active LCFRS parser described in [Kal10, Sec. 7.1.3].
`/naïve`	Implementation of a weighted naïve LCFRS parser.
`/sDCPevaluation`	Implementation of an sDCP evaluator.
`/evaluation`	A database for experiments and an adapter for scoring scripts.
`/util`	3rd party utilities reside here.
`/`	The user interface.

Table 5.2: The structure of the implemented prototype for LCFRS/sDCP-hybird grammars, grammar induction, and LCFRS parsing.

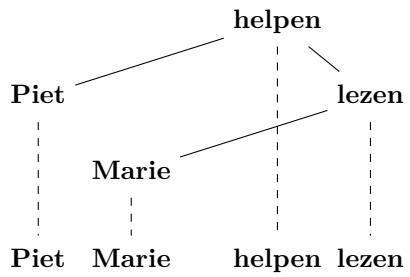
5.4 Implementation

The author implemented hybrid trees, LCFRS/sDCP-hybrid grammars, grammar induction, LCFRS parsing, and sDCP evaluation in Python by extending the software provided with [NV14]. Basically, the main contribution to the software lies in generalizing the existing implementation for constituent parsing to the dependency parsing scenario. This means that the grammar induction for constituent structures was not modified, but many shared components like the implementation of hybrid trees and sDCP evaluation were rewritten. Grammar induction for dependency structures was implemented as a new package. The basic structure of the resulting piece of software is given in Table 5.2. In the following we go through the packages and give a brief explanation of the contents and design decisions that we made. We supplement the documentation of LCFRS parsing and sDCP evaluation with a formal description of the implemented algorithms.

5.4.1 `/hybridtree`

In the package `hybridtree` a generic hybrid tree class and a wrapper for constituent structures are implemented. The author rewrote the existing class by [NV14] to allow for dependency structures and hybrid trees with multiple roots. We do not directly specify a monadic alphabet but use an interface `MonadicToken` for symbols. For the dependency scenario this interface gets implemented by the class `CoNLLToken`, which is in essence a record holding the form, lemma, CPOS, POS, FEATS, and DEPREL. In the constituent case we distinguish `ConstituentTerminal` and `ConstituentCategory` where the former has a form and a POS whereas the latter has a field for a syntactic category.

The class `HybridTree` contains a list `nodes` with a unique *node identifier* for each position of the s-term s of the hybrid tree $h = (s, \leq_s)$ it represents. In addition, there are two dictionaries of whom the first, called `node_token`, maps each node identifier to a `MonadicToken`. The second one, called `children`, maps each node identifier to the list of its children which are arranged according to the sibling order. There is also a list of node identifiers, named `id_yield`, that represents the linear order $\leq_s$. A second list, called `root`, holds those node identifiers that correspond to positions at the root of the s-term in the order from left to right. The class also provides a wide range of utility functions like accessing the parent or the siblings of a node or generating the recursive partitioning directly extracted from h, cf. Definition 4.4. An example is given in Figure 5.2.




```
HybridTree
    nodes       = ['a','b','c','d']
    node_token:
       'a' |->     'Piet'
       'b' |->     'helpen'
       'c' |->     'lezen'
       'd' |->     'Marie'
    root        = ['b']
    children:
       'a' |->     []
       'b' |->     ['a','c']
       'c' |->     ['d']
       'd' |->     []
    id_yield    = ['a','d','b','c']
```

Figure 5.2: A hybrid tree and a schematic representation of the data structure in the implementation.

5.4.2 `/corpora`

The `corpora`-package provides parsers for the TIGER format and the NEGRA format in the constituent case and a parser and printer for the CoNLL format. The implementation for the constituent corpora was taken from [NV14] whereas that for the CoNLL format has been added. The capability to output dependency structures in CoNLL format is crucial for using the official scoring script, cf. Section 5.4.9.

5.4.3 `/grammar`

The `grammar` package contains an implementation of LCFRSs where each production can optionally be equipped with an sDCP function. The class `LCFRS_rule` implements LCFRS productions and consists of an `LCFRS_lhs`, a weight, and a list of nonterminals representing those on the right-hand side. The `LCFRS_lhs` consists of the left-hand side nonterminal and a word-tuple function. An `LCFRS` consists of a start symbol, a list of `LCFRS_rule`s, and dictionaries for fast access to productions with rank 0, to productions with a certain left-hand side nonterminal, and to productions with a certain first terminal in the first component of the word-tuple function.

An sDCP function $\langle s^{(0)}_{1,\sigma_0}, s^{(1)}_{1,\iota_1}, \ldots, s^{(n)}_{1,\iota_n} \rangle$ of the signature $\tilde{s}$, cf. Definition 3.18, is represented as a list ℓ of `DCP_rule`s. Each `DCP_rule` defines one s-term $s^{(i)}_j$ where the i and j are given by a `DCP_variable` and the s-term is given by a list of `DCP_term`s. For each (i, j) in $\mathcal{S}(\tilde{s})$ there must be exactly one `DCP_rule` in ℓ but the order is not constraint. Each `DCP_term` corresponds to a term in $\mathcal{I}^*_{[q],(\Sigma,\mathrm{rk}_\Gamma),(\Delta,\mathrm{rk}_\Gamma)}(X)$ for appropriate alphabets Σ, Γ, and $\Delta = \Gamma \setminus \Sigma$ and some index set $[q]$. In other words, an `DCP_term` consists of a head that is either a `DCP_variable`, a `DCP_string`, or a `DCP_index`, and a list of children which are again `DCP_term`s. This list needs to be empty, if the head is a variable or corresponds to a symbol in $\Gamma^{(0)}$. If the head is a `DCP_string`, then a terminal occurrence not linked to the LCFRS production is modeled. Otherwise, if the head is a `DCP_index`, then it points to a terminal occurrence in the LCFRS component of the hybrid production.

The package does also contain an interface for an sDCP evaluator: a class that implements this interface may visit a `DCP_term`, `DCP_index`, `DCP_string`, or `DCP_variable` and each of these objects calls the respective evaluation method in the evaluator.

In the current implementation every terminal and nonterminal is a string. Consequently, strings need to be compared during parsing, which is likely to cause a high constant factor in parsing time complexity.

parameter	implemented strategies
terminal labeling:	form, CPOS, or POS
nonterminal labeling:	strict and child labeling with either form, CPOS, POS, DEPREL, CPOS and DEPREL, or POS and DEPREL as argument label for each position in $\bot_{\max}(J) \cdot \top_{\max}(J)$ where J is a node in the recursive partitioning
recursive partitioning:	left-branching, right-branching, directly extracted, or fanout-k-transformation of the directly extracted recursive partitioning (where $k > 0$)

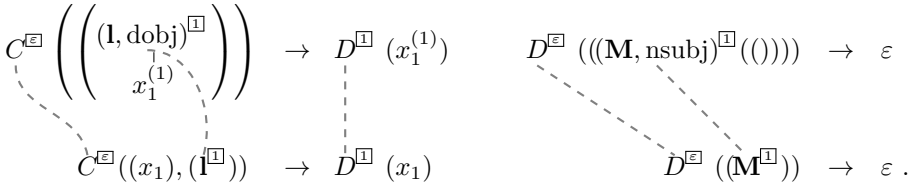
Table 5.3: Strategies for terminal labeling, nonterminal labeling, and recursive partitioning.

5.4.4 `/constituent`

The existing implementation of grammar induction for constituent structures by [NV14] resides almost unaltered in the `constituent` package. Also, the implementation of PARSEVAL (with the subtle variation described above) can be found here. The minor modifications concern changes due to generalizations in other packages.

5.4.5 `/dependency`

The package `dependency` implements the induction of an LCFRS/sDCP-hybrid grammar for a corpus of dependency structures. At this point we generalize the concept of hybrid grammars given in Definition 3.25: originally, the synchronized terminals in a hybrid production must be identical. Moreover, we modeled a labeled dependency structure such that the alphabet consists of tuples of forms and DEPRELs, cf. Definition 3.14. In practice, however, the input part of the CoNLL token does not include a DEPREL. We either obtain it by guessing a DEPREL for each token before presenting the sentence to the parser or, equivalently, by synchronizing a form in the LCFRS component with a pair of form and DEPREL in the sDCP component of a hybrid production. Note that the latter option amounts to postponing the guess of the DEPREL until the application of a production. This is beneficial for parsing since the search space for all possible combination of guesses can be explored simultaneously. Below we depicted two such generalized hybrid productions:



An equally important modification relates to the comprehensive input: a CoNLL token does not only consist of a form and a DEPREL. Thus, we may choose any combination of form, POS, CPOS, lemma, FEATS, and DEPREL as terminal instead of form and DEPREL. In fact, we can make this choice twice: once for the terminals, which we call *terminal labeling*, and where the DEPREL is never included in the word-tuple functions but always in the sDCP functions. The second choice relates to the nonterminal labeling strategies in Sections 4.3.3 and 4.3.4. Given some node J of a recursive partitioning we may select CoNLL token fields for each position in $\bot_{\max}(J) \cdot \top_{\max}(J)$ independently from the tokens used in terminal labeling. In the following we refer to this choice as *argument labels*.

We implemented the strategies for terminal labeling, nonterminal labeling, and recursive partitioning that are given in Table 5.3. From the user interface a high-level method called `induce_grammar` that is parameterized with a list of `HybridTree`s over `CoNLL_token`s, a nonterminal labeling strategy, a terminal labeling strategy, and a recursive partitioning strategy will be invoked. The package also contains the necessary auxiliary functions for computing $\bot_{\max}$ and $\top_{\max}$ and the traversal over a

recursive partitioning in order to extract word-tuple functions and sDCP functions as described in Section 4.2.

5.4.6 `/parser`

The package `parser` has three subpackages: the first two contain one implementation of an LCFRS parser each and the third one hosts an sDCP evaluator. Both LCFRS parsers share a common interface that specifies how they can be called (`AbstractParser`) and what structure a derivation tree needs to have (`AbstractDerivation`). Thus, the parsing strategy can be modified transparently to the user and also new parser implementations can be added later. Each derivation tree that meets the specification of `AbstractDerivation` can be transformed into a hybrid tree $h = (s, \leq_s)$ with the function `derivation_to_hybrid_tree` such that $\mathrm{str}(h)$ is the input string and s is the parse tree that corresponds to the derivation tree. By *parse tree* we mean the tree that is obtained from the derivation tree by replacing the production at each position by its left-hand side nonterminal and adding potential terminals that occur in the word-tuple function of this production as children. A formal definition can be found in [Kal10, Definition 6.9].

There are two implementations of an LCFRS parser: the first uses the *active strategy* [BL05; Kal10, Sec. 7.1.3] and is currently in an experimental state. Since it is complete in the sense that every possible derivation tree is generated, there exist duplicate parse items in case of ambiguity which causes a high memory consumption. As this implementation was not used for the experiments, we do not describe it in more detail.

The second parsing algorithm uses the *naïve strategy* [BL05; Kal10, Sec. 7.1.2]. The author rewrote parts of the implementation from [NV14]: some low-level improvements were made and the capability to generate derivation trees in compliance with the `AbstractDerivation` interface was added. A description of the parsing algorithm is given in the subsequent section.

5.4.7 `/parser/naïve`

The naïve LCFRS parsing strategy is close to the [CS70; You67; Kas65]-style algorithm by [Sek+91]: a passive parse item associates a nonterminal A with subsequences $w_1, \ldots, w_k$ of the input string w such that $A(w_1, \ldots, w_k) \Rightarrow_G^* \varepsilon$. The parser proceeds bottom-up by initially applying only productions with rank 0 to generate passive items. Later multiple passive items may be combined to a single one if there is a production that licenses it. The characteristic of the naïve strategy is that this combination of parse items is performed stepwise: it starts by predicting an active parse item from a production such that no right-hand side nonterminal is fixed. Then the variables of one nonterminal are replaced by the strings found in a passive parse item for this nonterminal. This process of *completing* the active item is repeated until every right-hand side nonterminal has been fixed. Now the result is converted into a passive item and may be used for completing other active items. Once the process of conversion and completion is saturated it is checked whether there is a parse item for the start nonterminal and the entire input string. If this is the case, then the string was recognized. In the following we describe this process formally.

To this end, let Σ be an alphabet, $G = (N, S, \Sigma, \varphi, P)$ be an LCFRS, $w \in \mathrm{T}_\Sigma^*$, and $n = |\operatorname{pos}(w)|$.

Definition 5.1. A *range of* w is a pair of integers $\langle l, r\rangle \in ([n]_0)^2$ such that $l \leq r$. Let Δ be an alphabet such that $\Delta = \{\langle l, r\rangle \mid 0 \leq l \leq r \leq n\}$. We define the *$w$-instantiation function* $\cdot^w \colon \Sigma \to \mathfrak{P}(\Delta)$:

$$\alpha \mapsto \{\langle i-1, i\rangle \in ([n]_0)^2 \mid w(i\varepsilon) = \alpha\}.$$

We denote $\cdot^w(\alpha)$ also by α^w and lift $\cdot^w$ to $\mathrm{T}_\Sigma^*(X) \to \mathfrak{P}(\mathrm{T}_\Delta^*(X))$ in the obvious way. If $s \in \mathrm{T}_\Sigma^*(X)$ and $d \in \mathrm{T}_\Delta^*(X)$ such that $d \in s^w$, then we say that d is a *w-instantiation of s*. Moreover, we define the

range-merging function $\mu\colon \mathrm{T}^*_\Delta(X) \to \mathfrak{P}(\mathrm{T}^*_\Delta(X))$ recursively such that

$$d \mapsto \begin{cases} \{(x) \diamond d'' \mid d'' \in \mu(d')\} & \text{if } \exists d' \in \mathrm{T}^*_\Delta\colon \exists x \in X\colon d = (x) \diamond d', \\ \{(\langle l_1, r_1\rangle) \diamond (x) \diamond d'' \mid d'' \in \mu(d')\} & \text{if } \exists d' \in \mathrm{T}^*_\Delta\colon \exists x \in X\colon d = (\langle l_1, r_1\rangle) \diamond (x) \diamond d', \\ \mu((\langle l_1, r_2\rangle) \diamond d') & \text{if } \exists d' \in \mathrm{T}^*_\Delta\colon d = (\langle l_1, r_1\rangle) \diamond (\langle l_2, r_2\rangle) \diamond d' \text{ and } r_1 = l_2, \\ \{(\langle l, r\rangle)\} & \text{if } d = (\langle l, r\rangle), \\ \{()\} & \text{if } d = (), \text{ and} \\ \emptyset & \text{otherwise.} \end{cases}$$

We lift μ to $\bar{\mu}\colon \mathfrak{P}(\mathrm{T}^*_\Delta(X)) \to \mathfrak{P}(\mathrm{T}^*_\Delta(X))$ such that $D \mapsto \bigcup_{d \in D} \mu(d)$. We say that two ranges $\langle l_1, r_1\rangle$ and $\langle l_2, r_2\rangle$ *overlap* if $l_1 \leq l_2 < r_1$ or $l_2 \leq l_1 < r_2$. Finally, we define the *w-instantiation of a word-tuple function* $(s_1, \ldots, s_k)$ *for* w as

$$\langle s_1, \ldots, s_k\rangle^w = \left\{ \langle s'_1, \ldots, s'_k\rangle \in (\mathrm{T}^*_\Delta(X))^k \;\middle|\; \begin{array}{l} \forall i \in [k]\colon s'_i \in \bar{\mu}(s_i^w) \text{ and} \\ \text{the occurrences of ranges in } \langle s'_1, \ldots, s'_k\rangle \\ \text{are pairwise non-overlapping} \end{array} \right\}. \qquad \blacksquare$$

In the above definition the instantiation function $\cdot^w$ replaces terminal occurrences in s-terms by ranges of w that span the same terminal. We capture the nondeterminism in the replacement by sets: there may be multiple occurrences of the same terminal or no occurrence at all. In the former case we obtain a set with multiple ranges. In the latter case the set is empty. The range-merging function μ transforms an s-term $d \in \mathrm{T}^*_\Delta(X)$ such that multiple ranges in consecutive positions are merged into one range. In the process it is also checked that merged ranges are adjacent, i.e., $\langle l_1, r_1\rangle$ and $\langle l_2, r_2\rangle$ may only be merged to $\langle l_1, r_2\rangle$ if $r_1 = l_2$. In case of failure $\mu(d)$ is the empty set. The instantiation of a word-tuple function $\langle s_1, \ldots, s_k\rangle$ amounts to w-instantiation and subsequent range-mering of each s-terms s_i, such that the results are pairwise non-overlapping.

Using the notion of w-instantiation we describe the naïve parser as a deduction system [SSP95] with active and passive items and three kinds of rules.

1. The **Predict**-rule creates a new active item.

$$\textbf{Predict}\ \frac{\big(A(s_{1,k}) \to B_1(x_{1,m_1}) \cdots B_n(x_{m_{n-1}+1,m_n})\big) \in P}{[A \to \bullet B_1 \cdots B_n; \phi]} \quad \phi \in \langle s_{1,k}\rangle^w$$

2. In the **Convert**-rule, a completely recognized active item is converted into a passive one.

$$\textbf{Convert}\ \frac{[A \to B_1 \cdots B_n \bullet; \phi]}{[A; \phi]}$$

   The next rule will imply that each component of ϕ consists of a single range of w if Convert is applicable.

3. The **Complete**-rule stepwise completes an active item by replacing the variables belonging to the i-th right-hand side nonterminal by ranges of a passive item for this nonterminal. By application of μ it is enforced that the inserted ranges are adjacent with the surrounding ones.

$$\textbf{Complete}\ \frac{[A \to B_1 \cdots B_{i-1} \bullet B_i\, B_{i+1} \cdots B_n; \langle \phi_1, \ldots, \phi_{\varphi(A)}\rangle], [B_i, \langle \psi_1, \ldots, \psi_{\varphi(B_i)}\rangle]}{[A \to B_1 \cdots B_i \bullet B_{i+1} \cdots B_n; \phi']}$$

   where

$$\phi' \in \{\langle \phi'_1, \ldots, \phi'_{\varphi(A)}\rangle \mid \forall l \in [\varphi(A)]\colon \phi'_l \in \mu(\phi_l[x_j^{(i)}/\psi_j \mid j \in [\varphi(B_i)]])\}$$

The above rules are used in a bottom-up parsing algorithm which we describe below. During its execution parse items are stored in a processing queue Q and a table T.

1. For every production with rank 0 all Predict-rules are performed and the resulting active items are added to Q and T.
2. While Q is non-empty do the following.
   a) Pop an item I from Q.
   b) If I is active and can be converted to $[A, \phi]$ and $[A, \phi]$ is not in T, then add $[A, \phi]$ to T and Q.
   c) If I is active and cannot be converted, then try to complete I with every passive item $[B_i, \psi]$ in T. If a Complete-step was successful and yielded an item I' not already present in T, then I' is added to T and Q.
   d) If $I = [A, \phi]$ is a passive item, then apply the Predict-rule for every production whose first right-hand side nonterminal is A and add each resulting active item I' to Q and T if I' is not in T. Add also every active item in T to Q, if its next unfixed nonterminal is A.
3. If $[S, \langle\langle 0, |w|\rangle\rangle]$ was derived, then there exists a successful parse.

So far we described only a recognizer for $\mathcal{L}(G)$. It can be easily extended to a parser generating derivation trees: if we obtain a passive item by conversion or an active item by completion, we keep a trace to the predecessor items in the table. By allowing for multiple traces for one item we preserve the ambiguity of the grammar.

As yet, we completely excluded the productions' probabilities: they are only considered after successful unweighted parsing. Before searching the most probable parse we compute the maximal probability for each entry I of T. To this end, we search the best parse of I's predecessors first. We avoid infinite loops which could arise from chain productions by setting the best weight to 0 when reaching an entry for the first time. This value gets updated by the best weight obtained by multiplication of the weight of the production and the weights of all predecessor. By remembering the items that contributed to the best weight we can finally construct the derivation tree.

Example 5.2. In Figure 5.3 the proof tree π for parsing the string $(\mathbf{P}, \mathbf{M}, \mathbf{h}, \mathbf{l})$ with the string component of the LCFRS/sDCP-hybrid grammar in 3.26 is depicted. Of course we deindexed the LCFRS productions before using them for parsing.

The derivation tree ξ that corresponds to this proof can be constructed by traversing the proof tree. From each Predict-rule we can read off the hybrid production and a reindexing function where the latter is uniquely determined by the used instantiation of the word-tuple function. For instance, during the application of the Predict-rule for $[r_4, r_4']$ the only terminal occurrence $\mathbf{l}^{\boxed{1}}$ in the word-tuple function is instantiated by the range $\langle 3, 4\rangle$. Thus, the reindexing for this production must be $\{1 \mapsto 4\}$. Moreover, the structure of ξ is obtained by collapsing Combine-rules and Convert-rules. Thus, in case of our example we obtain the derivation tree depicted in Figure 3.7. ■

It is obvious that the described parsing algorithm has much potential for optimization. For instance, since we are only interested in the best parse, one could consider weights much earlier in the parsing process. We may use a priority queue to greedily expand the best item found so far. Thus, we are able to maintain completeness but hopefully prune large parts of the search space.

Also, we do not account for special properties of the induced LCFRS. For instance, LCFRSs with fanout 1 are context-free grammars and, thus, one could apply more efficient parsing strategies in this case [Ear70; Tom87; Tom85]. Furthermore, if the hybrid grammar was induced using the left-branching or the right-branching recursive partitioning strategy, then its LCFRS component generates a regular language. An optimal parse may be found in time linear in the length of w with an algorithm similar to the *Viterbi algorithm* [Vit67]. This method could be refined by using a greedy strategy inspired by the *Dijkstra algorithm* [Dij59].

5.4.8 `/parser/sDCPevaluation`

To evaluate the sDCP components of a derivation tree we can use *treewalk attribute evaluators* that have been described for attribute grammars [KW76]. Basically, such an evaluator computes the value

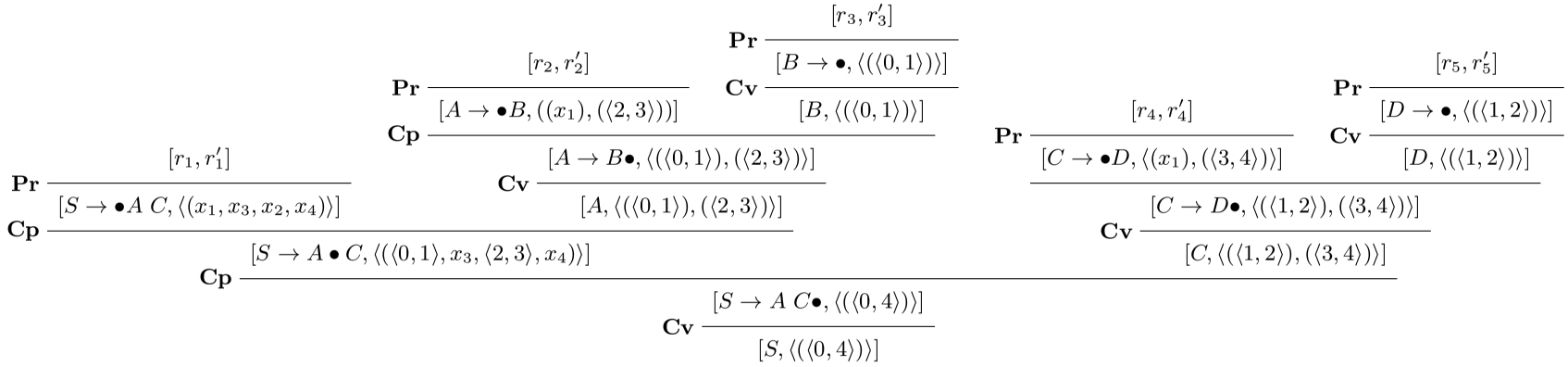


Figure 5.3: A proof tree for LCFRS parsing. We abbreviated **Predict**, **Convert**, and **Complete** by **Pr**, **Cv**, and **Cp**, respectively.

of the s-term corresponding to the only synthesized attribute in the word-tuple functions of the root by traversing the derivation tree. Whenever a variable occurs in the s-term, the value of this variable needs to be computed. Thus, the evaluator walks to the child, the sibling, or the parent node in the derivation tree that defines the value of this variable. Since we only consider non-circular sDCPs, such a traversal terminates and is correct.[3] Note also that the reindexing functions need to be applied to any index symbol in order to establish the synchronization with the terminals in the string.

Now we specify the treewalk attribute evaluator formally. Let $G = ((N_1, S_1, \Sigma, \varphi), (N_2, S_2, \Gamma, \iota, \sigma), P)$ be an LCFRS/sDCP-hybrid grammar and $\Delta = \Gamma \setminus \Sigma$. Also, let $\xi \in \mathrm{T}_{P_\mathcal{R}}$ be a derivation tree, $p \in \mathrm{pos}(\xi)$, $\mathrm{sDCP}(\xi(p))$ the sDCP function of the hybrid production at $\xi(p)$, $\tilde{\mathrm{s}}_p$ the signature of $\mathrm{sDCP}(\xi(p))$, and $\mathcal{R}_p$ the reindexing function at $\xi(p)$. Let $p \in \mathrm{pos}(\xi)$ and $\langle s^{(0)}_{1,\sigma_0}, s^{(1)}_{1,\iota_1}, \dots s^{(n)}_{1,\iota_n}\rangle = \mathrm{sDCP}(\xi(p))$. For every (s-)term $s \in \{s^{(i)}_j|_q \mid (i,j) \in \mathcal{S}(\tilde{\mathrm{s}}_p),\ q \in \mathrm{pos}(s^{(i)}_j)\}$, we define $[\![s]\!]^p_\xi$ recursively as follows:

$$s \mapsto \begin{cases} [\![s_1]\!]^p_\xi \diamond \cdots \diamond [\![s_n]\!]^p_\xi & \text{if } s = (s_1, \dots, s_n) \in \mathrm{T}^*_{\mathcal{I}(\mathbb{N},(\Sigma,\mathrm{rk}_\Gamma),(\Delta,\mathrm{rk}_\Gamma))}(X), \\ (\delta([\![s']\!]^p_\xi)) & \text{if } s = \delta(s') \text{ and } \delta \in \Delta^{(1)}, \\ (\delta) & \text{if } s = \delta \in \Delta^{(0)}, \\ (\mathcal{R}_p(\sigma^{\boxed{j}})([\![s']\!]^p_\xi)) & \text{if } s = \sigma^{\boxed{j}}(s') \text{ and } \sigma^{\boxed{j}} \in \mathcal{I}(\mathbb{N}, (\Sigma, \mathrm{rk}_\Gamma))^{(1)} \\ (\mathcal{R}_p(\sigma^{\boxed{j}})) & \text{if } s = \sigma^{\boxed{j}} \text{ and } \sigma^{\boxed{j}} \in \mathcal{I}(\mathbb{N}, (\Sigma, \mathrm{rk}_\Gamma))^{(0)} \\ \left[\!\!\left[s^{(0)}_j \right]\!\!\right]^{p\,i}_\xi & \text{if } s = x^{(i)}_j,\ i > 0, \text{ and } \mathrm{sDCP}(\xi(p\,i)) = \langle s^{(0)}_{1,\sigma_0}, s^{(1)}_{1,\iota_1}, \dots, s^{(n)}_{1,\iota_n}\rangle \\ \left[\!\!\left[s^{(i')}_j \right]\!\!\right]^{p'}_\xi & \text{if } s = x^{(i)}_j,\ i = 0,\ p = p'\,i', \text{ and } \mathrm{sDCP}(\xi(p')) = \langle s^{(0)}_{1,\sigma_0}, s^{(1)}_{1,\iota_1}, \dots, s^{(n)}_{1,\iota_n}\rangle. \end{cases}$$

Let now $\mathrm{sDCP}(\xi(\varepsilon)) = \langle s^{(0)}_{1,\sigma_0}, s^{(1)}_{1,\iota_1}, \dots, s^{(n)}_{1,\iota_n}\rangle$, where we know from Definition 3.20 that $\sigma_0 = 1$. We set t to $\left[\!\!\left[s^{(0)}_1 \right]\!\!\right]^\varepsilon_\xi$. Thus, we can define the *result of LCFRS parsing and sDCP evaluation* to be the canonical indexed pair of s-terms $[s,t]$ where s is such that $\mathrm{pos}(s) = \mathrm{pos}(w)$ and, for every $p \in \mathrm{pos}(s)$, we have that $s(p) = w(p)^{\boxed{p}}$.

Example 5.3. The evaluation of the derivation tree of the ongoing example is as follows:

$$\begin{aligned} t = \left[\!\!\left[x^{(1)}_1 \right]\!\!\right]^\varepsilon_\xi &= \left[\!\!\left[\mathbf{h}^{\boxed{1}}((x^{(1)}_1, x^{(0)}_1)) \right]\!\!\right]^1_\xi \\ &= \mathbf{h}^{\boxed{3}} \left(\left[\!\!\left[(x^{(1)}_1) \right]\!\!\right]^1_\xi \diamond \left[\!\!\left[x^{(0)}_1 \right]\!\!\right]^1_\xi \right) && \mathcal{R}_1 = \{1 \mapsto 3\} \\ &= \mathbf{h}^{\boxed{3}} \left(\left[\!\!\left[\mathbf{P}^{\boxed{1}}(()) \right]\!\!\right]^{11}_\xi \diamond \left[\!\!\left[x^{(2)}_1 \right]\!\!\right]^\varepsilon_\xi \right) \\ &= \mathbf{h}^{\boxed{3}} \left((\mathbf{P}^{\boxed{1}}(())) \diamond \left[\!\!\left[\mathbf{l}^{\boxed{1}}((x^{(1)}_1)) \right]\!\!\right]^2_\xi \right) && \mathcal{R}_{11} = \{1 \mapsto 1\} \\ &= \mathbf{h}^{\boxed{3}} \left((\mathbf{P}^{\boxed{1}}(())) \diamond \left(\mathbf{l}^{\boxed{4}} \left(\left[\!\!\left[x^{(1)}_1 \right]\!\!\right]^2_\xi \right) \right) \right) && \mathcal{R}_2 = \{1 \mapsto 4\} \\ &= \mathbf{h}^{\boxed{3}} \left((\mathbf{P}^{\boxed{1}}(())) \diamond \left(\mathbf{l}^{\boxed{4}} \left(\left[\!\!\left[\mathbf{M}^{\boxed{1}}(()) \right]\!\!\right]^{21}_\xi \right) \right) \right) \\ &= \mathbf{h}^{\boxed{3}} \left(\left(\mathbf{P}^{\boxed{1}}(()), \quad \mathbf{l}^{\boxed{4}} \left(\mathbf{M}^{\boxed{2}}(()) \right) \right) \right) && \mathcal{R}_{21} = \{1 \mapsto 2\}. \end{aligned}$$

Observe that our evaluator visits the root node of the tree twice. This is due to information passing from its second child to its first child. Pairing t with $s = (\mathbf{P}^{\boxed{1}}, \mathbf{M}^{\boxed{2}}, \mathbf{h}^{\boxed{3}}, \mathbf{l}^{\boxed{4}})$ yields the canonically indexed pair of s-terms $[s,t]$ that is depicted with its corresponding hybrid tree in Example 3.24. ■

[3] The evaluator also terminates if it is applied to a derivation tree with circular flow of information. This is because the single syntactic use requirement of sDCPs prevents the evaluator from entering any cycle.

We implemented this evaluator using a visitor pattern [Gam+95] and the prepared interface described in Section 5.4.3.

5.4.9 `/evaluation`

The package `evaluation` contains the essential infrastructure for running and evaluating the experiments in the dependency parsing scenario. Its first module is a database that stores the hybrid trees of the corpus, the parameterization of grammar induction, properties of the induced grammar, and the parsing results. The package's second module is a wrapper that exports the gold standard and the system output for subsets of the test corpus and runs the official scoring tool `eval.pl`[4] of the CoNLL-X shared task. Given a set of parameterizations it is possible to compute scores only on the intersection of recognized sentences or to define some fall-back strategy in case of failure. Also, one can restrict the test corpus to sentences of some maximal length during scoring. A third module allows for the generation of compact LaTeX-tables from the database. In fact, the tables below (cf. Section 5.5) were plotted using this module.

5.4.10 `/util`

The directory `util` contains scripts from the CoNLL-X shared task. Among others, the implementation of UAS, LAS, and LA, called `eval.pl`, and format verifiers for the CoNLL format are located here.

5.4.11 `/`

At the root of the project we located the scripts that a user has to run to start or evaluate an experiment. In the following we describe the scripts and how they can be used.

1. To run a constituent parsing experiment one first needs to specify the location of the corpora and the size of training and test section in `experiment.py`. Also, one needs to select the desired parameterizations by uncommenting the appropriate lines. Afterwards one runs
   ```
   python experiment.py
   ```
   and results are printed to `stdout`.

2. The parameterization of a dependency parsing experiment is specified by a configuration file of the following form, where the text in serif typeface indicates the default values in optional lines.

   ```
   #                              This is a comment
   Database:                      path/to/experiment-db
   Training Corpus:               path/to/training/corpus
   Test Corpus:                   path/to/test/corpus
   Nonterminal Labeling:          child-cpos+deprel
   Terminal Labeling:             pos
   Recursive Partitioning:        left-branching
   Training Limit:                1000                    default: none
   Test Limit:                    200                     default: none
   Test Length Limit:             25                      default: none
   # Pre/Post-processing options
   Default Root DEPREL:           ROOT                    default: none
   Ignore Punctuation:            NO # YES or NO          default: NO
   Default Disconnected DEPREL:   PUNC                    default: _
   ```

   The `Limit` options can be used to restrict the experiment to a sub-set of the training and test corpus. Using the above configuration only the first 1000 sentences of the training corpus are used for training and from the first 200 sentences of the test corpus only those with length smaller or equal than 25 are used for testing. The `Default ROOT DEPREL`-option can be used to overwrite the DEPRELS of every root of the hybrid tree after parsing. If `Ignore Punctuation` is activated, then

[4] `eval.pl` can be downloaded at `http://ilk.uvt.nl/conll/software.html#eval`.

punctuation is removed from the hybrid trees before induction and parsing. It is reinserted into the system's output by connecting it to the first root with the DEPREL `Default Disconnected DEPREL`. Note that disconnecting punctuation is experimental and may lead to errors if it used with dependency structures where punctuation has children that are non-punctuation.

The dependency parsing experiment can be started by running
`python dependency_experiments_db.py run-experiment path/to/config/file`.
This will either create a new database or append new data to the existing one. Also, some statistics are printed to `stdout`, where the plotted scores include also punctuation symbols.

3. To list each experiment in the database with basic information on the used corpus, the parameterization, and the start time, run `python evaluate path/to/database list`. To generate a LaTeX-document that contains a table with rows for a selection of experiments in the database run the script
`python evaluate.py path/to/database plot -experiments=$SELECTION -outfile=path/to/tex/file`.
Here `$SELECTION` is a string over positive integers, `-`, and `,` that specifies a selection of parameterizations of the database. For instance, `3-5,1,19` will include a row for the 3rd, 4th, 5th, 1st, and 19th parameterization in the database in exactly this order. The option `-max-sentence-length=$N` can be used score only sentences with length $\leq$ `$N`.

   The execution of the above command will generate a table with the following columns: recursive partitioning, terminal labeling strategy, nonterminal labeling strategy, number of nonterminals with fanout 1 and fanout 2, UAS and LAS including punctuation, UAS, LAS, and LA excluding punctuation, and the parsing time. A selection of columns from the command line interface is currently not implemented. A different selection of columns and additional ones are available by modifying the python script `table_plotting.py` in the package `evaluation`.

5.4.12 Remarks on the programming language

The dynamically typed scripting language `python` [Ros95] is well-suited for rapid prototyping. It unites features from different families of programming languages: imperative and procedural statements, classes and inheritance, as well as lambda-abstractions and build-in higher order functions on lists.

There is, however, an inherent problem for our use case: the language is interpreted and not compiled to a binary. This gives a great penalty on running time and memory consumption since certain optimizations present in modern compilers cannot be performed. Also, type-checking is at run-time and, thus, some type-related errors in the current code base might be undetected since certain run-time configurations have not occurred so far.

A more powerful type system would also allow for a more sophisticated design of hybrid trees and grammars: these classes could be parameterized by a concrete instance of `MonadicToken`. Currently, the correct token type is just a matter of convention.

Given the size of the corpora involved and the general complexity of LCFRS parsing, a programming language that is directly compiled to machine code is desirable to keep the constant factor in the running time as low as possible. Alternatively, one may also try to use the existing state-of-the-art LCFRS parsers in `rparse` or the `GrammaticalFramework`.[5] The appeal of this approach could be the reimplementation of grammar induction and sDCP evaluation in a strongly-typed, functional language like Haskell, while the running time intense parsing task (which requires various kinds of low-level optimization) might be better solved in an imperative, system-oriented language. On the other hand, this decomposition imposes requirements to the interface of the external parser: presumably, we want to incorporate filtering and features on the input and certain fall-back strategies if parsing fails that still utilize successful parses of substrings.

[5] The grammatical framework is a programming language for the writing of grammars. It features an LCFRS parser and can be downloaded at `http://www.grammaticalframework.org/`.

5.5 Experiments

In this section we describe the experiments that we carried out with the implementation described above. We used a server with 64GB of RAM and two `2.6GHz Intel Xeon CPU E5-2630` CPUs with six cores each. The server's operating system was `Ubuntu 14.04` and we used Python in version 2.7.6. The implementation of the LCFRS parser does not feature parallelization. Still, some parameterizations were executed at the same time which could cause small distortions in the measured running time.

5.5.1 Results for dependency parsing

We carried out experiments for labeled dependency parsing only, i.e., we aim to correctly determine the HEAD and the DEPREL for each CoNLL token in a sequence. In this process, we tried to run a broad range of the parameterization options described in Section 5.4.5. However, parsing LCFRSs with high fanouts has an immense running time and early tests indicated that the improvement with a higher fanout is very small in most cases. Moreover, these tests indicated that certain parameterizations would result in too many parse failures to be relevant. Thus, we restricted our attention to an interesting subset of parameterizations that depends on the corpus in use.

As each corpus contains a significant amount of unseen forms (and lemmas if available), we used either POS or CPOS (if distinct) as terminal labels. Also, since parameterizations with CPOS as terminal labels turned out to perform worse than those with POS, we skipped many experiments with CPOS. The results of the experiments are reported in Tables 5.4, 5.5, 5.6, 5.7, and 5.8. We grouped experiments according to their nonterminal labeling strategies. Each row describes one parameterization and there are columns for a relevant subset of the following values where the selection depends on the corpus: the recursive partitioning strategy, the terminal labeling strategy, the argument label for each position in the nonterminal label, the number of nonterminals with a certain fanout, and the number of parse failures. Furthermore, we report the UAS, LAS, and LA, each with and without punctuation, and the parsing time. All scores and the parsing time concern only the recognized sentences with the respective parameterization, i.e., an experiment with many parse failures can have high scores and a low parsing time which only reflect easy sentences.

In case of TIGER we used the whole CoNLL-X version of the corpus for training and in all but one experiment the sentences with length ≤ 20 for testing. We used POS as terminal labels and set the DEPREL of the determined root to ROOT as a post-processing step. For NEGRA we used only sentences of length ≤ 25 in both training and testing. Again, terminal labels were POS in every configuration. For Turkish and Slovene we used also CPOS-tags for the labeling of terminals and nonterminal. In case of Turkish the DEPREL of each root was set to ROOT during post-processing. The small training corpus of both Turkish and Slovene induced a grammar sufficiently small to run the experiment without length restriction if the grammar's fanout was 1. We included a second table for Turkish (cf. Table 5.8) that also features parameterizations with $k = 2$, i.e., transformation of the recursive partitioning directly extracted to one with fanout 2, however, the sentence length was limited to 20 for testing.

In the following we analyze different phenomena, starting by general observations made with all corpora. Afterwards we will point out special characteristics in the results of each corpus. Whenever possible, we try to isolate single aspects of parameterization.

- In all corpora child labeling leads to fewer parse failures than strict labeling keeping the other parameters fixed. Thus, the desired higher coverage of child labeling is obtained.
- Based on the experiments with TIGER and SDT, where different configurations have a reasonable good coverage, we reckon the following. If only DEPRELs are used as argument labels, then the label accuracy is higher compared to sole POS. On the contrary, using only POS seems to lead to higher UAS than DEPREL alone. Using the DEPREL alone outperforms the POS when it comes to LAS. Although these effects could also be caused by the favorable ratio of DEPRELs and POS in TIGER (46 vs. 52) and SDT (28 vs. 25), a connection between DEPREL, i.e., syntactic function, and correct attachment seems also linguistically plausible. Unsurprisingly, at the price of more parse failures the best scores are obtained if POS and DEPREL are annotated . However,

the experiments with the Turkish treebank do not support this theory as the measured scores show only low fluctuation for the different labeling strategies.

- Strict labeling can result in a number of nonterminals that is an order of magnitude higher than those obtained with child labeling. The better coverage of child labeling resembles this trend.
- The number of nonterminals (and productions) has a substantial impact on the measured parsing time: a very high number causes a high constant in the parsing complexity. For instance, the number of nonterminals in the SDT experiments for child labeling with DEPREL as argument label is smaller than in the experiments for TIGER with similar parameterizations. Since the coverage is equally good in both cases and the average sentence length in TIGER is even smaller, the lower parsing times for SDT support our claim.
- Observe how the recursive partitioning strategy influences the number of nonterminals: the left-branching and the right-branching recursive partitioning lead to the highest number of nonterminals. Also, a more rigorous fanout-reduction raises the number of nonterminals. Thus, a very low number of nonterminals tends to coincide with a higher combinatoric potential of the productions which has a positive influence on coverage but increases the parsing time as well.
- In theory the choice of the recursive partitioning strategy is one of the main driver for parsing complexity. The measured parsing times reflect this correlation to the fanout of the LCFRS although there are exceptions. The parsing times measured for the left-branching and the right-branching recursive partitioning are sometimes higher than those for $k = 1$. Moreover, there are two parameterizations of TIGER where $k = 2$ is faster than $k = 1$, namely, strict labeling with POS+DEPREL or POS as argument labels.
- The scores of the fanout-k parameterizations for different k do not differ significantly for TIGER. In Turkish the positive impact is more substantial. There is no clear indicator that a lower fanout is worse, although the higher fanout influences the coverage positively. Left-branching and right-branching reach lower scores than $k = 1$ and have a worse coverage.
- There can be serious differences between the left-branching and the right-branching recursive partitioning in the number of nonterminals and the measured scores, coverage, and parsing times. The picture is favorable for the left-branching strategy with any of the used corpora. In TIGER the parsing time is lower, the scores are slightly better, the number of nonterminals is marginally lower, but the coverage is similar. The results for NEGRA are comparable to that of TIGER, however, for child labeling with DEPREL the parsing time with the left-branching recursive partitioning is higher than with the right-branching one. In Turkish the scores show a low level of fluctuation, however, the coverage with the left-branching recursive partitioning is much better. This holds especially for strict labeling with CPOS as argument labels. The trends for Slovene follow those of the other corpora but are less appreciable.
- Both the experiments for Turkish and Slovene indicate that the use of CPOS instead of POS as terminal label or argument label improves the coverage but reduces the scores.

For TIGER strict labeling with DEPREL and $k = 1$, left-branching, or right-branching recursive partitioning, and child labeling with POS+DEPREL and $k = 1$ turned out to be the best choices. The obtained scores are below the best numbers reported in the CoNLL-X shared task for this data set, but they are close to the average results. Recall however, that we imposed a length restriction due to the computational costs, which was not the case for the competitors in the CoNLL-X shared task. Therefore, we included one TIGER experiment without length-restrictions which indicates a moderate decrease in the scores compared to the length-limited experiment with equal parameterization.

For NEGRA the best results are obtained using child labeling with DEPREL and $k = 1$. The scores are slightly better than those reported for the LCFRS-based system by [MK10] but they are inferior to the scores of MSTParser, too. We did not include results for strict labeling, as the numbers of parse failures were too high to be of interest, e.g., already 57 parse failures in the case of DEPREL and $k = 1$.

For Turkish the scores obtained with the different parameterizations differ only slightly. A good coverage is only obtained with child labeling or strict labeling with DEPREL and any recursive partitioning strategy, or with strict labeling with CPOS and $k = 1$. Interestingly, child labeling with DEPREL and $k = 1$ has a slightly higher LA and LAS than strict labeling with DEPREL and $k = 1$, whereas the latter has the better UAS. Strict labeling with CPOS and $k = 1$ has an even better UAS. Our system significantly misses both the average and the overall scores of the CoNLL-X shared task.

The best coverage for Slovene is obtained with strict or child labeling with DEPREL. The best scores are reached with child labeling with CPOS and $k = 1$. Again we miss the reference scores significantly.

5.5.2 Results for constituent parsing

The results are displayed in Table 5.9 and allow for similar conclusions as in [NV14]. Unsurprisingly, the larger training set leads to smaller proportions of parse failures and to improvements of F-measures. Another consequence is that the more fine-grained strict labeling now outperforms child labeling.

5.6 Discussion

The experiments suggest that the developed induction method for LCFRS/sDCP-hybrid grammars has the potential to be used in practice. The desired separation of parsing complexity and accuracy by different recursive partitioning strategies was confirmed in our experiments, although the variations tend to be relatively small. But this is in fact an advantageous observation as it favors the use of hybrid grammars that are efficiently parseable but still sufficiently accurate.

Clearly, the scores achieved with the current implementation are lower than those of state-of-the-art dependency parsers, but, for TIGER and NEGRA they are not too far away from them either. One reason for the superior performance of the existing systems is that they use the available input information more exhaustively. Normally, most of the input fields in a CoNLL token are taken into consideration whereas we only use POS or CPOS. As mentioned in Chapter 2.3, the POS-tag often does not capture every syntactic feature of a word. Thus, it seems natural that a parser reasonably making use of these features, outperforms our approach. The second advantage of the existing approaches is their usually linear or quadratic running time. In contrast, we have a very high demand of computational resources that is due to the higher parsing complexity of LCFRSs and the inefficient implementation.

Finally, it should be noted that using PARSEVAL and comparable metrics for comparing the performance of parsers on different treebanks has been criticized [RG07]. Also, one can question the relevance of LAS, UAS, and LA in a non-projective parsing scenario. Due to the relatively low amount of discontinuous tokens a high (but imperfect) score might also be achieved by a model not even capable of representing discontinuity at all. Thus, [Mai+14a] compiled a test suite containing samples for several kinds of grammatical constructions in German that are discontinuous/non-projective. Since this is a qualitative test that requires linguistic interpretation of data, we leave such an analysis of the hybrid grammar approach open for future investigation.

	argument labels	f 1	f 2	lim	fail	UAS	LAS	LA	secs
strict labeling									
$k = 1$	POS+DEPREL	129,013	0	20	65	89.4	85.5	90.1	2,431
$k = 2$	POS+DEPREL	86,945	33,929	20	57	88.5	84.3	89.2	2,054
r-branch	POS+DEPREL	293,179	0	20	132	92.3	89.0	91.8	3,923
l-branch	POS+DEPREL	296,093	0	20	131	93.5	89.7	92.2	811
$k = 1$	POS	85,618	0	20	28	84.4	66.5	74.6	4,048
$k = 2$	POS	54,437	22,666	20	26	84.1	66.0	74.9	3,471
r-branch	POS	227,695	0	20	112	89.7	69.5	74.0	3,390
l-branch	POS	231,595	0	20	111	90.5	70.5	74.7	855
$k = 1$	DEPREL	51,660	0	20	3	85.4	80.0	87.1	14,225
$k = 2$	DEPREL	31,792	15,569	20	3	85.4	79.9	87.0	53,796
r-branch	DEPREL	154,639	0	20	3	85.0	78.8	85.9	12,817
l-branch	DEPREL	147,439	0	20	3	85.2	79.0	85.9	9,292
child labeling									
$k = 1$	POS+DEPREL	21,158	0	20	6	88.0	82.1	87.4	2,242
$k = 1$	POS+DEPREL	21,158	0	∞	6	85.7	80.4	87.0	13,029
$k = 2$	POS+DEPREL	8,270	4,123	20	3	87.8	81.8	87.2	6,207
r-branch	POS+DEPREL	93,856	0	20	39	86.5	80.4	86.7	2,077
l-branch	POS+DEPREL	89,999	0	20	31	86.0	79.8	86.0	780
$k = 1$	POS	7,302	0	20	2	84.6	65.7	74.0	1,800
$k = 2$	POS	1,605	1,616	20	2	84.7	66.2	74.2	13,368
r-branch	POS	53,384	0	20	24	84.2	65.5	73.5	1,362
l-branch	POS	48,672	0	20	19	85.1	66.9	74.4	862
$k = 1$	DEPREL	5,193	0	20	0	78.3	69.9	80.5	13,343
r-branch	DEPREL	40,037	0	20	0	75.6	66.8	79.0	12,906
l-branch	DEPREL	35,362	0	20	0	76.4	67.7	79.2	30,290

Table 5.4: Experiments for dependency parsing on TIGER: label information per argument position, number of nonterminals with fanout 1, number of nonterminals with fanout 2, limit on sentence length, number of parse failures, unlabeled attachment score, labeled attachment score, label accuracy, and parsing time.

	arg. labels	f 1	f 2	fail	UASp	LASp	UAS	LAS	LA	secs
child labeling										
$k = 1$	P+D	20,887	0	378	88.0	82.9	86.3	80.4	85.0	4,431
$k = 2$	P+D	8,626	8,160	283	86.6	81.3	84.8	78.7	83.8	21,214
r-branch	P+D	43,889	0	857	88.4	83.0	86.6	80.4	84.3	6,609
l-branch	P+D	41,206	0	834	87.8	82.4	85.9	79.5	83.5	1,709
$k = 1$	P	7,025	0	83	85.1	68.7	83.3	64.2	71.4	5,717
$k = 2$	P	1,832	2,879	39	84.6	68.2	82.8	63.8	71.3	120,925
r-branch	P	21,672	0	465	84.5	68.1	82.1	63.0	70.8	4,070
l-branch	P	19,292	0	426	84.6	68.2	82.3	63.3	71.1	2,348
$k = 1$	D	13,266	0	1	80.8	74.5	78.7	71.4	78.5	38,981
r-branch	D	29,462	0	1	77.2	70.0	75.0	66.5	75.4	36,149
l-branch	D	27,396	0	1	77.8	70.5	75.5	67.1	75.7	44,492

Table 5.5: Experiments for dependency parsing on NEGRA. Columns are as in Table 5.4. Additional columns present unlabeled and labeled attachment scores that include punctuation (UASp and LASp, respectively).

	term.	argument labels	f 1	fail	UAS	LAS	LA	secs
strict labeling								
$k = 1$	POS	CPOS+DEPREL	9,389	119	68.0	54.4	68.5	886
r-branch	POS	CPOS+DEPREL	16,116	332	82.7	71.3	79.1	620
l-branch	POS	CPOS+DEPREL	15,310	338	90.3	79.8	84.3	156
$k = 1$	CPOS	CPOS+DEPREL	9,389	59	63.5	47.8	60.9	1,474
$k = 1$	POS	DEPREL	6,679	0	68.3	53.7	66.8	4,455
r-branch	POS	DEPREL	12,622	5	62.6	46.8	59.6	1,748
l-branch	POS	DEPREL	11,730	9	62.1	47.1	59.9	2,820
$k = 1$	POS	POS	7,100	76	68.3	45.1	60.0	917
$k = 1$	POS	CPOS	5,126	18	69.6	45.0	60.1	1,727
child labeling								
$k = 1$	POS	POS+DEPREL	3,803	34	73.4	58.5	70.6	1,325
r-branch	POS	POS+DEPREL	8,651	217	73.9	59.7	70.9	390
l-branch	POS	POS+DEPREL	8,685	206	71.1	57.0	69.9	362
$k = 1$	CPOS	POS+DEPREL	3,803	6	66.1	50.0	62.5	3,168
$k = 1$	POS	CPOS+DEPREL	2,845	5	73.2	57.8	70.4	2,934
r-branch	POS	CPOS+DEPREL	7,367	49	66.8	52.4	65.4	506
l-branch	POS	CPOS+DEPREL	7,068	28	68.8	52.9	65.7	1,761
$k = 1$	CPOS	CPOS+DEPREL	2,845	5	66.2	50.2	63.3	3,402
$k = 1$	POS	DEPREL	1,370	0	67.5	53.4	67.6	11,801
r-branch	POS	DEPREL	5,171	0	66.4	51.3	65.3	2,944
l-branch	POS	DEPREL	4,587	0	67.0	52.2	66.5	10,720
$k = 1$	POS	POS	1,462	13	73.4	48.2	61.8	1,639

Table 5.6: Experiments for dependency parsing on the Slovene Dependency Treebank. Columns are as in Table 5.4.

	term.	argument labels	f 1	fail	UASp	LASp	UAS	LAS	LA	secs
strict labeling										
$k = 1$	POS	CPOS+DEPREL	13,842	37	73.2	58.9	60.1	42.3	54.5	3,978
r-branch	POS	CPOS+DEPREL	25,655	199	76.1	60.4	62.9	42.9	54.4	2,107
l-branch	POS	CPOS+DEPREL	24,448	193	75.8	60.4	62.6	42.4	54.1	1,389
$k = 1$	CPOS	CPOS+DEPREL	13,842	29	71.9	56.3	58.8	40.5	52.5	4,871
$k = 1$	POS	DEPREL	7,937	0	69.7	55.8	56.7	39.8	54.1	10,247
r-branch	POS	DEPREL	15,479	12	71.1	55.9	58.9	39.4	53.3	3,770
l-branch	POS	DEPREL	15,340	8	72.1	57.0	59.1	39.3	53.0	10,167
$k = 1$	POS	POS	8,688	25	71.2	52.5	60.7	39.4	54.1	3,780
$k = 1$	POS	CPOS	5,660	8	72.0	53.0	61.0	39.4	54.2	4,145
r-branch	POS	CPOS	12,706	128	74.8	55.9	62.5	40.9	55.2	1,350
l-branch	POS	CPOS	11,486	61	72.0	53.5	61.2	39.7	54.3	852
child labeling										
$k = 1$	POS	POS+DEPREL	5,130	26	72.8	59.0	59.4	43.0	55.5	2,672
r-branch	POS	POS+DEPREL	10,472	70	70.4	55.2	57.5	38.7	52.1	1,149
l-branch	POS	POS+DEPREL	9,560	61	71.5	56.2	58.8	39.5	52.8	2,937
$k = 1$	CPOS	POS+DEPREL	5,130	12	71.4	56.0	58.3	39.9	51.7	4,395
$k = 1$	POS	CPOS+DEPREL	3,364	14	73.2	59.5	59.6	43.0	55.9	3,666
r-branch	POS	CPOS+DEPREL	7,716	36	70.5	55.6	57.3	39.0	52.1	1,236
l-branch	POS	CPOS+DEPREL	7,098	34	71.2	56.2	58.4	39.7	52.9	3,966
$k = 1$	CPOS	CPOS+DEPREL	3,364	11	71.7	57.0	58.9	41.7	53.3	4,338
$k = 1$	POS	DEPREL	1,288	0	69.8	55.7	56.1	39.9	53.8	10,721
r-branch	POS	DEPREL	3,537	0	67.8	52.5	54.7	36.6	51.8	2,959
l-branch	POS	DEPREL	3,469	0	70.0	54.4	55.7	36.9	51.7	14,619
$k = 1$	POS	POS	1,347	12	73.6	53.4	62.0	40.5	55.5	2,644
r-branch	POS	POS	4,417	53	73.0	53.6	60.7	40.5	55.6	596
l-branch	POS	POS	3,565	39	72.1	53.2	61.1	40.5	55.7	1,823

Table 5.7: Experiments for dependency parsing on the METU-Sabanci Turkish Treebank. Columns are as in Table 5.5.

	term.	argument labels	f 1	f 2	fail	UAS	LAS	LA	secs
strict labeling									
$k = 1$	POS	CPOS+DEPREL	13,842	0	32	64.9	46.2	56.1	1,861
$k = 2$	POS	CPOS+DEPREL	7,834	5,058	25	63.5	44.5	55.5	5,630
l-branch	POS	CPOS+DEPREL	24,448	0	146	65.3	44.7	55.2	699
r-branch	POS	CPOS+DEPREL	25,655	0	150	66.2	45.6	55.3	1,401
$k = 1$	POS	DEPREL	7,937	0	0	62.1	44.0	55.8	4,512
$k = 2$	POS	DEPREL	4,246	3,005	0	63.2	45.0	57.0	21,515
r-branch	POS	DEPREL	15,479	0	9	63.5	43.5	55.4	2,447
l-branch	POS	DEPREL	15,340	0	6	63.7	43.5	55.0	4,230
$k = 1$	POS	CPOS	5,660	0	7	65.7	42.8	55.2	1,726
$k = 2$	POS	CPOS	2,899	2,062	10	64.8	41.9	54.5	8,787
l-branch	POS	CPOS	11,486	0	52	65.5	43.4	55.8	396
r-branch	POS	CPOS	12,706	0	96	65.9	44.2	56.0	894
child labeling									
$k = 1$	POS	POS+DEPREL	5,130	0	24	64.7	46.7	57.1	1,123
$k = 2$	POS	POS+DEPREL	2,537	1,141	20	64.7	46.1	56.5	10,500
r-branch	POS	POS+DEPREL	10,472	0	61	61.7	41.9	53.4	739
l-branch	POS	POS+DEPREL	9,560	0	51	62.7	42.2	53.6	1,310
$k = 1$	POS	CPOS+DEPREL	3,364	0	13	65.2	47.6	58.3	1,457
$k = 2$	POS	CPOS+DEPREL	1,637	801	12	65.5	47.2	57.4	18,786
r-branch	POS	CPOS+DEPREL	7,716	0	33	62.0	42.6	54.0	785
l-branch	POS	CPOS+DEPREL	7,098	0	29	63.1	43.4	54.8	1,694
$k = 1$	POS	DEPREL	1,288	0	0	62.4	44.8	55.8	3,868
$k = 2$	POS	DEPREL	476	323	0	64.3	47.0	57.6	107,819
r-branch	POS	DEPREL	3,537	0	0	59.5	41.0	54.2	1,753
l-branch	POS	DEPREL	3,469	0	0	60.8	41.7	54.4	6,198
$k = 1$	POS	POS	1,347	0	11	66.6	43.6	56.4	978
$k = 2$	POS	POS	459	316	10	68.4	44.8	56.7	19,351
r-branch	POS	POS	4,417	0	46	65.2	43.6	56.3	385
l-branch	POS	POS	3,565	0	32	65.3	43.6	56.2	773

Table 5.8: Experiments with the METU-Sabanci Turkish Treebank featuring also the $k = 2$ recursive partitioning. Only sentences of length ≤ 20 were used for testing. Columns are as in Table 5.4.

	fail	R	P	F1	# cons. str.	secs
strict labeling						
$k = 1$	11	77.3	77.3	76.5	1.0133	113,905
$k = 2$	9	77.6	77.9	77.0	1.0135	1,692,216
r-branch	325	67.0	63.5	64.6	1.0121	127,645
l-branch	1,841	65.6	58.3	60.8	1.0118	5,429
child labeling						
$k = 1$	1	75.3	74.9	74.5	1.0141	124,910
r-branch	23	75.5	74.8	74.5	1.0132	37,570
l-branch	79	75.6	75.3	75.0	1.0123	39,082

Table 5.9: Number of parse failures, recall, precision, F-measure, average number of consecutive strings per constituent, and running time.

6 Related work

This short chapter gives an overview of related work on constituent parsing and dependency parsing.

6.1 Dependency parsing

Much progress was made in syntactic dependency parsing in the first decade of the 21st century in the surrounding of the CoNLL shared-tasks of the years 2006 and 2007 [BM06; Niv+07]. Essentially, there are two classes of dependency parsers: either transition-based models or graph-based models are used to represent (non-projective) dependency structures. In the following we outline the central ideas of both approaches. Moreover, we give a summary of the less examined area of grammar-based models where also hybrid grammars are located.

6.1.1 Transition-based models

The idea of transition based parsing is reading the input stepwise and composing it to a dependency structure. Thus, a dependency structure is modeled by a sequence of parser actions (or transitions). The action of the parser depends on certain characteristics of the current input token, of its surrounding in the sentence, the partially built structure, and, lastly, on previous parser actions.

We briefly sketch the stack-based approach by [Niv09] as an example for transition based parsing. There are two stacks in this approach: one holds the input tokens and is called *input* in the following whereas the other serves as work space and is called *stack*. Depending on the mentioned characteristics of the input and stack configuration, one of the following actions is triggered:

- an arc between the two top-most stack tokens is created (in either direction) and the dominated node is removed from the stack, or
- the top-most elements on the stack are swapped, or
- the top-most element is popped from the input and pushed onto the stack.

Due to swapping on the stack non-projective structures can be obtained. Training of this model amounts to feature selection and the learning of linear classifiers or support vector machines. Clearly, a big advantage is the inclusion of various kinds of input features and the expected linear run-time of the parser.[1] This and similar parsing strategies were implemented in `maltparser` [NHN06]. Transition-based parsers have been extended successfully with beam search strategies [ZC08] and neural networks for the trigging of parser actions [CM14].

6.1.2 Graph-based models

The idea of graph-based models is to learn which dependency structure is good among all possible dependency structures for a sentence. As an example, we briefly describe the maximum spanning tree approach by [McD+05]. Let w be a sequence or sentence of tokens. Let $G = (V, E)$ be a directed graph such that V contains the set T of tokens of w and a virtual root node v_0 and $E = (\{v_0\} \times T) \cup (T \times T)$, i.e., G is complete except for v_0 not having incoming edges. Further, let f be a weight function with the signature $f\colon E \to \mathbb{R}$. A spanning tree of G is a subgraph $t = (V', E')$ of G such that $V = V'$ and t is

[1] Note that due to swapping on the stack a quadratic running time is possible, although linear running time is measured in practice [Niv09].

a tree. We assign t the weight $\sum_{e \in E'} f(e)$. The maximum spanning tree G is the spanning tree with maximal weight.

The set of spanning trees of G and the dependency structures of w are isomorphic. There are algorithms with running time in $\mathcal{O}(n^3)$ for finding the maximum spanning trees of a directed graph [CL65; Edm67], which can be refined to algorithms with a running time in $\mathcal{O}(n^2)$ if the graph is dense [Tar77]. Thus, one can train a weight function f and run one of these algorithms for dependency parsing. The results obtained with graph-based approaches are comparable to those obtained with transition-based models. The basic model has been extended to support higher-order dependencies [Koo+10; ZM12], i.e., dependencies not only between parent and child but also between two siblings, parent and grand-child, etc.

6.1.3 Grammar-based models

The attempts to describe non-projective dependency structures with formal grammars are relatively new. This is because existing grammar formalisms were either not expressive enough or lacked in efficient parser implementations. Due to the advent of more efficient LCFRS parsers [BL05; KM09; KM15; Mai15] and the study of LCFRS binarization [GS09] and restrictions like well-nestedness [GKS10] and fanout constraints [MKK12] the use of grammar-based parsing approaches has become a more realistic objective. Alternatively, *tree adjoining grammars* have been proposed as a plausible model for dependency structures [KK12].

A system based on LCFRSs for parsing non-projective dependency structures is described in [Kuh13] and implemented by [MK10]. In this approach, the grammar's derivation trees initially resemble the dependency structure with nonterminals being DEPRELs. The running time of the parser and the generalization of the grammar are improved by binarization and Markovization, respectively. Experimental results are reported for NEGRA and the Prague Dependency Treebank [Haj+00].

6.2 Constituent parsing

In the last decade of the 20th century sentences from the Wall Street Journal with over 4.5 million words were syntactically annotated [MSM94]. The availability of this corpus of constituent trees known as *Penn treebank* fostered the research on statistical parsing with context-free grammars [Col97; Cha00]. Soon, corpora for other languages like German [Sku+98; Bra+02] or Chinese [XCP02] were created. Among the techniques that have been developed to improve both coverage and accuracy of the grammars are lexicalization [SW93], Markovization [KM03], latent annotation of nonterminals [MMT05], and automated splitting and merging of nonterminals [Pet+06; PK07]. These techniques have been implemented in the `stanford parser`[2] and the `berkeley parser`.[3]

This research focused mainly on parsing English, however other languages with free-word order like German stimulated research on automated learning of grammars capable of representing discontinuity. First theoretical work was by [Pla04] and [MS08] and implementations and refinements of the latter LCFRS-based approach followed with [MK10; MKK12; KM15; Mai15].

[2]The `stanford parser` is available at `http://nlp.stanford.edu/software/lex-parser.shtml`.
[3]The `berkeley parser` is available at `http://github.com/slavpetrov/berkeleyparser`.

7 Conclusions and outlook

We showed that discontinuous constituent structures and non-projective dependency structures can be represented as hybrid trees. We considered hybrid grammars, which generate languages of hybrid trees, and presented methods to learn a probabilistic LCFRS/sDCP-hybrid grammar G from a corpus of constituent structures or a corpus of dependency structures. We demonstrated that G can afterwards be used to parse unseen sentences and produce their constituent or dependency structure. The required time for parsing can be adjusted by the recursive partitioning strategy that was chosen during grammar induction.

We implemented hybrid trees, hybrid grammars, grammar induction, and parsing and tested our syntax model on four language corpora. One of the key insights is that the desired decoupling of the expressiveness and the parsing complexity of the hybrid grammar is achieved: using an LCFRS with small fanout as string component in the hybrid grammar allows us to reduce the parsing time. Nonetheless, the loss in accuracy and coverage is small. Although the experimental results yield that our prototype is not competitive with state-of-the-art systems its accuracy is lower and it consumes much time and memory the scores indicate that the hybrid grammar approach works in principle. Also, the comparison with state-of-the-art parsers suggests that a broad range of improvements can be incorporated into the framework of hybrid trees and hybrid grammars.

- The good coverage obtained with POS as terminals in the LCFRS is payed with accuracy. A sensible inclusion of lexical information with suitable fall-back strategies in case of unknown words should improve the accuracy while maintaining the coverage. Additional morphological information might be useful as well [Mai+14b].
- In the Markovization technique by [KM03] the width of the horizontal and vertical context that is encoded in the nonterminal label can be adjusted. Perhaps the context information encoded in our labeling strategies can be extended and adjusted similarly.
- It can be examined whether the techniques for latent annotation [MMT05] and automated splitting and merging [Pet+06] of nonterminals can be transferred to hybrid grammars.
- We weight our grammar by plain relative frequency estimation. The inside-outside variant of the expectation-maximization algorithm has been used to train probabilistic context-free grammars [Bak79; LY90; Pre01]. Perhaps these algorithms can be adopted for hybrid grammars to obtain better weight-distributions.
- The implementation of the LCFRS parser can be either improved or replaced by an existing, more efficient implementation.
- We manually ran many experiments in order to find good combinations of strategies for terminal labeling, nonterminal labeling, and recursive partitioning. Automating this process is desirable.

Moreover, we specified Grammar induction in a more formal way than [NV14]. Our presentation also captures corner cases of hybrid trees like multiple nodes at root level and empty categories in constituent structures. However, this level of detail also reveals that sDCPs are quite bulky in constructions and proofs. Thus, for future theoretic work the author suggests the following.

- There might exist a class of higher-order grammars that is equivalent to (non-circular) sDCPs and more suitable for formal considerations. For instance, *2nd-order abstract categorial grammars* [Gro01] have been shown to have the same string-generating power as LCFRSs [GP04], but can also be used to model tree grammars. One should consider the replacement of sDCPs by this or a similar formalism.

- The rank of an LCFRS affects its parsing complexity. Can the binarization algorithm for LCFRSs [GS09] be transfered to LCFRS/sDCP-hybrid grammars?
- Hybrid grammars that combine a string grammar and a tree grammar different from LCFRSs and sDCPs can be explored and compared w.r.t. their expressiveness and parsing complexity.
- Certain classes of tree transducers [FMV11; Ost14] and synchronous grammars [Shi04; Shi06; NV12] can be described by a regular tree grammar and two tree transformations [AD82]. Since our conception of the derivation trees of hybrid grammars is that of trees over an infinite ranked set (the infinity is due to the reindexing), we cannot characterize this tree language by a tree grammar. However, if the synchronization of nonterminals is encoded differently, then one might obtain a similar *bitransformation* result for hybrid grammars. A promising perspective for such an endeavor might be that of graph grammars [Ehr+06].

Bibliography

[AD82] A. Arnold and M. Dauchet. Morphismes et Bimorphismes d'Arbres. *Theoretical Computer Science* 20(1), (1982), 33–93.

[Bak79] J. Baker. Trainable grammars for speech recognition. In: *Speech Communication Papers for the 97th Meeting of the Acoustical Society of America*. 1979, pp. 547–550.

[Ben+15] E. M. Bender, D. Flickinger, S. Oepen, W. Packard, and A. Copestake. Layers of Interpretation: On Grammar and Compositionality. In: *Proceedings of the 11th International Conference on Computational Semantics*. 2015, pp. 239–249.

[Ben13] E. M. Bender. "Linguistic Fundamentals for Natural Language Processing: 100 Essentials from Morphology and Syntax". Ed. by G. Hirst. Morgan & Claypool Publishers, 2013.

[BL05] H. Burden and P. Ljunglöf. Parsing linear context-free rewriting systems. In: *Proceedings of the Ninth International Workshop on Parsing Technology*. 2005.

[Bla+91] E. Black, S. P. Abney, D. Flickinger, C. Gdaniec, R. Grishman, P. Harrison, D. Hindle, R. Ingria, F. Jelinek, J. Klavans, M. Liberman, M. P. Marcus, S. Roukos, B. Santorini, and T. Strzalkowski. A Procedure for Quantitatively Comparing the Syntactic Coverage of English Grammars. In: *Proceedings of the Workshop on Speech and Natural Language*. 1991, pp. 306–311.

[BM06] S. Buchholz and E. Marsi. CoNLL-X Shared Task on Multilingual Dependency Parsing. In: *Proceedings of the Tenth Conference on Computational Natural Language Learning*. 2006, pp. 149–164.

[Boj+14] O. Bojar, C. Buck, C. Federmann, B. Haddow, P. Koehn, J. Leveling, C. Monz, P. Pecina, M. Post, H. Saint-Amand, R. Soricut, L. Specia, and A. Tamchyna. Findings of the 2014 Workshop on Statistical Machine Translation. In: *Proceedings of the Ninth Workshop on Statistical Machine Translation*. 2014, pp. 12–58.

[Bou05] P. Boullier. Range Concatenation Grammars. *New Developments in Parsing Technology*. Text, Speech and Language Technology. 2005, 269–289.

[Bra+02] S. Brants, S. Dipper, S. Hansen, W. Lezius, and G. Smith. The TIGER Treebank. In: *Proceedings of the 2nd International Conference on Language Resources and Evaluation*. 2002, pp. 24–41.

[Büc15] M. Büchse. "Algebraic decoder specification: coupling formal-language theory and statistical machine translation". PhD thesis. Technische Universität Dresden, 2015.

[Cha00] E. Charniak. A Maximum-entropy-inspired Parser. In: *Proceedings of the 1st North American Chapter of the Association for Computational Linguistics Conference*. 2000, pp. 132–139.

[Cho56] N. Chomsky. Three models for the description of language. *IRE Transactions on Information Theory* 2(3), (1956), 113–124.

[Cho59] N. Chomsky. On certain formal properties of grammars. *Information and Control* 2(2), (1959), 137–167.

[Cho65] N. Chomsky. "Aspects of the theory of syntax". MIT Press, 1965.

[CL65] Y.-J. Chu and T.-H. Liu. On shortest arborescence of a directed graph. *Scientia Sinica* 14(10), (1965), 1396–1400.

[CM14] D. Chen and C. Manning. A Fast and Accurate Dependency Parser using Neural Networks. In: *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing*. 2014, pp. 740–750.

[Col97] M. Collins. Three Generative, Lexicalised Models for Statistical Parsing. In: *Proceedings of the Eighth Conference on European Chapter of the Association for Computational Linguistics*. 1997, pp. 16–23.

[CS04] A. Culotta and J. Sorensen. Dependency Tree Kernels for Relation Extraction. In: *Proceedings of the 42nd Annual Meeting on Association for Computational Linguistics*. 2004.

[CS63] N. Chomsky and M.-P. Schützenberger. The algebraic theory of context-free languages. *Computer Programming and Formal Systems*. 1963, 118–161.

[CS70] J. Cocke and J. T. Schwartz. *Programming languages and their compilers*. Tech. rep. Courant Institue of Mathematical Sciences, New York University, 1970.

[Dij59] E. W. Dijkstra. A note on two problems in connection with graphs. *Numerische Mathematik* 1(1), (1959), 269–271.

[DP05] Y. Ding and M. Palmer. Machine Translation Using Probabilistic Synchronous Dependency Insertion Grammars. In: *Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics*. 2005, pp. 541–548.

[De+06] S. Deroski, T. Erjavec, N. Ledinek, P. Pajas, Z. abokrtsky, and A. ele. Towards a Slovene dependency treebank. In: *Proceedings of the International Conference on Language Resources and Evaluation*. 2006.

[Ear70] J. Earley. An Efficient Context-free Parsing Algorithm. *Communications of the ACM* 13(2), (1970), 94–102.

[Edm67] J. Edmonds. Optimum branchings. *Journal of Research of the National Bureau of Standards* 71B(4), (1967), 233–240.

[Ehr+06] H. Ehrig, K. Ehrig, U. Prange, and G. Taentzer. "Fundamentals of Algebraic Graph Transformation". Springer-Verlag New York, Inc., 2006.

[ES77] J. Engelfriet and E. M. Schmidt. IO and OI. I. *Journal of Computer and System Sciences* 15(3), (1977), 328–353.

[Fea01] S. Featherston. "Empty Categories in Sentence Processing". John Benjamins Publishing Company, 2001.

[FMV11] Z. Fülöp, A. Maletti, and H. Vogler. Weighted Extended Tree Transducers. *Journal Fundamenta Informaticae* 111(2), (2011), 163–202.

[Fow26] H. W. Fowler. "A dictionary of modern english usage". Oxford University Press, 1926.

[Gam+95] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. "Design Patterns: Elements of Reusable Object-oriented Software". Addison-Wesley Longman Publishing Co., Inc., 1995.

[Gie88] R. Giegerich. Composition and evaluation of attribute coupled grammars. *Acta Informatica* 25, (1988), 355–423.

[Gil10] D. 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*. 2010, pp. 769–776.

[GKS10] C. Gómez-Rodrguez, M. Kuhlmann, and G. Satta. Efficient Parsing of Well-Nested Linear Context-Free Rewriting Systems. In: *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the ACL*. 2010, pp. 276–284.

[Gor67] S. Gorn. Explicit definitions and linguistic dominoes. In: *Computer and System Sciences*. 1967, pp. 77–115.

[GP04] P. de Groote and S. Pogodalla. On the Expressive Power of Abstract Categorial Grammars: Representing Context-Free Formalisms. *Journal of Logic, Language and Information* 13(4), (2004), 421–438.

[Gro01] P. de Groote. Towards Abstract Categorial Grammars. In: *Proceedings of the 39th Annual Meeting on Association for Computational Linguistics*. 2001, pp. 252–259.

[GS09] C. Gómez-Rodrguez and G. Satta. An Optimal-Time Binarization Algorithm for Linear Context-Free Rewriting Systems with Fan-Out Two. In: *Proceedings of the 47th Annual Meeting of the Association for Computational Linguistics and the 4th International Joint Conference on Natural Language Processing of the AFNLP*. 2009, pp. 985–993.

[Haj+00] J. Haji, A. Böhmová, E. Hajiová, and B. Vidová-Hladká. The Prague Dependency Treebank: A Three-Level Annotation Scenario. *Treebanks: Building and Using Parsed Corpora*. 2000, 103–127.

[Hin+04] E. Hinrichs, S. Kübler, K. Naumann, H. Telljohann, J. Trushkina, et al. Recent developments in linguistic annotations of the TüBa-D/Z treebank. In: *Proceedings of the Third Workshop on Treebanks and Linguistic Theories*. 2004, pp. 51–62.

[Jou84] M. Jourdan. Strongly Non-circular Attribute Grammars and Their Recursive Evaluation. In: *Proceedings of the 1984 SIGPLAN Symposium on Compiler Construction*. 1984, pp. 81–93.

[Kal10] L. Kallmeyer. "Parsing Beyond Context-Free Grammars". Springer Publishing Company, Inc., 2010.

[Kas65] T. Kasami. *An efficient recognition and syntaxanalysis algorithm for context-free languages*. Tech. rep. Air Force Cambridge Research Laboratories, 1965.

[KK12] L. Kallmeyer and M. Kuhlmann. A formal model for plausible dependencies in lexicalized tree adjoining grammar. In: *Proceedings of the Eleventh International Conference on Tree Adjoining Grammars and Related Formalisms*. 2012, pp. 108–116.

[KM03] D. Klein and C. D. Manning. Accurate Unlexicalized Parsing. In: *Proceedings of the 41st Annual Meeting on Association for Computational Linguistics*. 2003, pp. 423–430.

[KM09] L. Kallmeyer and W. Maier. An Incremental Earley Parser for Simple Range Concatenation Grammar. In: *Proceedings of the 11th International Conference on Parsing Technologies*. 2009, pp. 61–64.

[KM15] L. Kallmeyer and W. Maier. LR Parsing for LCFRS. In: *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. 2015, pp. 1250–1255.

[Knu68] D. E. Knuth. Semantics of context-free languages. *Mathematical Systems Theory* 2(2), (1968), 127–145.

[Koo+10] T. Koo, A. M. Rush, M. Collins, T. Jaakkola, and D. Sontag. Dual Decomposition for Parsing with Non-projective Head Automata. In: *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing*. 2010, pp. 1288–1298.

[KSK06] Y. Kato, H. Seki, and T. Kasami. Stochastic multiple context-free grammar for RNA pseudoknot modeling. In: *Proceedings of the Eighth International Workshop on Tree Adjoining Grammar and Related Formalisms*. Association for Computational Linguistics. 2006, pp. 57–64.

[Kuh13] M. Kuhlmann. Mildly Non-projective Dependency Grammar. *Computational Linguistics* 39(2), (2013), 355–387.

[KW76] K. Kennedy and S. K. Warren. Automatic Generation of Efficient Evaluators for Attribute Grammars. In: *Proceedings of the 3rd ACM SIGACT-SIGPLAN Symposium on Principles on Programming Languages*. 1976, pp. 32–49.

[Lop08] A. Lopez. Statistical Machine Translation. *ACM Computing Surveys* 40(3), (2008), 8:1–8:49.

[LY90] K. Lari and S. Young. The estimation of stochastic context-free grammars using the Inside-Outside algorithm. *Computer Speech and Language* 4, (1990), 35–56.

[Mai+14a] W. Maier, M. Kaeshammer, P. Baumann, and S. Kübler. Discosuite - A parser test suite for German discontinuous structures. In: *Proceedings of the Ninth International Conference on Language Resources and Evaluation*. 2014, pp. 2905–2912.

[Mai+14b] W. Maier, S. Kübler, D. Dakota, and D. Whyatt. Parsing German: How Much Morphology Do We Need? In: *Proceedings of the First Joint Workshop on Statistical Parsing of Morphologically Rich Languages and Syntactic Analysis of Non-Canonical Languages*. 2014, pp. 1–14.

[Mai15] W. Maier. Discontinuous Incremental Shift-reduce Parsing. In: *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing*. 2015, pp. 1202–1212.

[Mar+14] M.-C. de Marneffe, T. Dozat, N. Silveira, K. Haverinen, F. Ginter, J. Nivre, and C. D. Manning. Universal Stanford dependencies: A cross-linguistic typology. In: *Proceedings of the Ninth International Conference on Language Resources and Evaluation*. 2014, pp. 4585–4592.

[McC82] J. D. McCawley. Parentheticals and discontinuous constituent structure. *Linguistic Inquiry*, (1982), 91–106.

[McD+05] R. McDonald, F. Pereira, K. Ribarov, and J. Haji. Non-projective Dependency Parsing Using Spanning Tree Algorithms. In: *Proceedings of the Conference on Human Language Technology and Empirical Methods in Natural Language Processing*. 2005, pp. 523–530.

[MK10] W. Maier and L. Kallmeyer. Discontinuity and Non-Projectivity: Using Mildly Context-Sensitive Formalisms for Data-Driven Parsing. In: *Proceedings of the Tenth International Conference on Tree Adjoining Grammars and Related Formalisms*. 2010, pp. 119–126.

[MKK12] W. Maier, M. Kaeshammer, and L. Kallmeyer. Data-Driven PLCFRS Parsing Revisited: Restricting the Fan-Out to Two. In: *Proceedings of the Eleventh International Conference on Tree Adjoining Grammars and Related Formalisms*. 2012, pp. 126–134.

[MMT05] T. Matsuzaki, Y. Miyao, and J. Tsujii. Probabilistic CFG with latent annotations. In: *Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics*. 2005, pp. 75–82.

[MS08] W. Maier and A. Søgaard. Treebanks and Mild Context-Sensitivity. In: *Proceedings of the 13th Conference on Formal Grammar*. 2008, pp. 61–76.

[MSM94] M. P. Marcus, B. Santorini, and M. A. Marcinkiewicz. Building a Large Annotated Corpus of English: The Penn Treebank. *Computational Linguistics* 19(2), (1994), 313–330.

[NHN06] J. Nivre, J. Hall, and J. Nilsson. Maltparser: A data-driven parser-generator for dependency parsing. In: *Proceedings of the fifth international conference on Language Resources and Evaluation*. 2006, pp. 2216–2219.

[Niv+07] J. Nivre, J. Hall, S. Kübler, R. T. McDonald, J. Nilsson, S. Riedel, and D. Yuret. The CoNLL 2007 Shared Task on Dependency Parsing. In: *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*. 2007, pp. 915–932.

[Niv09] J. Nivre. Non-projective Dependency Parsing in Expected Linear Time. In: *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP*. 2009, pp. 351–359.

[NV12] M.-J. Nederhof and H. Vogler. Synchronous Context-Free Tree Grammars. In: *Proceedings of the 11th International Workshop on Tree Adjoining Grammars and Related Formalisms*. 2012, pp. 55–63.

[NV14] M.-J. Nederhof and H. Vogler. Hybrid Grammars for Discontinuous Parsing. In: *Proceedings of the 25th International Conference on Computational Linguistics*. 2014, pp. 1370–1381.

[Ofl+03] K. Oflazer, B. Say, D. Hakkani-Tür, and G. Tür. Building a Turkish Treebank. *Treebanks*. Text, Speech and Language Technology. 2003, 261–277.

[Ost14] J. Osterholzer. Pushdown Machines for Weighted Context-Free Tree Translation. In: *Proceedings of the 19th International Conference on Implementation and Application of Automata*. 2014, pp. 290–303.

[PDM12] S. Petrov, D. Das, and R. McDonald. A Universal Part-of-Speech Tagset. In: *Proceedings of the Eight International Conference on Language Resources and Evaluation*. 2012, pp. 2089–2096.

[Pet+06] S. Petrov, L. Barrett, R. Thibaux, and D. Klein. Learning Accurate, Compact, and Interpretable Tree Annotation. In: *Proceedings of the 21st International Conference on Computational Linguistics and the 44th Annual Meeting of the Association for Computational Linguistics*. 2006, pp. 433–440.

[PK07] S. Petrov and D. 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*. 2007, pp. 404–411.

[Pla04] O. Plaehn. New Developments in Parsing Technology. 2004. Chap. Computing the Most Probable Parse for a Discontinuous Phrase Structure Grammar, 91–106.

[Pre01] D. Prescher. Inside-Outside Estimation Meets Dynamic EM. In: *Proceedings of the 7th International Workshop on Parsing Technologies (IWPT-01), October 17-19*. 2001, pp. 241–244.

[RG07] I. Rehbein and J. V. Genabith. Evaluating evaluation measures. In: *Proceedings of the 16th Nordic Conference of Computational Linguistics*. 2007, pp. 372–379.

[Ros95] G. van Rossum. *Python Tutorial*. Tech. rep. CS-R 9526. Centrum Wiskunde & Informatica, 1995.

[Rou69] W. C. Rounds. Context-free grammars on trees. *Proceedings of the ACM Symp. on Theory of Computing*, (1969), 143–148.

[Sch+99] A. Schiller, S. Teufel, C. Stöckert, and C. Thielen. *Guidelines für das Tagging deutscher Textcorpora mit STTS*. Tech. rep. Universitäten Stuttgart und Tübingen, 1999.

[Sek+91] H. Seki, T. Matsumura, M. Fujii, and T. Kasami. On multiple context-free grammars. *Theoretical Computer Science* 88, (1991), 191–229.

[SFH04] H. Schmid, A. Fitschen, and U. Heid. SMOR: A German Computational Morphology Covering Derivation, Composition, and Inflection. In: *Proceedings of the 4th International Conference on Language Resources and Evaluation*. 2004, pp. 1263–1266.

[Shi04] S. Shieber. Synchronous grammars as tree transducers. In: *Proceedings of the Seventh International Workshop on Tree Adjoining Grammars and Related Formalisms*. 2004, pp. 88–95.

[Shi06] S. Shieber. Unifying synchronous tree-adjoining grammars and tree transducers via bimorphisms. In: *In Proceedings of the 11th Conference of the European Chapter of the Association for Computational Linguistics*. 2006, pp. 377–384.

[SJN05] R. Snow, D. Jurafsky, and A. Y. Ng. Learning Syntactic Patterns for Automatic Hypernym Discovery. In: *Advances in Neural Information Processing Systems*. 2005, pp. 1297–1304.

[SK08] H. Seki and Y. Kato. On the Generative Power of Multiple Context-Free Grammars and Macro Grammars. *IEICE - Transactions on Information and Systems* E91-D(2), (2008), 209–221.

[Sku+98] W. Skut, T. Brants, B. Krenn, and H. Uszkoreit. A Linguistically Interpreted Corpus of German Newspaper Text. In: *Proceedings of the Conference on Language Resources and Evaluation*. 1998, pp. 705–711.

[SP05] G. Satta and E. Peserico. Some Computational Complexity Results for Synchronous Context-free Grammars. In: *Proceedings of the Conference on Human Language Technology and Empirical Methods in Natural Language Processing*. 2005, pp. 803–810.

[SS90] S. M. Shieber and Y. Schabes. Synchronous tree-adjoining grammars. In: *Proceedings of the 13th International Conference on Computational Linguistics*. 1990, pp. 253–258.

[SSP95] S. M. Shieber, Y. Schabes, and F. C. Pereira. Principles and Implementation of Deductive Parsing. *Journal of Logic Programming* 24(1-2), (1995), 3–36.

[SW93] Y. Schabes and R. C. Waters. Lexicalized Context-free Grammars. In: *Proceedings of the 31st Annual Meeting on Association for Computational Linguistics*. 1993, pp. 121–129.

[SWB03] I. A. Sag, T. Wasow, and E. M. Bender. "Syntactic Theory: A Formal Introduction". Center for the Study of Language and Information, Stanford University, 2003.

[Sza12] Z. G. Szabó. *Compositionailty*. The Stanford Encyclopedia of Philosophy. 2012. URL: `http://plato.stanford.edu/entries/compositionality/`.

[Tar77] R. E. Tarjan. Finding optimum branchings. *Networks* 7(1), (1977), 25–35.

[Tes59] L. Tesnière. "Éléments de syntaxe structurale". Klincksieck, 1959.

[Tom85] M. Tomita. "Efficient Parsing for Natural Language: A Fast Algorithm for Practical Systems". Kluwer Academic Publishers, 1985.

[Tom87] M. Tomita. An Efficient Augmented-context-free Parsing Algorithm. *Comput. Linguist.* 13(1-2), (1987), 31–46.

[Vau68] B. Vauquois. A survey of formal grammars and algorithms for recognition and transformation in mechanical translation. In: *Proceedings of the Information Processing Congress 1968*. 1968, pp. 1114–1122.

[Vit67] A. J. Viterbi. Error Bounds for Convolutional Codes and an Asymptotically Optimum Decoding Algorithm. *IEEE Transactions on Information Theory* 13(2), (1967), 260–269.

[VWJ87] K. Vijay-Shanker, D. J. Weir, and A. K. Joshi. Characterizing Structural Descriptions Produced by Various Grammatical Formalisms. In: *Proceedings of the 25th Annual Meeting on Association for Computational Linguistics*. 1987, pp. 104–111.

[XCP02] N. Xue, F.-D. Chiou, and M. Palmer. Building a Large-scale Annotated Chinese Corpus. In: *Proceedings of the 19th International Conference on Computational Linguistics*. 2002, pp. 1–8.

[You67] D. H. Younger. Recognition and parsing of context-free languages in time n^3. *Information and Control* 10(2), (1967), 189–208.

[ZC08] Y. Zhang and S. Clark. A Tale of Two Parsers: Investigating and Combining Graph-based and Transition-based Dependency Parsing Using Beam-search. In: *Proceedings of the Conference on Empirical Methods in Natural Language Processing*. 2008, pp. 562–571.

[ZM12] H. Zhang and R. McDonald. Generalized Higher-order Dependency Parsing with Cube Pruning. In: *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*. 2012, pp. 320–331.