Technische Universität Dresden
Fakultät Informatik
Institut für Theoretische Informatik
Lehrstuhl Grundlagen der Programmierung

Bachelorarbeit

Split-Merge Training for Linear Context-free Rewriting Systems

Kevin Mitlöhner

Eingereicht am 26.06.2017
Bearbeitet vom 03.04.2017
bis zum 26.06.2017

Verantwortlicher Hochschullehrer:
Prof. Dr.-Ing. habil. Dr. h.c./Univ. Szeged Heiko Vogler
Betreuer:
Dipl.-Inf. Kilian Gebhardt

Aufgabenstellung für die Bachelorarbeit

„Split-Merge Training von LCFRS“

Technische Universität Dresden
Fakultät Informatik

Student:	Kevin Mitlöhner
Geburtsdatum:	21. Dezember 1995
Matrikelnummer:	4041754
Studiengang:	Bachelor Informatik
Immatrikulationsjahr:	2014
Studienleistung:	Bachelorarbeit
Beginn am:	3. April 2017
Einzureichen am:	26. Juni 2017
Verantw. Hochschullehrer:	Prof. Dr.-Ing. habil. Heiko Vogler
Betreuer:	Dipl.-Inf. Kilian Gebhardt

Ein aktueller Trend in den Forschungsgebieten des *Natural Language Processing (NLP)* und der *Machine Translation (MT)* ist der zunehmende Bezug auf die syntaktische, d.h. grammatikalische, Struktur der zu verarbeitenden Sprache. Diese wird oft durch eine kontextfreie Grammatik (CFG) angegeben [MS99; YK01] und kann dazu genutzt werden, einen vorgelegten Satz zu zerlegen und die Zerlegung durch einen *Phrasenstrukturbaum* (PSB) darzustellen, wobei die Wörter des Satzes an den Blättern stehen. Solche PSB sind kontinuierlich in dem Sinne, dass für jeden Teilbaum die Menge der durch ihn überdeckten Satzpositionen eine kontinuierliche Sequenz ist, d.h. ein Intervall. Bei Sprachen mit relativ freier Wortordnung (z.B. Deutsch oder Niederländisch) – aber nicht nur dort [KM13] – treten auch *diskontinuierliche* PSB auf [KP95; Mül04].

Linear context-free rewriting systems. Diskontinuierliche PSB sind nicht durch kontextfreie Grammatiken darstellbar. Um allerdings die hohe Parsingkomplexität kontextsensitiver Grammatiken (nämlich PSPACE-complete) zu vermeiden, betrachtet man Formalismen, die Diskontinuität zwar darstellen können, aber dennoch polynomiell parsbar sind. Man fasst solche Formalismen unter dem Begriff *mildly context-sensitive formalisms* zusammen. Dazu gehören z.B. head grammars, tree adjoining grammars, combinatory categorial grammars, linear indexed grammars, multiple context-free grammars, and minimalist grammars. *Linear context-free rewriting systems* (kurz: LCFRS) wurden von Vijay-Shanker, Weir und Joshi [VWJ87] eingeführt, um mildly context-sensitive formalisms einheitlich darzustellen. Es hat sich herausgestellt, dass alle oben genannten (und noch einige weitere) Formalismen bzw. deren Frontsprachen eine kleinere oder die gleiche Sprachklasse wie LCFRS erzeugen [Sek+91; VWJ86; WJ88; Vij87; Mic01a; Mic01b]. Das Parsing von LCFRS ist daher von besonderer Bedeutung für die Verarbeitung natürlicher Sprache [Eva11].

Gewichtete LCFRS. In der Verarbeitung natürlicher Sprache wird die Zugehörigkeit eines Wortes zu einer Sprache oftmals nicht als Wahrheitswert, sondern als reelle Zahl zwischen 0 und 1 formuliert. Man erhält damit einen *Grad der Zugehörigkeit* des Wortes zur Sprache. So können z.B. häufige von seltenen sprachlichen Konstruktionen unterschieden werden. Zur Gewichtung einer LCFRS können nicht nur reelle Zahlen, sondern auch andere Algebren verwendet werden; nötig ist dabei nur eine Operation zur *Multiplikation* der Regelgewichte in einem Parsebaum und eine Operation zur *Addition* der Gewichte verschiedener Parsebäume. Siehe dazu Goodman [Goo99] und Nederhof [Ned03].

Induktion und Training von LCFRS Eine LCFRS kann direkt aus einem Korpus über (potentiell) nichtkontinuierlichen Phrasenstrukturen abgelesen (induziert) werden. Eine derart gewonnene „treebank grammar“ kodiert bestimmte Kontextunabhängigkeitsannahmen: ob eine Regel anwendbar ist, hängt ausschließlich vom Nichtterminalsymbol aber nicht vom Kontext ab, in welchem das Nichtterminal auftaucht. Da die Anzahl der in den Korpora verwendeten Knotenbeschriftungen für syntaktische Kategorien eher klein ist, sind diese Unabhängigkeitsannahmen zu stark. Man würde zum Beispiel annehmen, dass *Nominalphrasen* (NP), die in Subjektpositionen vorkommen, genau so in Objektphrasen auftauchen.

Aus diesem Grund wurden Nichtterminale manuell um zusätzliche Informationen angereichert, sowohl aus dem syntaktischen [KM03] als auch aus dem lexikalen [Col03] Kontext. Dies erfordert einen hohes Maß an manueller Feinjustierung. Es ist jedoch vorstellbar, dass Unterteilungen von Nichtterminalen automatisch gelernt werden können. Dabei kann man zunächst von einer treebank grammar ausgehen und Regelgewichte mit Hilfe des *Expectation-Maximization* (EM) Algorithmus trainieren. Anschließend wird jedes Nichtterminal in jeweils zwei neue Nichtterminale aufgeteilt (*Splitting*) und unter Verwendung des bisherigen Trainingsergebnisses erneut trainiert. Danach können ggf. einzelne, wenig gewinnbringende Zustandsaufteilungen wieder zurückgenommen werden (*Merging*). Dieses Splitting und Merging kann so oft wiederholt werden (Zyklus), bis eine ausreichend gute Grammatik erzielt wurde.

Diese Idee des Splitting und Merging wurde für probabilistische kontextfreie Grammatiken mit latenten Annotationen (PCFG-LA) unter dem Namen *Split-Merge-Verfahren* umgesetzt [PK07; Pet+06; DE06; MMT05]. LCFRS lassen sich als Verallgemeinerung von

CFG auffassen, was eine Übertragung des Split-Merge-Verfahren auf LCFRS nahelegt. In der Tat wurde diese Übertragung auch schon vorgeschlagen [EK11], eine empirische Untersuchung erfolgte unseres Wissens aber nicht.

Aufgabe. Im Rahmen seiner Bachelor-Arbeit soll der Student folgende Aufgaben bearbeiten.

- Der Student soll LCFRS formal definieren.
- Er soll das Split-Merge-Verfahren für LCFRS formal beschreiben und die dafür benötigten Algorithmen darlegen.
- Der Student soll die Eignung des Split-Merge-Verfahren für LCFRS empirisch untersuchen. Dabei soll er wie folgt vorgehen:
  1. Aus einem Korpus soll mit Hilfe der Software `treetools`[1] eine binarisierte LCFRS extrahiert werden. Gegebenenfalls, kann diese LCFRS analog zu [Pet+06] (X-bar grammar) nachbearbeitet werden (Postprocessing).
  2. Das Split-Merge-Verfahren soll auf die LCFRS angewandt werden. Die praktisch durchführbare Zahl an Split-Merge-Zyklen soll ermittelt werden. Die Zahl der Epochen des im Split-Merge-Verfahren enthaltenen EM-Trainings soll variiert werden.
  3. Nach jedem Split-Merge-Zyklus soll die Güte der resultierenden LCFRS-LA für syntaktisches Parsing evaluiert werden. Dazu soll der LCFRS Parser von Angelov und Ljunglöf [AL14] auf die Development-Daten angewandt werden und die resultierenden Parsebäume anhand der PARSEVAL/EVALB-Metrik [Bla+91] mit den gold standard Annotationen verglichen werden.
  4. Sobald sinnvolle Größen für Zyklenzahl und EM Epochen bestimmt wurden, soll das Split-Merge-Verfahren und Parsing auf Development-Daten ausgehend von der Grammatik aus Schritt 1 mit vier verschiedenen Random Seeds wiederholt werden. (Unterschiedliche Random Seeds haben PCFG-LA mit signifikanten Abweichungen zur Folge [Pet10].) Die LCFRS-LA, welche auf den Development-Daten die besten Ergebnisse erreicht, soll anschließend mit den Test-Daten evaluiert werden.

  Diese Experimente sollen für mindestens zwei Korpora, die Diskontinuität enthalten, aus dem SPMRL Shared Task [SKT14] erfolgen.
- Grammatikextraktion, Split-Merge-Algorithmus, Parsing und Evaluation sollen aus **einem** parametrisierbarem Skript aufrufbar sein. Gegebenfalls notwendige Formatumwandlungen der LCFRS zwischen Extraktion, Training und Parsing sollen automatisch ohne Eingriff des Anwenders erfolgen. Mindestens folgende Parameter sollen gesetzt werden können: Ein- und Ausgabepfade für Korpora, Zwischenergebnisse und Grammatiken, Parameter der Grammatikinduktion von `treetools` sowie des Postprocessings, Anzahl der Split-Merge-Zyklen, Anzahl der

[1] `https://github.com/wmaier/treetools`

EM-Epochen je Split-Merge-Zyklus, Einschränkungen auf Teilkorpora für Training und Test (z.B. die Sätze 20-2000), sowie der Random Seed.

Eine Implementierung von LCFRS einschließlich Parsing und des Split-Merge-Algorithmus, die notwendige Rechenkazität sowie die relevanten Korpora werden dem Studenten zur Verfügung gestellt.

Form. Die Arbeit muss den üblichen Standards wie folgt genügen. Die Arbeit muss in sich abgeschlossen sein und alle nötigen Definitionen und Referenzen enthalten. Die Urheberschaft von Inhalten – auch die eigene – muss klar erkennbar sein. Fremde Inhalte, z.B. Algorithmen, Konstruktionen, Definitionen, Ideen, etc., müssen durch genaue Verweise auf die entsprechende Literatur kenntlich gemacht werden. Lange wörtliche Zitate sollen vermieden werden. Gegebenenfalls muss erläutert werden, inwieweit und zu welchem Zweck fremde Inhalte modifiziert wurden. Die Struktur der Arbeit muss klar erkenntlich sein, und der Leser soll gut durch die Arbeit geführt werden. Die Darstellung aller Begriffe und Verfahren soll mathematisch formal fundiert sein. Für jeden wichtigen Begriff sollen Erläuterungen und Beispiele angegeben werden, ebenso für die Abläufe der beschriebenen Verfahren. Wo es angemessen ist, sollen Illustrationen die Darstellung vervollständigen. Bei Diagrammen, die Phänomene von Experimenten beschreiben, muss deutlich erläutert werden, welche Werte auf den einzelnen Achsen aufgetragen sind, und beschrieben werden, welche Abhängigkeit unter den Werten der verschiedenen Achsen dargestellt ist.

Für die Implementierung soll eine ausführliche Dokumentation erfolgen, die sich angemessen auf den Quelltext und die schriftliche Ausarbeitung verteilt. Dabei muss die Funktionsfähigkeit des Programms glaubhaft gemacht und durch geeignete Beispielläufe dokumentiert werden.

Einer späteren Veröffentlichung der Implementierung unter einer Freien Software Lizenz stimmt der Student zu. Der Student verpflichtet sich, ihm im Rahmen dieser Arbeit zugänglich gemachte Daten und Software (einschließlich Quellcode) lediglich zur Erledigung der Aufgaben zu verwenden und ansonsten vertraulich zu behandeln.

Dresden, 16. März 2017

Unterschrift von Heiko Vogler

Unterschrift von Kevin Mitlöhner

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, 26. Juni 2017

Contents

1 Introduction

An important part of the natural language processing refers to the syntactic structure of the processed language. A given sentence can be partitioned and represented as a parse tree using, in many cases, a context free grammar (CFG). Parse trees produced by such a CFG are *continuous* which means each node covers a complete interval of positions of the sentence. However, there are languages like German, Polish and Swedish which cannot be adequately described by a CFG. They have a relatively free word order and certain cases whereby non-local dependencies arise. To represent these languages so-called *linear context-free rewriting systems* (LCFRS) introduced by Vijay-Shanker, Weir, and Joshi [VSWJ87] can be used. Parsing with this formalism produces potentially *discontinuous* parse trees, where nodes could cover several parts of the sentence such that there are further parts of the sentence between them. To obtain such an LCFRS from a corpus over (potentially) discontinuous phrase structures, the induction method introduced by Maier and Søgaard [MS08] can be used.

LCFRS are a generalization of CFG in that they allow for the description of discontinuous phrase structures. However, they inherit the context-free independence assumption of CFG. They model context-free dependencies in the sense that the application of a rule depends only on the nonterminals and not on the derivational context of the nonterminals. For example, we observe a nominal phrase in subject position and another time in object position, the derivation is completely independent of this observation. These strong independence assumptions are due to the small number of node labels given by the tree banks. One way to solve this problem is to refine the nonterminals by annotating additional information. This was proposed by Matsuzaki, Miyao, and Tsujii [MMT05] for CFGs by adding a latent annotation to each nonterminal. They tested different uniform numbers of splits, but not every nonterminal needs the same number of latent annotations. Hence, it is desirable to find a good number of splits for each nonterminal in an automated manner. One such approach was developed by Petrov, Barrett, Thibaux, and Klein [PBTK06] by using a so-called split-merge procedure. A splitting and a merging step are called in a loop, where in between an expectation-maximization algorithm is performed. In the splitting step each nonterminal is splitted into two new nonterminals (represented through latent annotation). The merging step tries to figure out which state splits were not very advantageous and reverses these splits where appropriate. As mentioned before, we can understand LCFRS as a generalization of CFG. This is why we hope to successfully apply this split-merge approach also to LCFRS.

In this thesis we define linear context-free rewriting systems with latent annotation

(LCFRS-LA) and a split-merge algorithm with the neccessary components for training, splitting, and merging. Afterwards, we evaluate the quality of the split-merge algorithm for LCFRS-LA on the languages German and Polish.

2 Preliminaries

2.1 Notation

Let $\mathbb{N}$ be the set of all non negative natural numbers and $\mathbb{N}_+ = \mathbb{N} \setminus \{0\}$.

Let $\mathbb{R}_{\geq 0}$ be the set of all real numbers greater or equal 0.

Let S be a totally ordered set and $a, b \in S$. We denote the closed interval from a to b with $[a, b] = \{s \mid s \in S, a \leq s \leq b\}$.

An *alphabet* is a finite, non-empty set. We call the elements of such a set symbols.

Let S be some set. A *corpus over S* is a mapping $h \colon S \to \mathbb{R}_{\geq 0}$ that assigns each element of S a frequency inside the corpus.

Let S be a countable set. An *S-sorted set* is a tuple *(A, sort)* where A is some set and $sort \colon A \to S$ where $A_s = \{a \mid a \in A,\ sort(a) = s\}$. For simplicity, we will write A for *(A, sort)*.

Let S be a countable set, Σ be a finite $(S^* \times S)$-sorted set and H be an S-sorted set. *The set of trees over Σ and H*, denoted by $T_\Sigma(H)$, is the smallest S-sorted set T such that

- $H_s \subseteq T_s$ for every $s \in S$ and
- for every $k \geq 0$, $s \in S$, $s_1, \ldots, s_k \in S$, $\sigma \in \Sigma_{(s_1 \ldots s_k, s)}$, and $d_1 \in T_{s_1}, \ldots, d_k \in T_{s_k}$ it holds, that $\sigma(d_1, \ldots, d_k) \in T_s$.

We will denote $T_\Sigma(\emptyset)$ by T_Σ.

Let S be a countable set, Σ be a finite $(S^* \times S)$-sorted set, H be an S-sorted set, and $d \in T_\Sigma(H)$. We define $pos(d) \subseteq (\mathbb{N}_+)^*$ as follows:

$$pos(d) = \begin{cases} \{\varepsilon\} & \text{if } d \in H_s \text{ for every } s \in S \\ \{\varepsilon\} \cup \bigcup\limits_{i=1}^{k} \{i\} \cdot pos(d_i) & \text{if } d = \sigma(d_1, \ldots, d_k) \text{ for every } k \in \mathbb{N}, s \in S, \\ & s_1, \ldots, s_k \in S, \sigma \in \Sigma_{(s_1 \ldots s_k, s)}, d_1 \in T_{s_1}, \ldots, d_k \in T_{s_k} . \end{cases}$$

Let S be a countable set, Σ be a finite $(S^* \times S)$-sorted set, H be an S-sorted set, $d \in T_\Sigma(H)$, and $q \in pos(d)$. The *label of a tree d at position q* is defined as:

$$d(q) = \begin{cases} h & \text{if } q = \varepsilon, d = h \in H_s \text{ for every } s \in S \\ \sigma & \text{if } q = \varepsilon, d = \sigma(d_1, \ldots, d_k) \text{ for every } k \in \mathbb{N}, s \in S, s_1, \ldots, s_k \in S, \\ & \sigma \in \Sigma_{(s_1 \ldots s_k, s)}, d_1 \in T_{s_1}, \ldots, d_k \in T_{s_k} \\ d_i(v) & \text{if } q = iv, d = \sigma(d_1, \ldots, d_k) \text{ for every } i \in \mathbb{N}_+,\ v \in (\mathbb{N}_+)^*, k \in \mathbb{N}, \\ & s \in S, s_1, \ldots, s_k \in S\ , \sigma \in \Sigma_{(s_1 \ldots s_k, s)}, d_1 \in T_{s_1}, \ldots, d_k \in T_{s_k}. \end{cases}$$

Let S be a countable set, Σ be a finite $(S^* \times S)$-sorted set, H be an S-sorted set and $d \in T_\Sigma(H)$. We define the sets $leaves(d)$ and $labels(d)$ by

$$\begin{aligned} leaves(d) &= \{q \mid q \in pos(d), \nexists\, i \in \mathbb{N}_+ \colon qi \in pos(d)\} \text{ and} \\ labels(d) &= \{d(q) \mid q \in pos(d)\}\ , \end{aligned}$$

respectively.

2.2 Linear context-free Rewriting System

For every $k \in \mathbb{N}$ and $l_1, \ldots, l_k \in \mathbb{N}$ we define

$$X_{(l_1, \ldots, l_k)} = \{x_j^{(i)} \mid i \in \{1, \ldots, k\}, j \in \{1, \ldots, l_i\}\}\ .$$

Let Δ be an alphabet. We define Ω^Δ be the $(\mathbb{N}^* \times \mathbb{N})$-sorted set such that for every $k \in \mathbb{N}$, $l_1, \ldots, l_k \in \mathbb{N}$, and $n \in \mathbb{N}$ it holds that

$$\begin{aligned} \Omega^\Delta_{(l_1 \ldots l_k,\, n)} = \{\langle w_1, \ldots, w_n \rangle \mid w_1, \ldots, w_n \in (\Delta \cup X_{(l_1, \ldots, l_k)})^*, &\text{ such that each} \\ x_j^{(i)} \in X_{(l_1, \ldots, l_k)} &\text{ occurs exactly once in } w_1 \ldots w_n\}\ . \end{aligned}$$

Let S be a set of sorts and Σ be a finite $(S^* \times S)$-sorted set. An *S-sorted Σ-algebra* is a pair(A, ϕ) where

- A is an S-sorted set and
- for every $(s_1 \ldots s_k, s) \in (S^* \times S), \sigma \in \Sigma_{(s_1 \ldots s_k, s)}$ is $\phi(\sigma) \colon A_{s_1} \times \ldots \times A_{s_k} \to A_s$.

Let Δ be an alphabet. The $\mathbb{N}$-*sorted* Ω^Δ-*algebra* $\mathcal{LCFRS}$ is the pair (W, ϕ) where

- W is the $\mathbb{N}$-sorted set such that for every $k \in \mathbb{N}$ it holds that $W_n = (\Delta^*)^n$ and
- for every $k, n \in \mathbb{N}$, $(l_1 \ldots l_k, n) \in \mathbb{N}^* \times \mathbb{N}, u_1^{(1)}, \ldots, u_{l_1}^{(1)}, \ldots, u_1^{(k)}, \ldots, u_{l_k}^{(k)} \in \Delta^*$, and $\omega = \langle w_1, \ldots, w_n \rangle \in \Omega^\Delta_{(l_1 \ldots l_k, n)}$ we have
$$\phi(\omega)((u_1^{(1)} \ldots u_{l_1}^{(1)}), \ldots, (u_1^{(k)} \ldots u_{l_k}^{(k)})) = (w'_1, \ldots, w'_n) \ .$$
To obtain w'_κ from w_κ replace each $x_j^{(i)}$ by $u_j^{(i)}$ for every $\kappa \in \{1, \ldots, n\}, 1 \leq i \leq k$, and $1 \leq j \leq l_k$.

Definition 2.1. A *linear context-free rewriting system* (LCFRS) is a tuple $G = (N, \Delta, S, R)$ where:

- N is an $\mathbb{N}$-sorted set (*nonterminals*),
- Δ is an alphabet (*terminals*),
- $S \in N_1$ (*initial nonterminal*), and
- R is a finite set (*rules*), where each rule has the form $A \to \omega(B_1, \ldots, B_k)$ with $k \in \mathbb{N}, n \in \mathbb{N}, A \in N_n, l_1, \ldots, l_k \in \mathbb{N}, \omega \in \Omega^\Delta_{(l_1 \ldots l_k, n)}$, and $B_1 \in N_{l_1}, \ldots, B_k \in N_{l_k}$.

We will call the sort of an element of N its *fan-out*. □

Example 2.1. Let $\Delta = \{a, b_1, b_2, c\}$ and $N = \{S^1, A^1, B^1, D^2, C^1\}$ be an $\mathbb{N}$-sorted set, where the sort of the element is denoted as superscript. Defining the set R as:

$$\begin{aligned} R = \{ & S \to \langle x_1^{(2)} x_1^{(1)} x_2^{(2)} \rangle (B, D), \\ & D \to \langle x_1^{(2)}, x_1^{(1)} \rangle (A, C), \\ & A \to \langle a \rangle (), \\ & B \to \langle b_1 \rangle (), \\ & B \to \langle b_2 \rangle (), \\ & C \to \langle c \rangle () \\ \}, & \end{aligned}$$

we obtain an LCFRS (N, Δ, S^1, R). □

Let $G = (N, \Delta, S, R)$ be an LCFRS. We call G start separated if S does not occur on the right-hand side of any rule in R.

Let $G = (N, \Delta, S, R)$ be an LCFRS. For every $k \in \mathbb{N}$ we can understand R as an $(N^* \times N)$-sorted set such that

$$R_{B_1 \ldots B_k, A} = \{\rho \mid \rho \in R, n \in \mathbb{N}, l_1, \ldots, l_k \in \mathbb{N}, \omega \in \Omega^\Delta_{(l_1 \ldots l_k, n)}, \rho = (A \to \omega(B_1, \ldots, B_k))\} \ .$$

Let $G = (N, \Delta, S, R)$ be an LCFRS and $A \in N$. The *set of all abstract syntax trees of G rooted in A* is the set $(T_R)_A$.
The *set of all abstract syntax trees of G*, denoted by D_G, is $(T_R)_S$.

Let $G = (N, \Delta, S, R)$ be an LCFRS, $A \in N, k \in \mathbb{N}, \rho = (A \to \omega(B_1, \ldots, B_k)) \in R$, and $d = \rho(d_1, \ldots, d_k) \in (T_R)_A$. We define the mapping $\pi_\Omega \colon (T_R)_A \to T_{\Omega^\Delta}$ where

$$\pi_\Omega(d) = \omega(\pi_\Omega(d_1), \ldots, \pi_\Omega(d_k)) \ .$$

Let Δ be an alphabet, $n \geq 0$, $k \geq 0$, $\omega = \langle w_1, \ldots, w_n \rangle \in \Omega^\Delta_{(l_1 \ldots l_k, n)}$, $d_1 \ldots d_k \in T_{\Omega^\Delta}$, and $d = \omega(d_1, \ldots, d_k) \in T_{\Omega^\Delta}$. The *semantics of an abstract syntax tree under $\mathcal{LCFRS}$*, denoted by $[\![.]\!]^{\mathcal{LCFRS}}$, is $[\![.]\!]^{\mathcal{LCFRS}} \colon (T_{\Omega^\Delta})_n \to (\Delta^*)^n$ for every $n \in \mathbb{N}$ such that

$$[\![d]\!]^{\mathcal{LCFRS}} = \phi(\omega)([\![d_1]\!]^{\mathcal{LCFRS}}, \ldots, [\![d_k]\!]^{\mathcal{LCFRS}}).$$

Let G be an LCFRS. *The language generated by G* is $L(G) = \{[\![\pi_\Omega(d)]\!]^{\mathcal{LCFRS}} \mid d \in D_G\}$.

Example 2.2. Let G be the LCFRS constructed in example 2.1 and let

$$d = \begin{tikzcd}
& S \to \langle x_1^{(2)} x_1^{(1)} x_2^{(2)} \rangle (B, D) \arrow[dl, dash] \arrow[dr, dash] & & \\
B \to \langle b_1 \rangle () & & D \to \langle x_1^{(2)}, x_1^{(1)} \rangle (A, C) \arrow[dl, dash] \arrow[dr, dash] & \\
& A \to \langle a \rangle () & & C \to \langle c \rangle () \ .
\end{tikzcd}$$

We let $d' = \pi_\Omega(d)$. The evaluation of the semantics of d' under $\mathcal{LCFRS}$ is as follows:

$$\begin{aligned}
[\![d']\!]^{\mathcal{LCFRS}} &= \phi(\langle x_1^{(2)} x_1^{(1)} x_2^{(2)} \rangle) \Big([\![\langle b_1 \rangle]\!]^{\mathcal{LCFRS}}, \left[\!\!\left[\begin{array}{ccc} & \langle x_1^{(2)}, x_1^{(1)} \rangle & \\ / & & \backslash \\ \langle a \rangle & & \langle c \rangle \end{array} \right]\!\!\right]^{\mathcal{LCFRS}} \Big) \\
&= \phi(\langle x_1^{(2)} x_1^{(1)} x_2^{(2)} \rangle) \Big(\phi(\langle b_1 \rangle)(), \phi(\langle x_1^{(2)}, x_1^{(1)} \rangle) \big([\![\langle a \rangle]\!]^{\mathcal{LCFRS}}, [\![\langle c \rangle]\!]^{\mathcal{LCFRS}} \big) \Big) \\
&= \phi(\langle x_1^{(2)} x_1^{(1)} x_2^{(2)} \rangle) \Big((b_1), \phi(\langle x_1^{(2)}, x_1^{(1)} \rangle) \big(\phi(\langle a \rangle)(), \phi(\langle c \rangle)() \big) \Big) \\
&= \phi(\langle x_1^{(2)} x_1^{(1)} x_2^{(2)} \rangle) \Big((b_1), \phi(\langle x_1^{(2)}, x_1^{(1)} \rangle) \big((a), (c) \big) \Big) \\
&= \phi(\langle x_1^{(2)} x_1^{(1)} x_2^{(2)} \rangle) \Big((b_1), (c, a) \Big) \\
&= (c b_1 a)
\end{aligned}$$

□

2.3 Probabilistic linear context-free Rewriting System with Latent Annotation

Definition 2.2. A *linear context-free rewriting system with latent annotation* (LCFRS-LA) is a tuple (G, L) where:

- $G = (N, \Delta, S, R)$ is an LCFRS and
- $L = (L_A)_{A \in N}$ is a family of finite sets L_A (*latent variables*) for every $A \in N$.

$\square$

Let $((N, \Delta, S, R), L)$ be an LCFRS-LA. We define the sets $S[L]$, $R[L]$, and $R[L]_{A_x}$ by

$$\begin{aligned}
S[L] &= \{S_x \mid x \in L_S\}, \\
R[L] &= \{A_x \to \omega((B_1)_{y_1}, \ldots, (B_k)_{y_k}) \mid A \to \omega(B_1, \ldots, B_k) \in R,\, x \in L_A, \\
&\qquad y_1 \in L_{B_1}, \ldots, y_k \in L_{B_k}\}, \text{ and} \\
R[L]_{A_x} &= \{\rho \mid \rho \in R[L], \rho = A_x \to \omega((B_1)_{y_1}, \ldots, (B_k)_{y_k})\}
\end{aligned}$$

respectively.

Definition 2.3. A *probabilistic linear context-free rewriting system with latent annotation* (PLCFRS-LA) is a tuple $((N, \Delta, S, R), L, \gamma, p)$ where:

- $((N, \Delta, S, R), L)$ is an LCFRS-LA,
- $\gamma\colon S[L] \to [0, 1]$ such that $\sum\limits_{x \in L_S} \gamma(S_x) = 1$, and
- $p\colon R[L] \to [0, 1]$ such that for every $A \in N$ and $x \in L_A$ is $\sum\limits_{\rho \in R[L]_{A_x}} p(\rho) = 1$

$\square$

Let G be an LCFRS and $G' = (G, L, \gamma, p)$ be a PLCFRS-LA. The language generated by G' is $L(G') = L(G)$.

Example 2.3. Let G be the LCFRS constructed in example 2.1 . We let $L_S = \{w_1, w_2\}$, $L_A = \{x\}$, $L_B = \{y_1, y_2\}$, $L_C = \{z\}$, and $L_D = \{v\}$ such that $L = (L_S, L_A, L_B, L_C, L_D)$. Moreover, we let $\gamma\colon S[L] \to [0, 1]$ and set $\gamma(S_{w_1}) = 0.6$ and $\gamma(S_{w_2}) = 0.4$. We let $p\colon R[L] \to [0, 1]$ and set it for each element of $R[L]$ as follows:

$$
\begin{aligned}
p(S_{w_1} \to \langle x_1^{(2)} x_1^{(1)} x_2^{(2)} \rangle (B_{y_1}, D_v)) &= 0.7, \\
p(S_{w_1} \to \langle x_1^{(2)} x_1^{(1)} x_2^{(2)} \rangle (B_{y_2}, D_v)) &= 0.3, \\
p(S_{w_2} \to \langle x_1^{(2)} x_1^{(1)} x_2^{(2)} \rangle (B_{y_1}, D_v)) &= 0.4, \\
p(S_{w_2} \to \langle x_1^{(2)} x_1^{(1)} x_2^{(2)} \rangle (B_{y_2}, D_v)) &= 0.6, \\
p(D_v \to \langle x_1^{(2)}, x_1^{(1)} \rangle (A_x, C_z)) &= 1, \\
p(A_x \to \langle a \rangle ()) &= 1, \\
p(B_{y_1} \to \langle b_1 \rangle ()) &= 0.35, \\
p(B_{y_1} \to \langle b_2 \rangle ()) &= 0.65, \\
p(B_{y_2} \to \langle b_1 \rangle ()) &= 0.79, \\
p(B_{y_2} \to \langle b_2 \rangle ()) &= 0.21, \text{ and} \\
p(C_z \to \langle c \rangle ()) &= 1 \ .
\end{aligned}
$$

We obtain the PLCFRS-LA (G, L, γ, p) .

$\square$

Definition 2.4. Let $((N, \Delta, S, R), L, \gamma, p)$ be a PLCFRS-LA and $d \in D_G$. For every $A \in N, x \in L_A$, and $q \in pos(d)$ with $d(q) = A \to \omega(D_1, \ldots, D_k) \in R$ we define the inside and outside weights, denoted by *in* and *out* respectively, recursively as follows:

$$
in(q, x) = \sum_{y_1 \in L_{D_1}} \cdots \sum_{y_k \in L_{D_k}} p(A_x \to \omega((D_1)_{y_1}, \ldots, (D_k)_{y_k})) \cdot in(q1, y_1) \cdot \ldots \cdot in(qk, y_k),
$$

$out(q, x) = \gamma(A_x)$ if $q = \varepsilon$, $A = S$,

and if $q = vj, v \in (\mathbb{N}_+)^*, j \in \mathbb{N}_+, r \geq 0$, and $d(v) = C \to \omega(B_1, \ldots, B_{j-1}, A, B_{j+1}, \ldots, B_r)$ then

$$
\begin{aligned}
out(q, x) = \sum_{z \in L_C} \sum_{\substack{y_1 \in L_{B_1} \\ \cdots \\ y_{j-1} \in L_{B_{j-1}} \\ y_{j+1} \in L_{B_{j+1}} \\ \cdots \\ y_r \in L_{B_r}}} out(v, z) \cdot \prod_{m \in \{1, \ldots, r\} \setminus \{j\}} in(vm, y_m) \\
\cdot p(C_z \to \omega((B_1)_{y_1}, \ldots, (B_{j-1})_{y_{j-1}}, A_x, (B_{j+1})_{y_{j+1}}, \ldots, (B_r)_{y_r})).
\end{aligned}
$$

$\square$

Definition 2.5. Let $G = (N, \Delta, S, R)$ be an LCFRS, (G, L, γ, p) be a PLCFRS-LA, and $d \in D_G$. The *probability of an abstract syntax tree of G*, denoted by $\widetilde{p}$, is defined as follows:

$$
\widetilde{p}(d) = \sum_{x \in L_S} \gamma(S_x) \cdot in(\varepsilon, x).
$$

In the following we will denote $\widetilde{p}(d)$ by $p(d)$.

□

Example 2.4. Let (G, L, γ, p) be the PLCFRS-LA constructed in example 2.3 and

$$d = \begin{tikzcd}[column sep=small]
 & S \to \langle x_1^{(2)} x_1^{(1)} x_2^{(2)} \rangle (B, D) \arrow[dl, dash] \arrow[dr, dash] & & \\
B \to \langle b_1 \rangle () & & D \to \langle x_1^{(2)}, x_1^{(1)} \rangle (A, C) \arrow[dl, dash] \arrow[dr, dash] & \\
 & A \to \langle a \rangle () & & C \to \langle c \rangle () \;.
\end{tikzcd}$$

We calculate the inside weights for every position and annotated rule as follows:

$$\begin{aligned}
in(22, z) &= p(C_z \to \langle c \rangle ()) = 1 \\
in(21, x) &= p(A_x \to \langle a \rangle ()) = 1 \\
in(2, v) &= p(D_v \to \langle x_1^{(2)}, x_1^{(1)} \rangle (A_x, C_z)) \cdot in(21, x) \cdot in(22, z) \\
&= 1 \cdot 1 \cdot 1 \\
&= 1 \\
in(1, y_1) &= p(B_{y_1} \to \langle b_1 \rangle ()) = 0.35 \\
in(1, y_2) &= p(B_{y_2} \to \langle b_1 \rangle ()) = 0.79 \\
in(\varepsilon, w_1) &= p(S_{w_1} \to \langle x_1^{(2)} x_1^{(1)} x_2^{(2)} \rangle (B_{y_1}, D_v)) \cdot in(1, y_1) \cdot in(2, v) \\
&\quad + p(S_{w_1} \to \langle x_1^{(2)} x_1^{(1)} x_2^{(2)} \rangle (B_{y_2}, D_v)) \cdot in(1, y_2) \cdot in(2, v) \\
&= 0.7 \cdot 0.35 \cdot 1 + 0.3 \cdot 0.79 \cdot 1 \\
&= 0.482 \\
in(\varepsilon, w_2) &= p(S_{w_2} \to \langle x_1^{(2)} x_1^{(1)} x_2^{(2)} \rangle (B_{y_1}, D_v)) \cdot in(1, y_1) \cdot in(2, v) \\
&\quad + p(S_{w_2} \to \langle x_1^{(2)} x_1^{(1)} x_2^{(2)} \rangle (B_{y_2}, D_v)) \cdot in(1, y_2) \cdot in(2, v) \\
&= 0.4 \cdot 0.35 \cdot 1 + 0.6 \cdot 0.79 \cdot 1 \\
&= 0.614
\end{aligned}$$

Now we calculate the probability of d.

$$\begin{aligned}
p(d) &= \sum_{x \in L_S} \gamma(S_x) \cdot in(\varepsilon, x) \\
&= \gamma(w_1) \cdot in(\varepsilon, w_1) + \gamma(w_2) \cdot in(\varepsilon, w_2) \\
&= 0.6 \cdot 0.482 + 0.4 \cdot 0.614 \\
&= 0.5348
\end{aligned}$$

□

Definition 2.6. Let (G, L, γ, p) be a PLCFRS-LA and $w \in \Delta^*$. *The best abstract syntax tree $\hat{d}$ for w* is defined as follows:

$$\hat{d} = \operatorname*{argmax}_{d \in D_G \colon [\![\pi_\Omega(d)]\!]^{\mathcal{LCFRS}} = w} p(d) \,.$$

$\square$

2.4 The Expectation-Maximization Algorithm

Definition 2.7. Let (G, L, γ, p) be a PLCFRS-LA and $h \colon D_G \to \mathbb{R}_{\geq 0}$ be a corpus over abstract syntax trees. The *likelihood of h under γ and p*, denoted by $\mathcal{L}(h, \gamma, p)$, is defined as

$$\mathcal{L}(h, \gamma, p) = \prod_{d \in D_G} p(d)^{h(d)} \,.$$

$\square$

Definition 2.8. Let (G, L, γ, p) be a PLCFRS-LA and $h \colon D_G \to \mathbb{R}_{\geq 0}$ be a corpus over abstract syntax trees. The *maximum likelihood estimate for h* is defined as

$$(\hat{\gamma}, \hat{p}) = \operatorname*{argmax}_{\gamma, p} \mathcal{L}(h, \gamma, p).$$

$\square$

Let (G, L, γ, p) be a PLCFR-LA and $h : D_G \to \mathbb{R}_{\geq 0}$ a corpus over abstract syntax trees. Our goal is to learn a γ' and p' which maximize the likelihood of the corpus h. Therefore, we use the expectation-maximization algorithm (cf. Algorithm 2.1) which generates a sequence (γ_1, p_1), (γ_2, p_2), (γ_3, p_3), $\ldots$ such that $\mathcal{L}(h, \gamma_1, p_1) \leq \mathcal{L}(h, \gamma_2, p_2) \leq \cdots \leq \mathcal{L}(h, \hat{\gamma}, \hat{p})$. It consists of an expectation and a maximization step. In the expectation step (cf. Algorithm 2.1, line 7-15) the inside and outside weights for each annotated rule and position for every abstract syntax tree in $supp(h)$ are computed jointly. Next, the probability for each annotated rule and position for every abstract syntax tree in $supp(h)$ is computed, using the already available inside and outside weights, and counts for every annotated rule are accumulated by summarizing the corresponding probabilities. This will also be the case for each annotated initial nonterminal. In the maximization step (cf. Algorithm 2.1, line 16-27), the counts for each annotated rule and initial nonterminal are normalized to obtain new probability assignments.

A problem with the algorithm arises when the probabilities of the annotated rules and the annotated initial nonterminals are equally distributed. This leads the expectation-maximization algorithm to stagnate earlier. Hence, it is neccessary to break the symmetry to make sure the algorithm finds a local maximum.

Algorithm 2.1 The Expectation-Maximization Algorithm

```
Require: Let $G = (N, \Delta, S, R)$ be a start separated LCFRS, $(G, L, \gamma_0, p_0)$ be a PLCFRS-
    LA, $(\gamma_0, p_0)$ be an initial probability assignments, and $h\colon D_G \to \mathbb{R}_{\geq 0}$ be a corpus
    over abstract syntax trees such that $supp(h) \subseteq D_G$.
Ensure: approximation of a local maximum or saddle point of $\mathcal{L}(h, ., .)$ by comput-
    ing a sequence of probability assignments $(\gamma_1, p_1), (\gamma_2, p_2), (\gamma_3, p_3), \ldots$ such that
    $\mathcal{L}(h, \gamma_i, p_i) \leq \mathcal{L}(h, \gamma_{i+1}, p_{i+1})$.
 1: $i \leftarrow 1$
 2: while not converged do
 3:     $\gamma \leftarrow \gamma_{i-1}$
 4:     $p \leftarrow p_{i-1}$
 5:     $c(\rho) \leftarrow 0$ for every $\rho \in R[L]$
 6:     $c(S_x) \leftarrow 0$ for every $S_x \in S[L]$
 7:     for all $d \in supp(h)$ do
 8:         for all $q \in pos(d)$ do
 9:             let $\rho = (A \to \omega(B_1, \ldots, B_k)) \leftarrow d(q)$
10:             for all $x \in L_A$ and $y_1 \in L_{B_1}, \ldots, y_k \in L_{B_k}$ do
11:                 let $\rho' = (A_x \to \omega((B_1)_{y_1}, \ldots, (B_k)_{y_k})$
12:                 $\psi \leftarrow out(q, x) \cdot p(\rho') \cdot in(q1, y_1) \cdot \ldots \cdot in(qk, y_k)$
13:                 $c(\rho') \leftarrow c(\rho') + \psi \cdot h(d) \cdot \frac{1}{p(d)}$
14:         for all $x \in L_S$ do
15:             $c(S_x) \leftarrow c(S_x) + out(\varepsilon, x) \cdot in(\varepsilon, x) \cdot h(d) \cdot \frac{1}{p(d)}$
16:     for all $\rho = \big(A_x \to \omega((B_1)_{y_1}, \ldots, (B_k)_{y_k})\big) \in R[L]$ do
17:         $s \leftarrow \sum_{\rho' \in R[L]_{A_x}} c(\rho')$
18:         if $s > 0$ then
19:             $p_i(\rho) \leftarrow \frac{c(\rho)}{s}$
20:         else
21:             $p_i(\rho) \leftarrow p_{i-1}(\rho)$
22:     for all $x \in L_S$ do
23:         $s \leftarrow \sum_{S' \in S[L]} c(S')$
24:         if $s > 0$ then
25:             $\gamma_i(S_x) \leftarrow \frac{c(S_x)}{s}$
26:         else
27:             $\gamma_i(S_x) \leftarrow \gamma_{i-1}(S_x)$
28:     output$(\gamma_i, p_i)$
29:     $i \leftarrow i + 1$
```

3 The Split-Merge Algorithm

Consider the PLCFRS-LA (G, L, γ, p). Given that, we know how to optimize γ and p using the expectation-maximization algorithm (cf. Algorithm 2.1). Now our goal is to find a way to determine L. One way to do that, is to test different uniform numbers of splits e.g $\{1, 2, 4, 8, 16, \ldots\}$ as done by Matsuzaki et al. [MMT05]. However, this leads us to a huge amount of nonterminals for greater numbers of splits, which makes the grammar too complex for practical work. Moreover, not every nonterminal needs the same number of latent annotation. There are much less categories of punctuations than categories of nouns. So we want to have a way to learn and refine the latent annotation automatically. Petrov et al. [PBTK06] presented a so called split-and-merge approach which does exactly that for PCFG-LA. It consists of a splitting and a merging step, where in between an expectation-maximization algorithm is executed.

In this chapter we want to formalize a split-merge algorithm for PLCFRS-LA. Therefore we describe the splitting and merging step and bring this together in a final split-merge algorithm.

3.1 Splitting

Let $((N, \Delta, S, R), L, \gamma, p)$ be a PLCFRS-LA. When splitting we divide each latent variable into two new ones. This results in a new PLCFRS-LA $((N, \Delta, S, R), L', \gamma', p')$.

In detail, for every $A \in N$ it holds that $L'_A = \{(x)_i \mid x \in L_A, i \in \{1, 2\}\}$. We use a mapping $\beta \colon L'_A \to L_A$ to track the origin of a splitted latent variable such that for every $A \in N$ and $x \in L_A$, where $x_1, x_2 \in L'_A$ are the splits of x, it holds that $\beta(x_1) = \beta(x_2) = x$. Moreover, for every $x \in L_S$ we have to distribute the probability of $\gamma(S_x)$ over the splitted latent variables of x. Obviously we could set $\gamma'(S_{x_1}) = 0.5 \cdot \gamma(S_x)$ and $\gamma'(S_{x_2}) = 0.5 \cdot \gamma(S_x)$. However, we want to add a certain amount of randomness to break to the symmetry. Therefore we randomly choose an $\alpha \in [0.49, 0.51]$ such that the following applies: $\gamma'(S_{x_1}) = \alpha \cdot \gamma(S_x)$ and $\gamma'(S_{x_2}) = (1 - \alpha) \cdot \gamma(S_x)$. In addition, for every $A \in N, x \in L_A$,, rule $\rho = A_x \to \omega((B_1)_{y_1}, \ldots, (B_k)_{y_k})$ and $\rho \in R[L]$ we have to choose $(\alpha_{y_1})_1, \ldots, (\alpha_{y_k})_1 \in [0.49, 0.51]$ and we let

$$(\alpha_{y_1})_2 = 1 - (\alpha_{y_1})_1, \ldots, (\alpha_{y_k})_2 = 1 - (\alpha_{y_k})_1 \quad .$$

Now for every $i_0, \ldots, i_k \in \{1, 2\}$ we set

$$p'(A_{x_{(i_0)}} \to \omega((B_1)_{y_{1_{(i_1)}}}, \ldots, (B_k)_{y_{k_{(i_k)}}})) = (\alpha_{y_1})_{i_1} \cdot \ldots \cdot (\alpha_{y_k})_{i_k} \cdot p(\rho) \quad .$$

Algorithm 3.1 describes a possible implementation of this splitting step in pseudo code.

Algorithm 3.1 The Splitting Step

```
Require: Let $G = ((N, \Delta, S, R), L, \gamma, p)$ be a PLCFRS-LA.
Ensure: PLCFRS-LA $G' = ((N, \Delta, S, R), L', \gamma', p')$
 1: for all $A \in N$ do
 2:     $L'_A \leftarrow \{(x)_i \mid x \in L_A, i \in \{1, 2\}\}$
 3: for all $x \in L_S$ do
 4:     $\alpha \leftarrow$ randomly choose from $[0.49, 0.51]$
 5:     $\gamma'(S_{x_1}) \leftarrow \alpha \cdot \gamma(S_x)$
 6:     $\gamma'(S_{x_2}) \leftarrow (1 - \alpha) \cdot \gamma(S_x)$
 7: for all $\rho = A_x \to \omega((B_1)_{y_1}, \dots, (B_k)_{y_k}) \in R[L]$ do
 8:     $(\alpha_{y_1})_1 \leftarrow [0.49, 0.51], \dots, (\alpha_{y_k})_1 \leftarrow [0.49, 0.51]$
 9:     $(\alpha_{y_1})_2 \leftarrow (1 - (\alpha_{y_1})_1), \dots, (\alpha_{y_k})_2 \leftarrow (1 - (\alpha_{y_k})_1)$
10:     for all $i_0, \dots, i_k \in \{1, 2\}$ do
11:         $p'\Big((A_{x_{(i_0)}}) \to \omega((B_1)_{y_{1_{(i_1)}}}, \dots, (B_k)_{y_{k_{(i_k)}}})\Big) \leftarrow (\alpha_{y_1})_{i_1} \cdot \dots \cdot (\alpha_{y_k})_{i_k} \cdot p(\rho)$
```

3.2 Merging

Let $G = (N, \Delta, S, R)$ be a LCFRS. We assume that (G, L, γ, p) is the PLCFRS-LA after splitting and $h \colon D_G \to \mathbb{R}_{\geq 0}$ is a corpus over abstract syntax trees. We want to evaluate how useful some split was. One way to to this, is to reverse the split, which leads us to a new PLCFRS-LA (G, L', γ', p'), and to compare $\mathcal{L}(h, \gamma, p)$ with $\mathcal{L}(h, \gamma', p')$.
In formal terms, let $A \in N, x_1, x_2 \in L_A$ and $\beta(x_1) = x = \beta(x_2)$. We construct (G, L', γ', p') as follows: For every $A' \in N$ we set $L'_{A'} = L_{A'}$ if $A' \neq A$ and $L'_{A'} = L_{A'} \setminus \{x_1, x_2\} \cup \{x\}$ if $A' = A$. Moreover, we set $\gamma'(A_x) = \gamma(A_{x_1}) + \gamma(A_{x_2})$, if $A = S$ and for every $S_y \in S[L']$ we set $\gamma'(S_y) = \gamma(S_y)$ if $S_y \neq A_x$. It remains to the define the probabilities of rules containing A. Since A_x fills the role of A_{x_1} and A_{x_2}, we accumulate the probabilities of rules containing A_{x_1} and A_{x_2} if A occurs on the left-hand side of a rule. In order to obtain a proper probability assignment, we compute the average of the rule probability weighted according to the relative frequency of A_{x_1} and A_{x_2} in the corpus h. To this end, for $j \in \{1, 2\}$ we define the expected frequency of A_{x_j} by

$$freq(A_{x_j}) = \sum_{d \in supp(h)} h(d) \sum_{\substack{q \in pos(d): \\ d(q) = A \to \omega(B_1, \dots, B_k)}} in(q, x_j) \cdot out(q, x_j)$$

and, eventually, the relative frequency of A_{x_j} by

$$\alpha_j = \frac{freq(A_{x_j})}{freq(A_{x_1}) + freq(A_{x_2})} \ .$$

Moreover, we define $f\colon R[L] \to \mathbb{R}_{\geq 0}$ and $g_B\colon L_B \to L'_B$ for every $B \in N$ by

$$f(C_z \to \omega((B_1)_{y_1}, \dots, (B_k)_{y_k})) = \begin{cases} 1 & \text{if } C_z \notin \{A_{x_1}, A_{x_2}\}, \\ \alpha_1 & \text{if } C_z = A_{x_1}, \\ \alpha_2 & \text{if } C_z = A_{x_2} \end{cases}$$

and

$$g_B(y) = \begin{cases} \beta(y) & \text{if } (y = x_1 \text{ or } y = x_2) \text{ and } B = A, \\ y & \text{otherwise} \ , \end{cases}$$

respectively. We overload the definition of g_B and let $g_B\colon R[L] \to R[L']$ such that for every $\rho = C_z \to \omega((B_1)_{y_1}, \dots, (B_k)_{y_k}) \in R[L]$ it holds that

$$g_B(\rho) = C_{g_B(z)} \to \omega((B_1)_{g_B(y_1)}, \dots, (B_k)_{g_B(y_k)}) \, .$$

Now, for every $\rho' \in R[L']$ we set

$$p'(\rho') = \sum_{\rho \in g^{-1}(\rho')} p(\rho) \cdot f(\rho) \, .$$

As mentioned we want to compare the likelihoods to see how valuable the split was. Since the computation of the likelihood is very expensive, so its not feasible to recompute it for every split, we approximate the effect of merging x_1 and x_2 to x. Consider $d \in supp(h)$ and $q \in pos(d)$ such that $d(q) = A \to \omega(B_1, \dots, B_k)$. Now suppose that we merge at q only. As x combines the statistics of x_1 and x_2, the inside weight for x at q is calculated by the sum of the inside weights of x_1 and x_2 at q, weighted by their relative frequencies α_1 and α_2, which leads us to

$$in(q, x) = \alpha_1 \cdot in(q, x_1) + \alpha_2 \cdot in(q, x_2) \, .$$

Moreover, x can be generated by the parents of both x_1 and x_2, such that the outside weight is

$$out(q, x) = out(q, x_1) + out(q, x_2) \ .$$

Now we approximate the likelihood difference due to merging x_1 and x_2 in every $d \in supp(h)$ by the product of merging at single positions as follows:

$$\Phi(\{x\}, \{x_1, x_2\}) = \prod_{d \in supp(h)} \prod_{\substack{q \in pos(d), \\ d(q): A \to \omega(B_1, \dots, B_k)}} \frac{\Big(\sum\limits_{x' \in L'_A} out(q, x') \cdot in(q, x')\Big)^{h(d)}}{\Big(\sum\limits_{x' \in L_A} out(q, x') \cdot in(q, x')\Big)^{h(d)}} \ .$$

The numerator of the fraction expresses the probability of d after merging at a single occurrence and the denominator expresses the probability of d before merging. Hence if $\Phi(\{x_1\}, \{x_1, x_2\}) \geq 1$, then the split is presumably not advantageous and we should merge. If $\Phi(\{x_1\}, \{x_1, x_2\}) < 1$, we should keep it. Of course, we can also choose another threshold μ instead of 1.

On the one hand for every $x_1, x_2 \in L_A$, such that $\beta(x_1) = \beta(x_2) = x$, we want to merge x_1 and x_2 to x if $\Phi(\{x\}, \{x_1, x_2\}) \geq \mu$. On the other hand we want to have a second parameter $\nu \in [0, 1]$ which determines how many splits should be merged at least in order to keep the number of latent annotation small. To achieve this, we define the set

$$\begin{aligned}\mathcal{U} = \{(r, \{x\}, \{x_1, x_2\}, A) \mid A \in N, x_1, x_2 \in L_A, \beta(x_1) = x = \beta(x_2),\\ r = \Phi(\{x\}, \{x_1, x_2\}), \Phi(\{x\}, \{x_1, x_2\}) < \mu\},\end{aligned}$$

which gathers all the splits that are below the threshold. Now, as long as it holds that

$$\lceil \nu \cdot \frac{1}{2} \cdot \sum_{A \in N} |L_A| \rceil > \frac{1}{2} \cdot \Big(\sum_{A \in N} |L_A| \Big) - |\mathcal{U}|$$

we choose a maximal element $(r, \{z\}, \{z_1, z_2\}, A) \in \mathcal{U}$, merge z_1 and z_2 to z, and remove $(r, \{z\}, \{z_1, z_2\}, A)$ from $\mathcal{U}$. Intuitively, the left side of the inequation expresses the number of splits we want to reverse at least and the right side expresses the number of splits which were already merged. Since we remove an element from $\mathcal{U}$ in each iteration, the value on the right increases by one each time and at some point it becomes greater or equal to the value on the left.

This process of merging is described by Algorithm 3.2 .

Algorithm 3.2 The Merging Step

```
Require: Let $G = (N, \Delta, S, R)$ be a start separated LCFRS, $(G, L, \gamma, p)$ be the PLCFRS-LA after the splitting step, $\mu \in \mathbb{R}_{\geq 0}$ be the threshold for merging, $\nu \in [0, 1]$ be the minimal rate of splits to merge, and $h\colon D_G \to \mathbb{R}_{\geq 0}$ be a corpus over abstract syntax trees such that $supp(h) \subseteq D_G$.
Ensure: PLCFRS-LA $G' = (G, L', \gamma', p')$
 1: $(G, L', \gamma', p') \leftarrow (G, L, \gamma, p)$
 2: $\mathcal{U} \leftarrow \emptyset$
 3: for all $A \in N$ do
 4:     for all splits $\{x_1, x_2\} \subseteq L_A$ such that $\beta(x_1) = \beta(x_2) = x$ do
 5:         $r \leftarrow$ compute $\Phi(\{x\}, \{x_1, x_2\})$
 6:         if $r \geq \mu$ then
 7:             REVERSE($\{x\}, \{x_1, x_2\}, A$)
 8:         else
 9:             $\mathcal{U} \leftarrow \mathcal{U} \cup \{(r, \{x\}, \{x_1, x_2\}, A)\}$
10: $merged_{req} \leftarrow \lceil \nu \cdot 0.5 \cdot \sum_{A \in N} |L_A| \rceil$
11: $merged_{cur} \leftarrow 0.5 \cdot \sum_{A \in N} |L_A| - |\mathcal{U}|$
12: while $merged_{req} > merged_{cur}$ do
13:     $(r, \{z\}, \{z_1, z_2\}, D) \leftarrow \operatorname{argmax}_{(r, \{x\}, \{x_1, x_2\}, A) \in \mathcal{U}} (r, \{x\}, \{x_1, x_2\}, A)$
14:     REVERSE($\{z\}, \{z_1, z_2\}, D$)
15:     $\mathcal{U} \leftarrow \mathcal{U} \setminus \{(r, \{z\}, \{z_1, z_2\}, D)\}$
16:     $merged_{cur} \leftarrow merged_{cur} + 1$
17: function REVERSE($\{x\}, \{x_1, x_2\}, A$)
18:     $L'_A \leftarrow L'_A \setminus \{x_1, x_2\} \cup \{x\}$
19:     $\alpha_1 \leftarrow$ compute $\alpha_1$
20:     $\alpha_2 \leftarrow$ compute $\alpha_2$
21:     for all $\rho' = C_z \to \omega((B_1)_{y_1}, \ldots, (B_k)_{y_k}) \in R[L']$ do
22:         $p'(\rho') = \sum_{\rho \in g^{-1}(\rho')} p(\rho) \cdot f(\rho)$
23:     if $A = S$ then
24:         $\gamma'(A_x) \leftarrow \gamma(A_{x_2}) + \gamma(A_{x_2})$
```

3.3 The Split-Merge Algorithm

Let $G = (N, \Delta, S, R)$ be a start separated LCFRS and $h\colon D_G \to \mathbb{R}_{\geq 0}$ a corpus over abstract syntax trees. The split-merge algorithm(cf. Algorithm 3.3) consists of a loop in which we have a splitting (cf. Algorithm 3.1) and a merging (cf. Algorithm 3.2) part. After each part we train the current PLCFRS-LA with the expectation-maximization algorithm (cf. Algorithm 2.1). The number of latent variables is doubled in the splitting part which refines our PLCFRS-LA. Nevertheless this makes the grammar more complex. Since not every split is beneficial the merging part tries to identify such splits and reverse them. For this purpose we use a threshold μ which gives us a good trade-off between complexity and loss in the likelihood. To be able to restrict the degree of complexity and to prevent our grammar from oversplitting, we also have a parameter ν, which provides us with a minimal rate for merging. All splits whose score is above the threshold μ are merged, then we check if we have reached our required merge rate ν. If this is not the case, we have to merge some split with the highest score until the merge rate is hit.

We start the split-merge algorithm with an initial PLCFRS-LA (G, L_0, γ_0, p_0) where for every $A \in N$ we set $(L_0)_A = \{x\}$. Moreover, we set $\gamma_0(S_x) = 1$ and for every rule $\rho = A_x \to \omega((B_1)_{y_1}, \ldots, (B_k)_{y_k}) \in R[L]$ we set $p_0(\rho) = \frac{1}{|R[L]_{A_x}|}$. It ensures us a sequence of PLCFRS-LAs $(G, L_1, \gamma_1, p_1), (G, L_2, \gamma_2, p_2), \ldots$ such that
$\mathcal{L}(h, \gamma_1, p_1) \leq \mathcal{L}(h, \gamma_2, p_2) \leq \ldots$.

Algorithm 3.3 The Split-Merge Algorithm

```
Require: Let $G = (N, \Delta, S, R)$ be a start separated LCFRS, $G_0 = (G, L_0, \gamma_0, p_0)$ be
    an initial PLCFRS-LA, $\mu \in \mathbb{R}_{\geq 0}$ be the threshold for merging, $\nu \in [0,1]$ be the
    minimal rate of splits to merge, and $h\colon D_G \to \mathbb{R}_{\geq 0}$ is a corpus over abstract syntax
    trees such that $supp(h) \subseteq D_G$.
Ensure: sequence of $G_1, G_2, \ldots$ of PLCFRS-LAs such that $\mathcal{L}(h, \gamma_1, p_1) \leq$
    $\mathcal{L}(h, \gamma_2, p_2) \leq \ldots$.
1: for $i \leftarrow 1, 2, 3, \ldots$ do
2:     $G'_1 \leftarrow \textsc{Split}(G_{i-1})$                ▷ refers to Algorithm 3.1
3:     $G'_2 \leftarrow \mathrm{EM}(G'_1, h)$                  ▷ refers to Algorithm 2.1
4:     $G'_3 \leftarrow \textsc{Merge}(G'_2, h, \mu, \nu)$     ▷ refers to Algorithm 3.2
5:     $G_i \leftarrow \mathrm{EM}(G'_3, h)$
6:     output($G_i$)
```

4 Experimental evaluation of PLCFRS-LA

In this chapter we describe our experimental setup. Figure 4.1 illustrates our experimental pipeline in a high-level way. The first three steps address the induction of an LCFRS from a training corpus. Next, the initial PLCFRS-LA, obtained from the induced LCFRS, is trained and used for parsing. In the end, the parser output is evaluated.

The following sections are each dedicated to a step from the pipeline. Moreover, we present the results for our experiments on corpora of the German and Polish language.

4.1 Experiment Setup

4.1.1 Corpora

We consider corpora over so-called phrases structure trees. Before we continue, let us give a formal definition of phrase structure trees.

Definition 4.1. Consider $S = \{\emptyset\}$ and an $S^* \times S$-sorted alphabet Σ. A *phrase structure tree* (over Σ) is a pair (d, λ) where

- $d \in T_\Sigma$
- $\lambda \colon leaves(d) \to \{1, \ldots, |leaves(d)|\}$ is a bijection assigning a *sentence position* to each leaf.

The sentence of (d, λ) is the string $str(d, \lambda) = d(\lambda^{-1}(1)) \cdot \ldots \cdot d(\lambda^{-1}(|leaves(d)|))$

□

Figure 4.5a illustrates a phrase structure tree, where λ is indicated by the subscripts at the leaves.

In our experiments, we use the corpora provided by the SPMRL shared task 2014. Our experimental pipeline expects corpora in the tiger-xml format [ML00] which the shared task provides only for the languages German, Polish and Swedish. We run our experiments with the German and Polish corpora only, since the Swedish one breaks our experimental pipeline. Moreover, the SPRML shared task provides a split of the corpora into a training, a development, and a test set, which we use. The German corpus contains 5000 phrase structure trees for each of these sets.

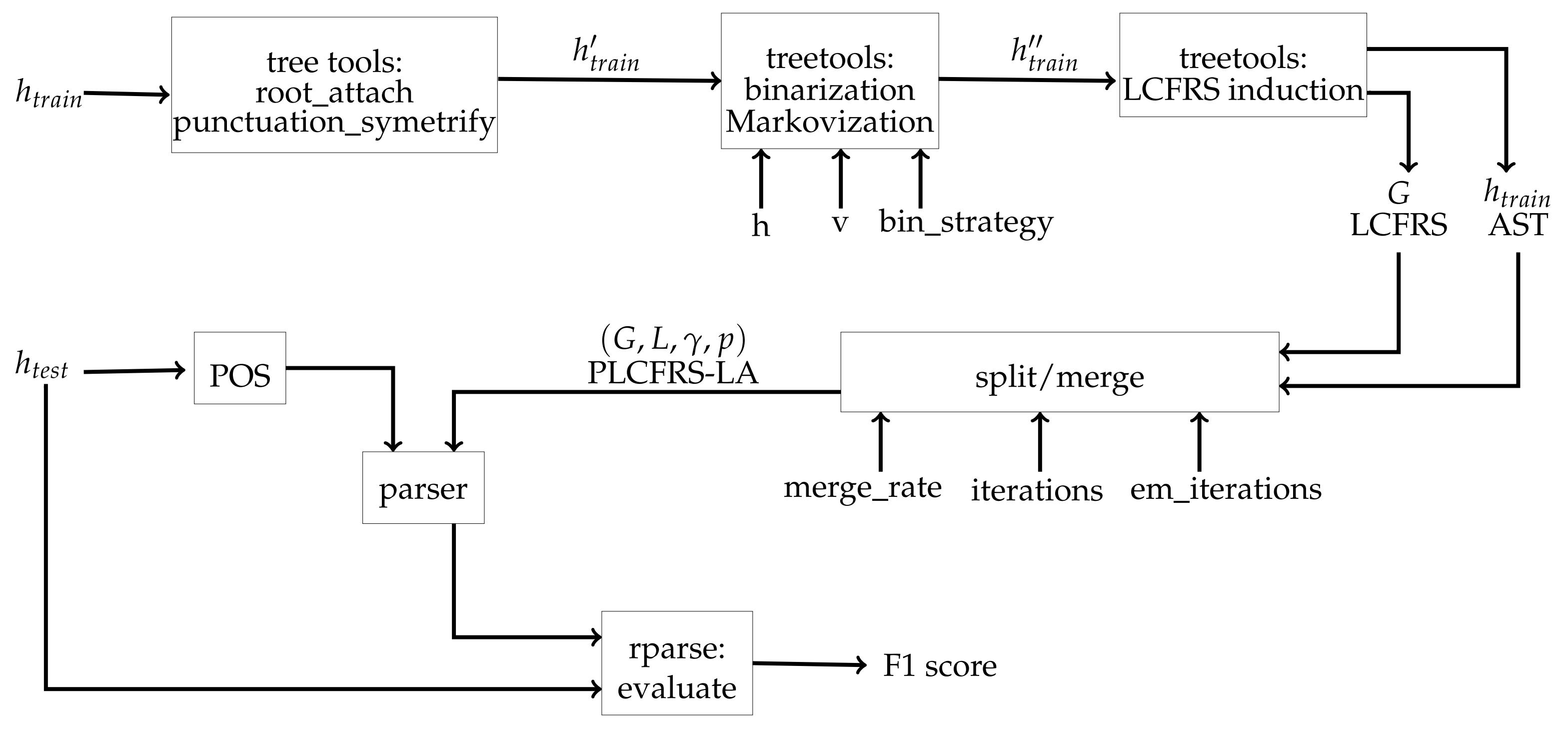


Figure 4.1: high-level illustration of our experimental pipeline

The Polish training, development, and test set contain 5000, 821, and 822 phrase structure trees, respectively.

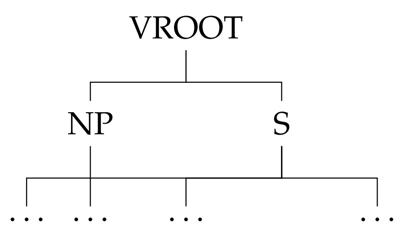
4.1.2 Treetools: root_attach and punctuation_symetrify

We perform two preprocessing steps with treetools[1] to reduce the fanout of the later extracted LCFRS. For instance, punctuation is often attached high in the tree (e.g., at the root) which causes discontinuity. It is not clear what is gained from this high attachment, but it causes higher parsing complexity and (as early experiments showed) leads to results worse than expected (F1 of 40% to 50%). So we apply the transformations `root_attach` and `punctuation_symetrify` to every phrase structure tree in the set. In the first transformation, treetools tries to reorder the tree such that there is only one nonterminal connected to the virtual root node (cf. Figure 4.2). In the second transformation, treetools tries to identify related punctuations such as opening and closing brackets or quotation marks. Afterwards it attempts to move those two punctuation symbols to the same level, as low as possible in the phrase structure tree, while preserving their positions in the linear order of the sentence (cf. Figure 4.3).

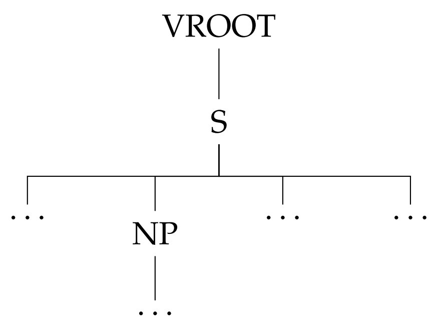
4.1.3 Treetools: Binarization and Markovization

In order to extract a binarized LCFRS in the next step, we perform an optimal binarization and Markovization on our preprocessed training set. This is done with the help of treetools again, using the parameter `optimal` for binarization and `-markov h:1 v:1` for the Markovization. The binarization ensures that each node of each tree in the set has at most two children. Therefore new nodes have the be created and inserted into the trees. The labels of these nodes are chosen accordingly to the parameters of the Markovization. We use 1 as value for both horizontal and vertical Markovization. So the label of a new node is assembled by one sibling node (horizontal) and the original parent node (vertical). For example, in Figure 4.4, the node *VP* has three children *NP*, *VAFIN*, and *VVPP*. After the transformation, the first child of *VP* remains, but the second child is the new node labelled with *@NP-VP*, encoding the last sibling node and the parent. Under this node, the remaining children *VAFIN* and *VVPP* are attached.

[1]https://github.com/wmaier/treetools

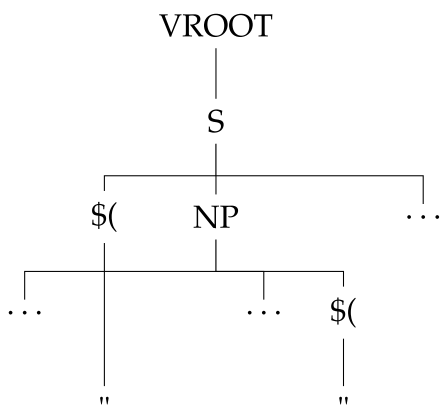


(a) phrase structure tree before `root_attach`

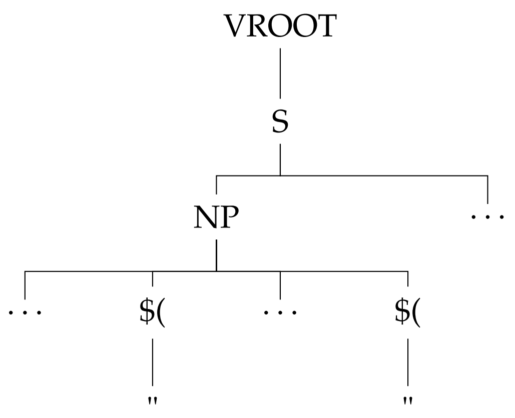


(b) phrase structure tree after `root_attach`

Figure 4.2: applying `root_attach`



(a) phrase structure tree before `punctuation_symetrify`

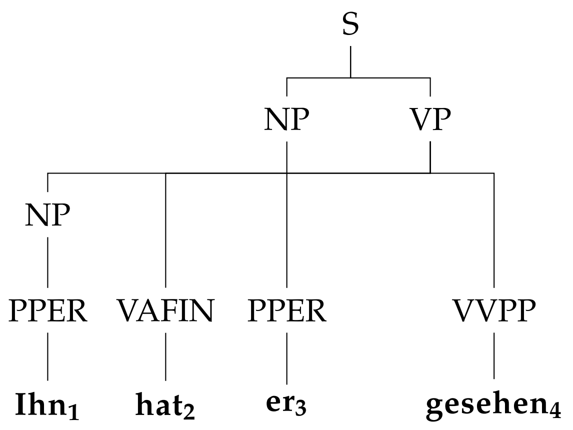


(b) phrase structure tree after `punctuation_symetrify`

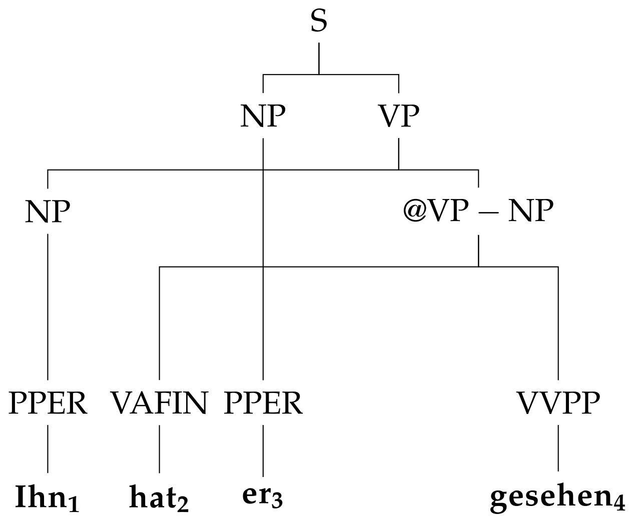
Figure 4.3: applying `punctuation_symetrify`

4.1.4 Treetools: LCFRS Induction

We extract an LCFRS from the binarized and Markovized trees given in our training set (after Section 4.1.3). Since we parse on part-of-speech tags (POS tags), we also remove the leaves from each tree. In consequence, the set of the terminal symbols of the LCFRS is the set constructed of POS tags. An induction method for LCFRS was introduced by Maier and Søgaard [MS08]. A formal treatment is beyond the scope of this bachelor thesis, but we illustrate the induction process by an example. For



(a) phrase structure tree



(b) phrase structure tree after binarization

Figure 4.4: binarization and markovization of a phrase structure tree

simplicity, the example is on a phrase structure tree, which was not binarized and Markovized and which still contains the words as leaves. Consider the node labeled with *VP* from the phrase structure tree in Figure 4.5. We construct the LCFRS rule for this node and its children. The left-hand side of the rule is *VP* and the children *NP, VAFIN*, and *VVPP*, read from left to right, are the nonterminals of the right-hand side. Next we determine which sequences of positions of the sentence are covered by *VP* and which subsequences of these sequences are generated by its children. The node *VP* covers the sequences $(1, 2)$ and (4). Its children *NP*, *VAFIN*, and *VVPP* cover the sequences (1), (2), and (4), respectively. The corresponding strings will be represented by the variables $x_1^{(1)}$, $x_1^{(2)}$, and $x_1^{(3)}$, respectively. Finally, we obtain the rule $VP \rightarrow \langle x_1^{(1)} x_1^{(2)}, x_1^{(3)} \rangle (NP, VAFIN, VVPP)$ as rule.
We use treetools to extract the LCFRS for each single tree separately. Thereby we also obtain the corresponding abstract syntax tree (cf. Figure 4.5). Afterwards we join all extracted LCFRS together into a single one.

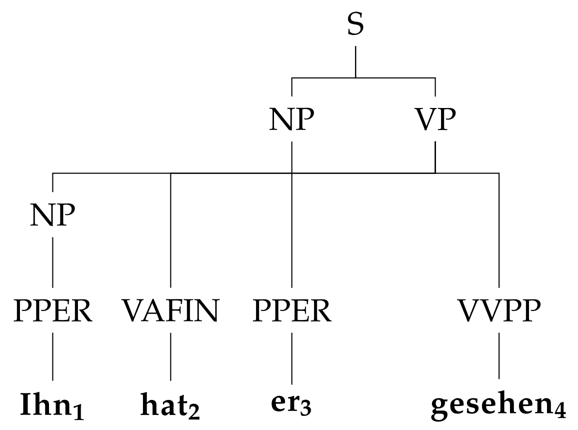
4.1.5 Split-Merge Algorithm

A general purpose implementation of the split-merge algorithm, which can be applied to different grammar classes, was provided by the Chair of Foundations of Programming. It allows to alter the merge rate and the number of iterations for the expectation-maximization algorithm. To initialize the algorithm, we take the set of abstract syntax trees from Section 4.1.4. Moreover, we take the LCFRS, also from Section 4.1.4, and construct an initial PLCFRS-LA as mentioned in Section 3.3.

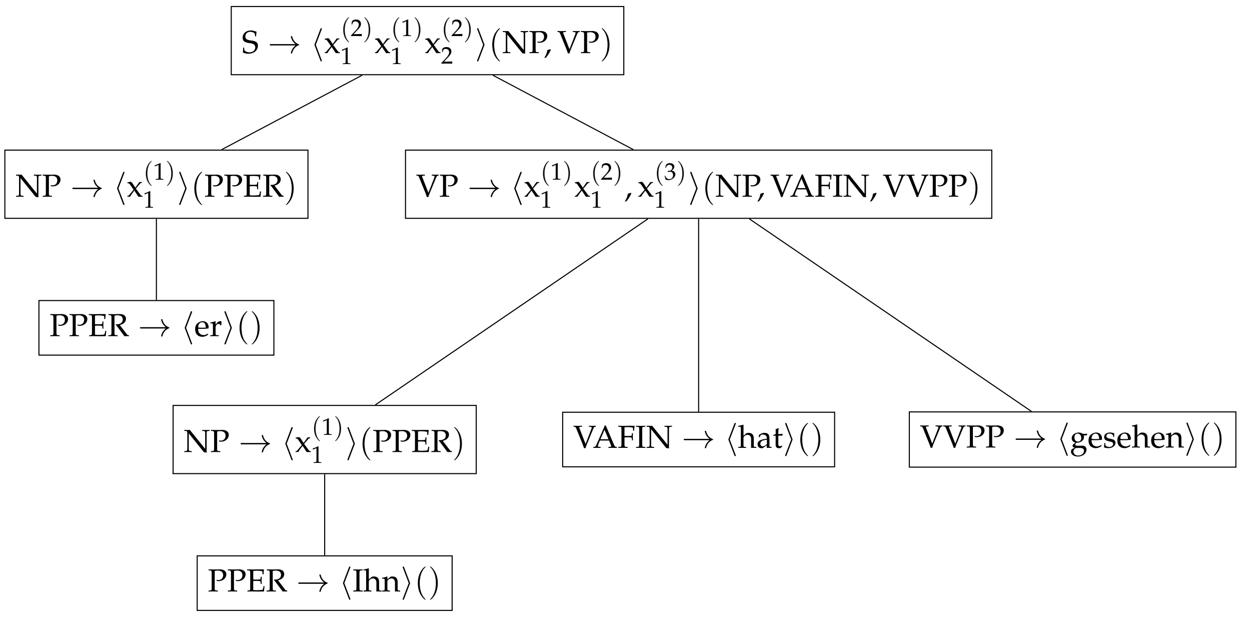
4.1.6 Parser

An implementation of a parser for PLCFRS-LA, which is based on the LCFRS parser by Angelov and Ljunglöf [AL14], was also provided by the Chair of Foundations of Programming. As mentioned earlier we parse on part-of-speech tags (POS). Therefore, for each tree in the set we read-off the POS from the left to the right, resulting in a sequence w. However, we restrict the maximal length of w to 40, otherwise we run out of RAM, while parsing. Next, we use the parser together with our final PLCFRS-LA to find the abstract syntax tree for this sequence of POS with the highest probability (cf. Definition 2.6). The best abstract syntax tree is converted into a phrase structure tree. This conversion is the inverse of the mapping from phrase structure trees to abstract syntax trees, described in Section 4.1.4. In the case the parser could not find any abstract syntax tree for w, we construct a dummy phrase structure tree. Each node of the dummy tree, which is not a leaf, has exactly two children such that

- the left child is always a leaf,
- the right child is either not a leaf and labeled with *NP*, or
- it is a leaf and the rightmost node.



(a) phrase structure tree


$S \rightarrow \langle x_1^{(2)} x_1^{(1)} x_2^{(2)} \rangle (NP, VP)$
$NP \rightarrow \langle x_1^{(1)} \rangle (PPER)$
$VP \rightarrow \langle x_1^{(1)} x_1^{(2)}, x_1^{(3)} \rangle (NP, VAFIN, VVPP)$
$PPER \rightarrow \langle er \rangle ()$
$NP \rightarrow \langle x_1^{(1)} \rangle (PPER)$
$VAFIN \rightarrow \langle hat \rangle ()$
$VVPP \rightarrow \langle gesehen \rangle ()$
$PPER \rightarrow \langle Ihn \rangle ()$

(b) abstract syntax tree

Figure 4.5: a phrase structure tree and its corresponding abstract syntax tree

The leaves from the left to the right are set to the symbols of w. Moreover, the root of this tree is labeled with S.

4.1.7 Evaluation with rparse

After parsing we want to evaluate the quality of the phrase structure trees obtained from the PLCFRS-LA. To this end, we compare the gold standard annotated trees from the test set with the corresponding parsed ones from section 4.1.6, using the evaluation function of rparse[2][MK10]. Specifically, rparse utilizes the PARSEVAL metric [BAF+91] and computes the *LF1* score.
The LF1 score is defined as follows. Consider a phrase structure tree $t = (d, \lambda)$. We define the mapping $cover\colon pos(d) \to labels(d) \times \mathcal{P}(\{1, \ldots, |leaves(d)|\})$ such that for every $q \in pos(d)$ it holds that

$$cover(q) = \Big(d(q), \big(\bigcup_{\substack{w \in (\mathbb{N}_+)^* \\ qw \in leaves(d)}} \{\lambda(qw)\}\big)\Big) \enspace .$$

Next we overload this function by letting

$$cover(t) = \{cover(q) \mid q \in pos(d), \exists\, i, j \in (\mathbb{N}_+)\colon qij \in pos(d)\} \enspace .$$

Now we can define labeled precision and labeled recall, denoted by *LP* and *LR* respectively, as follows: Let g be the gold standard annotated tree from the test set and t the corresponding phrase structure tree produced by the parser.

$$LP_{(g,t)} = \frac{|cover(g) \cap cover(t)|}{|cover(t)|}$$
$$LR_{(g,t)} = \frac{|cover(g) \cap cover(t)|}{|cover(g)|}$$

Since we have a set of gold standard annotated trees and a set of phrase structure trees computed by the parser we have to average *LP* and *LR*. Let $k \in \mathbb{N}_+$, $\{g_1, \ldots, g_k\}$ be the set of gold standard annotated trees, and $\{t_1, \ldots, t_k\}$ be the set of trees computed by the parser such that for every $i \in \{1, \ldots, k\}$, g_i corresponds to t_i. We let

$$\overline{LP} = \frac{\sum_{i=1}^{k} LP_{(g_i, t_i)}}{k} \quad \text{and}$$
$$\overline{LR} = \frac{\sum_{i=1}^{k} LR_{(g_i, t_i)}}{k}$$

[2] https://github.com/wmaier/rparse

and obtain the labeled F1 score

$$LF1 = \frac{2 \cdot \overline{LP} \cdot \overline{LR}}{\overline{LP} + \overline{LR}} \ .$$

Example 4.1. Let g be the phrase structure tree from Figure 4.6a and t be the phrase structure tree from Figure 4.6b

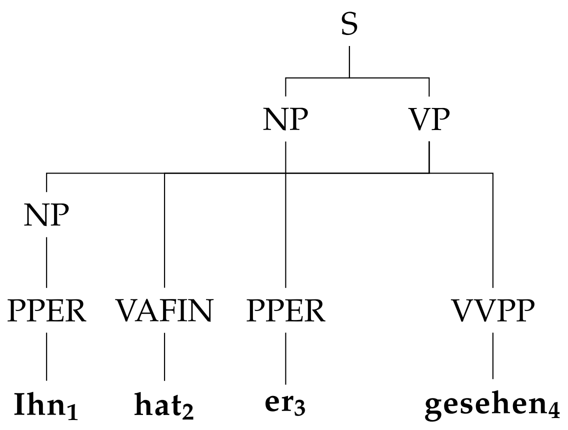
$cover(g) = \{(S, (\{1,2,3,4\})), (NP, (\{3\})), (VP, (\{1,2,4\})), (NP, (\{1\}))\}$ and
$cover(t) = \{(S, (\{1,2,3,4\})), (NP, (\{3\})), (VP, (\{2,4\})), (VP, (\{2,3,4\})), (NP, (\{1\}))\}$.

Now we calculate *labeled precision* and *labeled recall* as follows:

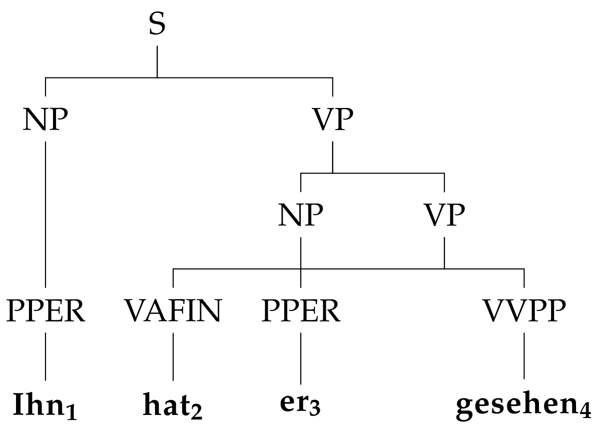
$$\begin{aligned}
LP_{g,t} &= \frac{|cover(g) \cap cover(t)|}{|cover(t)|} \\
&= \frac{|\{(S, (\{1,2,3,4\})), (NP, (\{1\})), (NP, (\{3\}))|}{|\{(S, (\{1,2,3,4\})), (NP, (\{3\})), (VP, (\{2,4\})), (VP, (\{2,3,4\})), (NP, (\{1\}))\}|} \\
&= \frac{3}{5} \\
&= \overline{LP} \\
LR_{g,t} &= \frac{|cover(g) \cap cover(t)|}{|cover(g)|} \\
&= \frac{|\{(S, (\{1,2,3,4\})), (NP, (\{1\})), (NP, (\{3\}))\}|}{|\{(S, (\{1,2,3,4\})), (NP, (\{3\})), (VP, (\{1,2,4\})), (NP, (\{1\}))\}|} \\
&= \frac{3}{4} \\
&= \overline{LR}
\end{aligned}$$

Finally we compute LF1.

$$\begin{aligned}
LF1 &= \frac{2 \cdot \overline{LP} \cdot \overline{LR}}{\overline{LP} + \overline{LR}} \\
LF1 &= \frac{2 \cdot \frac{3}{5} \cdot \frac{3}{4}}{\frac{3}{5} + \frac{3}{4}} = \frac{2}{3}
\end{aligned}$$



(a) gold standard annotated phrase structure tree



(b) phrase structure tree of parser output

Figure 4.6: a gold standard annotated and a corresponding parsed phrase structure tree

4.2 Implementation of our Experimental Pipeline

Our experimental pipeline is implemented in python. The script is located in `lcfrs-sdcp-hybrid-grammars/playground` and can be started by executing

```
PYTHONPATH=..  python2.7 play_split_merge_main.py .
```

The following table gives an overview over the provided parameters as well as a description and the default value for each of them. Possible values for the parameters are given in italic script in the description column.

parameter	default	description
`-binarize`	optimal	set binarization option for treetools *leftright* - simple left-to-right binarization *optimal* - optimal binarization
`-dest`	/tmp	select output path for files
`-doEval`	last	set evaluation frequency *all* - untrained grammar + after each split-merge iteration *cycle* - after each split-merge iteration *last* - after last split-merge iteration
`-dummyLabel`	NP	set label for the inner nodes of the dummy tree

`-dummyRoot`	S	set label for the root node of the dummy tree
`-emEpoche`	20	set number of em-iterations
`-evalCorpus`	dev	select corpus for evaluation *dev* - use development corpus *test* - use test corpus
`-l`	pos	select on what to parse *pos* - parse on part-of-speech tags *form* - parse on words
`-lang`	german	select language *polish, german*
`-markov`	h:1 v:1	Markovization option for treetools in format h:NUMBER v:NUMBER
`-maxLength`	40	set maximal length of the parsed sentence
`-merge`	50	set merge rate in percentage
`-rangeTest`	1 5000	select range of trees for evaluation (from the development or test set, depending on `-evalCorpus`)
`-rangeTrain`	1 5000	select range of trees from the training set to induce a grammar only from those ones
`-seeds`		set list of numbers to initialize the random number generator of the EM-algorithm
`-session`		set this parameter to recover an earlier induced grammar (grammar induction takes very long so using this option saves time if you already have a grammar induced), works only in combination with `-src`
`-smCycle`	2	set number of split-merge iterations
`-src`	/tmp	select path to a folder containing the files for an earlier induced grammar
`-srcCorpus`	../res	select path to the folder containing the unzipped version of the SPRML shared task 2014

`-trans`	root_attach punctuation_symetrify	select supported parameters for transformation with treetools, see treetools for available transformations and their effects

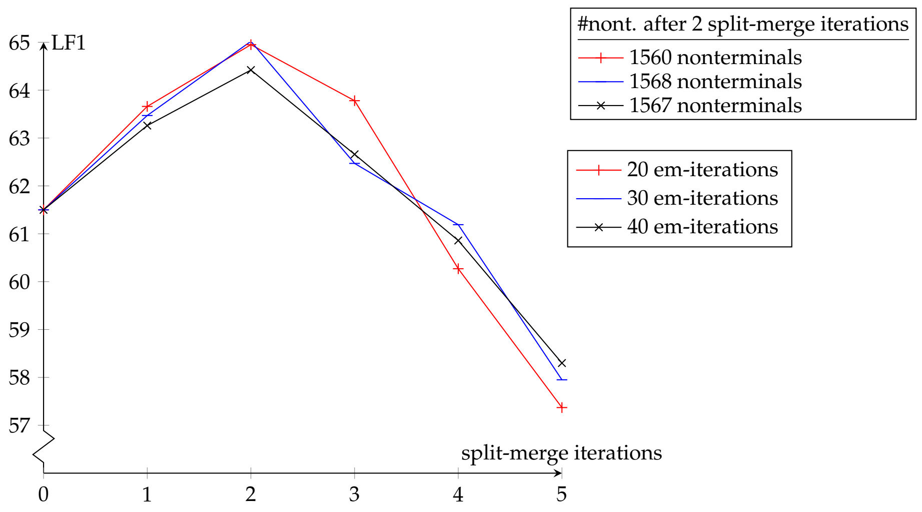
4.3 Results of the Experiments

We divided our experiments into two parts. In the first one, we tested different values for the individual parameters of our split-merge algorithm to determine promising choices. Therefore, we evaluated our grammar on the development set. The default seed for the random number generator (RNG) of the expectation-maximization algorithm for these experiments was 0. In the second part, we initialized the RNG of the expectation-maximization algorithm with four random seeds and repeated the experiments, using the parameters for the best grammar (initializing the RNG with random seeds leads to different starting points of the expectation-maximization algorithm and therefore to different LF1 scores for the grammar, as shown by Petrov [Pet10]). In the end, we chose the grammar with the highest LF1 score and evaluated it on the test set.

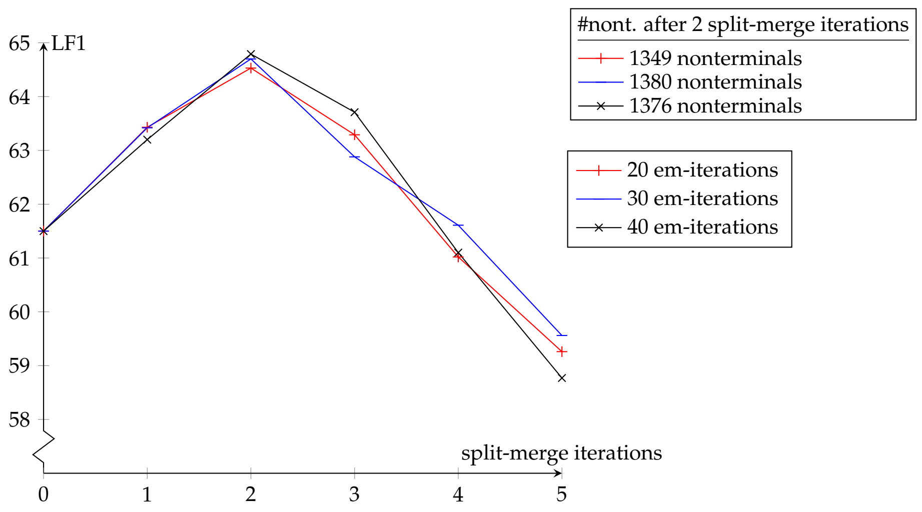
4.3.1 Results for the German Corpus

The LCFRS which we induce has 711 nonterminals. We obtain an initial PLCFRS-LA as described in Section 3.3 which reaches an LF1 score of 61.50% on the development set. We oriented ourselves towards Petrov et al. [PBTK06] and started with merge rates around 50%. The results are shown in Figure 4.7. Each of this three diagrams depicts the LF1 score when parsing the development set on the y-axis. The x-axis shows the number of split-merge iterations. Each diagram corresponds to a certain merge rate and contains a line for a different number of iterations for the expectation-maximization algorithm (em-iterations). Each of this diagrams shows, that the LF1 score grows after 1 split-merge iteration to about 63.5% and reaches its maximum after 2 split-merge iterations. Afterwards, the PLCFRS-LA seems to adapt too well to training data and does not generalize anymore which makes the LF1 score go down. In addition, the lines for the different em-iterations tend to overlap with increasing merge rate. It is striking that the LF1 score improves only minimally during the split-merge training. 3.52% is the maximal improvement we get, using a merge rate of 50% and 30 em-iterations. This is far away from the improvements achieved by Petrov et al. [PBTK06].

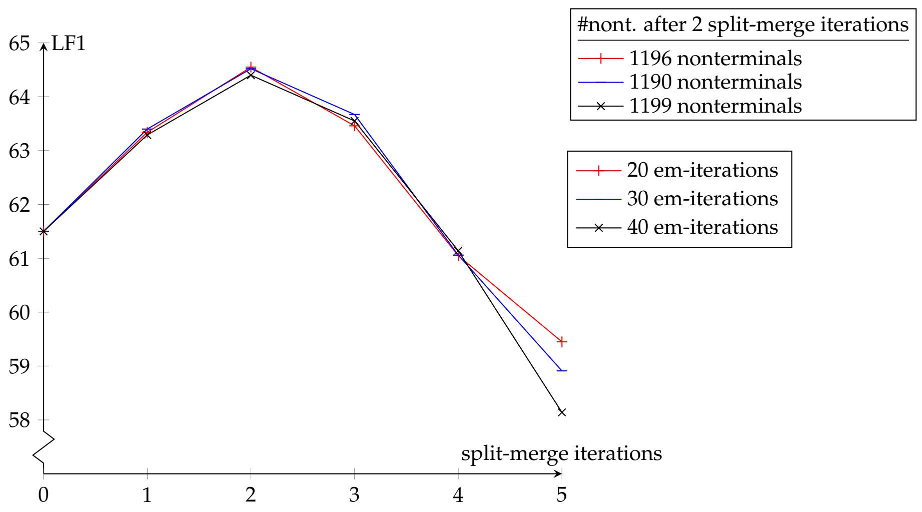
Since the number of nonterminals is located well over 1000, it is possible that the trained latent annotation are too fine which means the context dependencies are too strong. Therefore we attempt higher merge rates and a greater number of em-iterations. It turned out that the maxima of the LF1 scores are located after 2 split-



(a) merge rate of 50%



(b) merge rate of 60%



(c) merge rate of 70%

Figure 4.7: LF1 score dependent on split-merge iterations (german corpus)

em-iterations: / merge rate:	20	30	40	50	60	70	80	90	100
40	64.49	64.41	64.86	64.27	64.48	64.75	64.16	64.23	64.46
50	64.95	65.02	64.42	65.02	64.66	64.77	64.74	64.27	64.25
60	64.53	64.70	64.79	64.34	64.93	64.55	64.52	64.69	64.28
70	64.55	64.52	64.40	64.51	64.45	64.83	64.74	64.52	64.32
75	64.31	63.65	64.64	64.29	64.33	64.20	64.66	65.05	**65.08**
80	64.45	**63.33**	63.33	64.71	64.59	64.47	64.26	64.49	64.56
85	64.63	64.08	63.77	64.29	64.53	64.70	64.34	64.50	64.83
90	64.59	63.77	63.59	64.80	64.91	64.92	64.83	63.59	64.77
95	63.55	63.59	64.20	64.48	64.44	64.42	64.03	63.37	63.95

Figure 4.8: LF1 scores after 2 split-merge iterations for all tested parameters (german corpus)

seeds: / parameters:	2364	64536	342632	846532
50% merge rate, 30 em-iterations	64.09	64.37	64.29	64.46
50% merge rate, 50 em-iterations	64.78	64.16	**63.89**	64.90
75% merge rate, 100 em-iterations	64.62	64.18	64.63	**65.30**

Figure 4.9: LF1 scores for the four random seeds (german corpus)

merge iterations again. Taking a look at Figure 4.8, it becomes clear that the LF1 scores for the different merge-rates and em-iterations lie close together. It seems that the different values for the parameters influence the LF1 score only minimally. Moreover, we get a minimal increase of 0.06% towards the best LF1 score, reaching 65.08% for it by using a merge rate of 75% and 100 em-iterations.

Since we have three candidates with a similarly good LF1 score, we repeated the experiments with the random seeds for each of them. The candidates are 50% merge rate and 30 em-iterations, 50% merge rate and 50 em-iterations, and 75% merge rate and 100 em-iterations. Figure 4.9 shows the results of these runs. Only the last candidate could increase its LF1 score to a new high at 65.30%. The others remain behind the previously achieved scores. We use this grammar for the evaluation on the test set, where it reaches an LF1 score of 59.69%.

em-iterations: / merge rate:	20	30	40	50	60	70	80	90	100
40	90.10	89.48^1	89.79	90.05	89.84	89.85	89.95	89.83	89.53
50	90.10	89.48	89.79	90.16	89.86	89.84	90.04	89.81	89.57
60	90.09	89.58^1	89.68^1	90.07	90.11	90.18	89.58	89.75	89.49
70	90.15	89.49	89.51^1	89.34	89.22	89.36	89.58	89.57	89.68
75	89.60	89.94	89.89	89.62	89.52^1	89.41^1	89.57^1	89.72^1	89.90
80	89.99	89.53	89.45	89.42	89.50	89.84	89.82	89.70	89.87
85	89.67	89.71	89.81	89.63	89.65	89.90	89.99	89.76	89.68
90	89.62	89.57	88.98^1	90.35	90.01	89.82	90.18	90.31	**90.48**
95	89.42	89.76	89.44	89.12	**85.32**	85.32	85.32	85.32	85.32

Figure 4.10: the maximum LF1 scores for all tested parameters (polish corpus)

seeds:	2364	64536	342632	846532
LF1 score:	89.96	**89.68**	90.11	**90.30**

Figure 4.11: LF1 scores for the four random seeds using 90% merge rate and 100 em-iterations (polish corpus)

4.3.2 Results for the Polish Corpus

The LCFRS which we induce has 114 nonterminals. We obtain an initial PLCFRS-LA as described in Section 3.3 which reaches an LF1 score of 85.91% on the development set. We have transferred the experiences from the experiments with the German corpus and tested the same values for merge rate and em-iterations. It turned out that the maximum of the LF1 score is sometimes reached after 2, but more often after 3 split-merge iterations. This is one split-merge iteration more than for the German corpus. Striking is the behavior of the LF1 scores for a merge rate of 95% and 60, 70, . . ., 100 em-iterations. These remain constant at 85.32% over all 6 tested split-merge iterations. Compared to the untrained grammar that is a loss of 0.59%. Figure 4.10 shows the maximal reached LF1 score in dependence of the different parameter values. Generally, these are the LF1 scores after 3 split-merge iterations, except the number is annotated by the superscript 1, in which cases the LF1 score after 2 split-merge iterations is shown. The highest LF1 score is 90.48% which is achieved with a merge rate of 90% and 100 em-iterations after 3 split-merge iterations. This is an increase of 4.57% compared to the untrained grammar. Again, the influence of the different values for the parameters seems to be minimal. The LF1 scores lie close together, as in the case of the German corpus.

Once again we repeated the experiment with the random seeds. Since the difference between the best LF1 score and the other ones is greater than for the German corpus, we only considered a merge rate of 90% and 100 em-iterations. Figure 4.11 illustrates the results for this run. We get slightly worse values in comparison to the previously achieved LF1 score. The last random seed brings us again the best value with an LF1 score of 90.30%. Finally, this grammar reaches an LF1 score of 89.87% on the test set.

5 Conclusion

We introduced latent annotation for LCFRS and extended that to PLCFRS-LA as a formalism that can be used for parsing of natural language sentences. Next we defined an expectation-maximization algorithm, as well as a splitting and merging step, which we brought together to a split-merge algorithm to train PLCFRS-LA.

In the end, we tested the usefulness of this split-merge algorithm for PLCFRS-LA. To this end we used corpora of the German and Polish language. We examined if the quality of the grammar gets significantly better, or if the impact is not substantial. The experiments on the Polish corpus showed an improvement of approximately 4.50% on a rather good initial grammar while the German corpus showed only an improvement of approximately 3.50% on an actually poor initial grammar. Petrov et al. [PBTK06] who also started with an initial grammar which has a very low parsing performance got a clearly greater improvement for PCFG-LA on English. However, they used a validation set to stop em-training before overfitting occurs. Possibly this is necessary for the German language to prevent overfitting, but less for Polish. Investigating the reasons for this difference and repeating the experiments, using a validation set, is a task for future work.

Bibliography

[AL14] Krasimir Angelov and Peter Ljunglöf. Fast statistical parsing with parallel multiple context-free grammars. *Proceedings of the 14th Conference of the European Chapter of the Association for Computational Linguistics*, pages 368–376, April 2014.

[BAF^{+}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. *Proceedings of the Workshop on Speech and Natural Language*, pages 306–311, 1991.

[MK10] Wolfgang Maier and Laura Kallmeyer. Discontinuity and nonprojectivity: Using mildly contextsensitive formalisms for data-driven parsing. *Proceedings of TAG*, page 10, 2010.

[ML00] Andreas Mengel and Wolfgang Lezius. An xml-based representation format for syntactically annotated corpora. *Proceedings of the International Conference on Language Resources and Evaluation*, pages 121–126, 2000.

[MMT05] Takuya Matsuzaki, Yusuke Miyao, and Jun'ichi Tsujii. Probabilistic cfg with latent annotations. *Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics*, pages 75–82, 2005.

[MS08] W. Maier and A. Søgaard. Tree-banks and mildly context-sensitivity. *Proc. of Formal Grammar 2008*, page 61–76, 2008.

[PBTK06] Slav Petrov, Leon Barrett, Romain Thibaux, and Dan Klein. Learning accurate, compact, and interpretable tree annotation. *Proceedings of the 21st International Conference on Computational Linguistics and the 44th Annual Meeting of the Association for Computational Linguistics*, pages 433–440, 2006.

[Pet10] Slav Petrov. Products of random latent variable grammars. *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pages 19–27, 2010.

[VSWJ87] K. Vijay-Shanker, David J. Weir, and Aravind K. Joshi. Characterizing structural descriptions produced by various grammatical formalisms. *Proceedings of the 25th Annual Meeting on Association for Computational Linguistics*, pages 104–111, 1987.