EPSILON-VERARBEITUNG BEI MINIMALISTISCHEN GRAMMATIKEN FÜR ZAHLEN

Johannes Kuhn, Matthias Wolff, Borislav Borislavov

BTU Cottbus-Senftenberg
johannesfk.kuhn@b-tu.de

Kurzfassung: Um bei Minimalistischen Grammatiken (MG) Übergenerierung zu vermeiden, kann man Einträge mit leeren Exponenten (ε-Einträge) verwenden. Ein Eintrag besteht aus einem Exponenten, der die Äußerung oder Schrift eines Wortes repräsentiert, einer Merkmalsliste, welche die Syntax kodiert und einem λ-Ausdruck, der die Semantik repräsentiert. Leere Einträge führen allerdings zu einer schlechteren Verwendbarkeit der Grammatik für das Parsen. Die vorliegende Arbeit wird ein Umformungsalgorithmus für MGs vorstellen, sodass die Anzahl der ε-Einträge verringert werden kann, um sie wieder für Parser verwendbar zu machen. Hierzu werden die ε-Einträge mit den anderen Einträgen vorverarbeitet und dadurch neue Einträge geschaffen. Die nun überflüssigen ε-Einträge können dann problemlos entfernt werden. Der Algorithmus wurde anhand von über 260 Zahlwortgrammatiken getestet.

1 Motivation

In [1] wurden Minimalistische Grammatiken (MG) verwendet, um eine bidirektionale Übersetzung von symbolischen Folgen zu Semantik zu realisieren. Später konnte dann in [2, 3] gezeigt werden, dass es generell möglich ist, Zahlwörter im MG-Formalismus automatisch zu lernen, um sie später für bidirektionale Verarbeitung zwischen Mensch und Maschine zu verwenden. Inspiriert davon wurden nach Vorbild von [4, 5] mehr als 260 Zahlengrammatiken für MGs in unterschiedlichen Sprachen und Dialekten erstellt [6]. Eine Verwendung von MGs in der Sprachverarbeitung zum semantisch korrekten Erfassen von menschlicher Sprache ist also denkbar, ebenso wie die Generierung natürlichsprachlicher Texte basierend auf maschinensprachliche Ausdrücke.

Die geradlinigste Implementierung einer *Natural Language Processing*-Anwendung (NLP) zur Sprachverarbeitung mit MGs ist, einen aktuellen Parser für MGs zur Verarbeitung von Sätzen zu nehmen, der nur anhand der Merkmalsliste (Syntax) und der Eingabe (Exponenten) arbeitet. In einem zweiten Schritt wird der entstehende Ableitungsbaum in Bottom-Up-Richtung ein weiteres Mal durchlaufen, um Semantikausdrücke hinzuzufügen. So entsteht schlussendlich an der Wurzel des Ableitungsbaumes ein Äußerungs-Bedeutungs-Paar des eingegebenen Satzes, welches für ein gegebenes Lexikon einen semantischen Ausdrucks mit einer Äußerung verbindet (Römer et al. [1]). Dies wurde mit den Grammatiken aus [6] mit dem Left-Corner-Parser (LC-Parser) von Stanojević und Stabler [7] getestet. Dabei kam heraus, dass der Parser beim Großteil der zu parsenden Zahlwörter nach 5 Minuten laufenden Parsens abgebrochen werden musste. Der längste Testlauf wurde nach 3 Tagen erfolglos abgebrochen. Die Existenz der Ableitungsbäume konnte problemlos per Hand nachgewiesen werden. Als Problem wurden Einträge in den Grammatiken identifiziert, die leere Exponenten haben. Ein Ersetzen der leeren Exponenten durch einheitliche Platzhaltersymbole resultierte in eine Parsierungszeit von bis zu 5 Stunden.

 DOI: 10.35096/othr/pub-7083, URN: urn:nbn:de:bvb:898-opus4-70830

Um bei minimalistischen Grammatiken (MG) Übergenerierung zu vermeiden, kann man Einträge mit leeren Exponenten (ε-Einträge) verwenden. Leere Einträge führen allerdings zu einer schlechteren Verwendbarkeit der Grammatik für das Parsen. So z. B. mit dem getesteten LC-Parser nach [7], wo eben jene ε-Einträge zu mehr Backtracking bis hin zu Endlosschleifen während des Parsen führen können. Der Anteil an ε-Einträgen bei den Grammatiken in [6] liegt im Bereich von 20-50%. Dies deutet darauf hin, dass das Problem von ε-Ausdrücken, wie schon von anderen Formalismen bekannt (siehe z.B. [8]), keine vernachlässigbare Randerscheinung darstellt. Überlegungen zur Transformation von MGs wurden bereits in anderen Arbeiten vorgestellt [9, 10, 11], wobei dort das Ziel verfolgt wurde, MGs für ausgewählte Anwendungen zu optimieren, die nicht primär dem Parsen gelten. Bei Graf et al. [9] wurden Lexika in eine *single movement normal form* transformiert, die motiviert war, das umstrittene Shortest-Movement-Constraint für MGs nicht benutzen zu müssen. Die resultierenden Lexika zeichnen sich auch durch eine Erhöhung der Anzahl an Einträgen aus. Bei Kobele [10] und bei Ermolaeva [11] ging es darum, bestehende Einträge in einem Lexikon zu verallgemeinern im Bezug auf die Unterschiedlichkeiten der Merkmalsliste im Lexikon. Für die resultierenden Lexika wurde keine Auskunft ausgegeben, ob das Parsen mit ihnen sich dadurch verbesserte. Bei anderen Grammatikformalismen, wie zum Beispiel *Tree Adjoining Grammar* (TAG), wurden Überlegungen zur Verringerung von ε-Ausdrücken bereits eingehender untersucht (siehe z. B. [12]). Im vorliegenden Beitrag wird ein Umformungsalgorithmus für diese und ähnliche MGs vorgestellt, sodass die Anzahl der ε-Einträge verringert werden kann, um sie wieder für Parser wie in [7] verwendbar zu machen.

2 Grundlagen und Aufbau

Die in dieser Arbeit verwendete Darstellung der MG ist angelehnt an [3]. Ein Wort wird bei den MG durch einen Eintrag (engl. *Item*) repräsentiert, bestehend aus einem Exponenten π, einer Liste aus Merkmalen T und einem λ-Ausdruck σ. Die Sammlung dieser Triple (π,T,σ) wird Lexikon genannt und definiert den gesamten Wortschatz, den ein System zum Zeitpunkt eines Parse- oder Generierungsvorgangs zur Verfügung hat. Jedes Triple bekommt noch einen Typ $p \in \{::, :\}$, der kennzeichnet, ob ein Eintrag für die Verarbeitung direkt aus dem Lexikon genommen wurde (::), oder ob der Eintrag Produkt einer vorangegangen Operation war (:). Einträge mit (::) werden lexikalische Einträge genannt (engl. *Lexical Item*, kurz LI) und Einträge mit (:) werden abgeleitete Einträge genannt (engl. *Derived Item*, kurz DI). Zu dem Lexikon besitzen die MG in der hier verwendeten Form zwei Operationen zur Verarbeitung von LI und DI, *merge* und *move*, die es in drei bzw. zwei Ausprägungen gibt. Die Abbildung 1 gibt einen kurzen Überblick über deren Verhaltensweisen. Eine Verkettung von DI, etwa nach einem oder mehreren *merge*3-Operationen, wird *Kette* genannt. Werden die Exponenten von Einträgen kombiniert (etwa nach *merge*1 oder *move*1), wird das Ergebnis als neuer eigenständiger DI betrachtet.

Für den Algorithmus ist es noch wichtig zwischen zwei weitere Unterarten von Einträgen zu unterscheiden:

- η-Einträge: Einträge deren Exponenten nicht leer sind. DI mit nichtleeren Exponenten heißen η-DI. Bei LI heißt es dementsprechend η-LI.

- ε-Einträge: Einträge deren Exponenten leer sind. DI mit leeren Exponenten heißen ε-DI. Im Falle von LI heißt es dementsprechend ε-LI.

Wie oben erwähnt, können ε-LI verwendet werden, um Übergenerierung zu vermeiden. Hierzu werden die Merkmalslisten der η-LI so aufeinander abgestimmt, dass nur bestimmte Merkmalsfolgen miteinander erfolgreich kombiniert werden können. Um ein erwünschtes η-LI

merge 1 $$\frac{(\pi_1,::=f\ T_1,\sigma_1)\quad(\pi_2,\cdot f,\sigma_2),\alpha_1,...,\alpha_n}{(\pi_1\pi_2,:T_1,\sigma_1\sigma_2),\alpha_1,...,\alpha_n}$$

merge 2 $$\frac{(\pi_1,:=f\ T_1,\sigma_1),\alpha_1,...,\alpha_n\quad(\pi_2,\cdot f,\sigma_2),\beta_1,...,\beta_m}{(\pi_2\pi_1,:T_1,\sigma_1\sigma_2),\alpha_1,...,\alpha_n,\beta_1,...,\beta_m}$$

merge 3 $$\frac{(\pi_1,\cdot=f\ T_1 k\sigma_1),\alpha_1,...,\alpha_n\quad(\pi_2,\cdot f\ T_2,\sigma_2),\beta_1,...,\beta_m}{(\pi_1,:T_1,\sigma_1),\alpha_1,...,\alpha_n,(\pi_2,:T_2,\sigma_2),\beta_1,...,\beta_m}$$

move 1 $$\frac{(\pi_1,:+f\ T_1,\sigma_1),\alpha_1,...,\alpha_n,(\pi_2,:-f,\sigma_2),\beta_1,...,\beta_m}{(\pi_2\pi_1,:T_1,\sigma_1\sigma_2),\alpha_1,...,\alpha_n,\beta_1,...,\beta_m}$$

move 2 $$\frac{(\pi_1,:+f\ T_1,\sigma_1),\alpha_1,...,\alpha_n,(\pi_2,:-f\ T_2,\sigma_2),\beta_1,...,\beta_m}{(\pi_1,:T_1,\sigma_1),\alpha_1,...,\alpha_n,(\pi_2,:T_2,\sigma_2),\beta_1,...,\beta_m}$$

Abbildung 1 – Operationen des MG-Formalismus. π_i sind Zeichenketten, (::) und (:) sind der Typ des Eintrages. Als Platzhalter steht $(\cdot)$ an Stellen an denen ein beliebiger Typ stehen kann. T_i sind Merkmalslisten und f ist ein einzelnes Merkmal. Die Vorzeichen $+,-,=$ oder kein Vorzeichen markieren an welche Art von Operation ein Merkmal teilnehmen kann. *Positive* Merkmale sind diejenigen, die als Vorzeichen entweder $+$ oder $=$ haben. Merkmale ohne Vorzeichen oder $-$ werden als *negative* Merkmale bezeichnet. σ_i sind λ-Ausdrücke. α_i und β_j sind DI.

für mehrere Exponentenfolgen mit unterschiedlichen Generierungseinschränkungen nutzen zu können, werden ε-LI verwendet. Damit werden die entsprechenden Merkmalslisten während der Bearbeitung erstellt, ohne dabei unerwünschte η-LI zuzulassen. Eine detaillierte Beschreibung der Generierung solcher Lexika sind in [13] beschrieben.
Dies führt zu den Voraussetzungen an die Lexika für den in dieser Arbeit beschriebenen ε-Auslöschungsalgorithmus:

1. Solange ε-Einträge noch *positive* Merkmale haben, können sie nur mit einem η-LI oder η-DI in der Kette an Operationen beteiligt sein.

2. Wenn ein η-Eintrag mit einer Kette an einer Operation beteiligt ist, darf das erste Element der Kette kein ε-DI sein.

3. Wenn zwei η-Einträge an einer *merge*-Operation teilnehmen, dann kommt keine weitere *merge*-Operation mehr vor, bis nur noch ein eigenständiges DI existiert, welches nicht Teil einer Kette ist.

Die Voraussetzungen sind in abfallender Wichtigkeit für den Algorithmus aufgelistet. Die Umwandlung sollte noch für Grammatiken funktionieren, die Voraussetzung 3 nicht erfüllen, aber zum Zeitpunkt dieses Papers konnte dies mangels entsprechender Lexika noch nicht getestet werden. Lexika, die Voraussetzung 2 verletzen, könnten nach der Transformation unter Umständen nicht die gleiche Sprache wie vorher darstellen. Sollten Lexika die Voraussetzung 1 nicht erfüllen, wird das durch den Algorithmus erzeugte Lexikon nicht die gleiche Sprache darstellen, wie davor. In dem letzten Fall wäre durch eine Sonderbehandlung im Algorithmus, der ε-Einträge welche die Voraussetzung verletzen, eine korrekte Umwandlung möglich.

3 ε-Auslöschung

Um die Anzahl der benötigten ε-Einträge einer Grammatik zu verringern, werden die ε-Einträge mit den anderen Einträgen abgeglichen und ggf. fusioniert. Dazu wird überprüft, ob die negativen Merkmale eines η-Eintrags, mit den positiven Merkmalen eines oder mehrerer ε-Einträge kompatibel sind. Bei Übereinstimmung wird ein neuer η-Eintrag generiert, der die positiven Merkmale des ursprünglichen η-Eintrags und die verbleibenden negativen Merkmale der ε-Einträge hat. Um die in [3] vorgestellte semantische Form beizubehalten, muss beim Vergleich

der Einträge beachtet werden, wann entsprechende Merkmalslisten leer werden. Diese Reihenfolge muss auf die zugehörigen Lambda-Ausdrücke übertragen werden, um die ursprüngliche Semantik gewährleisten zu können. D. h. die Reihenfolge, in der die entsprechenden Merkmalslisten leer werden, muss auch die Reihenfolge sein, in der die Semantikausdrücke bei einem neu erstellten LI geordnet werden. Wenn alle möglichen Einträge mit dem Algorithmus generiert wurden, können die redundanten ε-Einträge aus der Grammatik gelöscht werden.

Der Ablauf der Transformation[1] ist folgender:

1. Alle negativen Merkmale von η-LI werden der Reihe nach mit den passenden positiven Merkmalen der ε-LI kombiniert, sodass von beiden keines übrigbleibt. Es werden neue LI aus den verbleibenden Merkmalslisten mit dem Exponent des η-LI erzeugt, falls dieses nicht bereits existieren.

2. Es wird versucht bestehende Ketten mit ε-LI zu kombinieren, sodass mindestens 1 Merkmal des verwendeten η-LI wegfällt und alle positiven Merkmale des neuen ε-LI. Aus dem Ergebnis werden neue LI und/oder Ketten, mit dem Exponenten des ε-LI und den verbleibenden Merkmalslisten erzeugt.

3. Es werden alle Ketten gelöscht, die nicht in dieser Runde neu erzeugt wurden.

4. Alle negativen Merkmale von η-LI werden mit den positiven Merkmalen der ε-LI kombiniert, sodass nur negative Merkmale vom ε-LI und η-LI übrigbleiben. Es werden neue Ketten daraus erzeugt, falls diese nicht bereits vorher existiert haben.

5. Falls keine neuen Ketten oder LI diese Runde erzeugt wurden, wird zu Schritt 6 gegangen. Andernfalls wird mit Schritt 1. das Verfahren ein weiters Mal fortgesetzt.

6. Es werden alle ε-LI gelöscht, die an mindesten einer Kombination beteiligt waren.

Durch die regelmäßige Löschung aller Ketten, die sich in einer Iteration nicht verändert haben bleibt die potentielle Anzahl an Ketten endlich. Die Speicherung aller Versuche eine Kette zu bilden ist stellt sicher, dass es nicht zu unendlicher Generation von Ketten und damit einer unendlichen Laufzeit kommen kann. Wegen der Vermeidung von Unendlichkeitsschleifen in der Verarbeitung, wird auch bei dem Bilden von Ketten nur dann neue ε-LI zugelassen, wenn diese auch die Merkmalsliste des η-Eintrags berühren. Alternativ könnte dies auch mit einer festgelegten maximalen Anzahl an Kettengliedern gelöst werden. Hier wurde sich für die erste Variante entschieden, um eine nur von den Lexika abhängiges Verhalten zu erhalten.

Um Lexika zu transformieren, welche die Voraussetzung 1 nicht erfüllen kann der Algorithmus etwas abgeändert werden, sodass das resultierende Lexikon auch noch die gleiche Sprache wie vorher repräsentiert. Hierzu werden nicht mehr nur die η-LI mit den ε-LI kombiniert, sondern nun werden auch diejenigen ε-LI, welche die Voraussetzung 1 verletzen, mit den anderen ε-LI kombiniert. Die Folge daraus ist, dass nach der Transformation deutlich mehr ε-LI übrigbleiben, aber nur für Lexika die Voraussetzung 1 nicht erfüllt haben. Bei den restlichen Lexika bleibt das Verhalten wie vor dieser Abänderung.

4 Auswertung

Für die Auswertung des Algorithmus wurden die Zahlwortlexika von [6] benutzt. Von den 263 dort zur Verfügung gestellten Lexika wurden drei ausgesondert, da für die verwendeten Zeichen

[1] Eine Implementierung ist auf Github zu finden: `https://github.com/FancyOink/MG-Lexikon-Transformer.git`

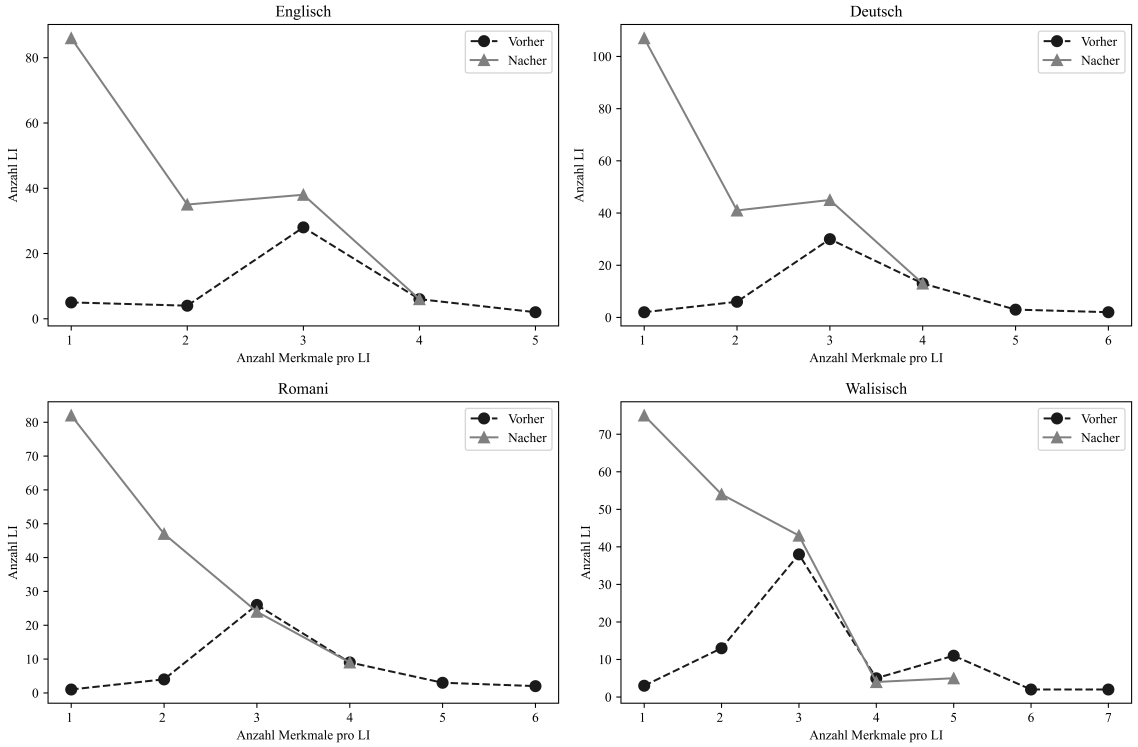

Abbildung 2 – Die Aufkommensverteilung der Anzahl an Merkmalen pro LI für vier verschieden Sprachen, vor und nach der Transformation. Es steigt nach der Transformation deutlich der Anteil an Einträgen mit kurzen Merkmalslisten. Die maximale Länge an Merkmalslisten sinkt hingegen durch die Transformation.

im Exponenten für die Implementierung keine korrekte Kodierung gefunden werden konnte. Von den verbleibenden Zahlwortlexika verletzten 36 die Voraussetzung 1 für den Algorithmus und wurden dem entsprechend ebenfalls aussortiert. Damit verblieben 224 Zahlwortlexika. Bei diesen kam es vor, dass Voraussetzung 2 für den Algorithmus verletzt wurde. Dies konnte aber durch gezieltes Einfügen von ε-LI berichtigt werden. Die Lexika erzeugen, sofern die Originalquelle es zu ließ, Zahlwörter im Wertebereich von 1 bis 999. Die Lexika haben im Mittel 61 Einträge mit einer Standardabweichung von 33. Der Anteil an ε-LI beträgt im Mittel 37% mit einer Standartabweichung von 12 Prozentpunkten.

Der schlimmstenfalls durch ε-Eliminierung zu erwartende quadratische Zuwachs an generierten Lexikoneinträgen ($m \cdot (n-1)$; mit m: Anzahl ε-LI, n: Anzahl η-LI) konnte i. A. nicht beobachtet werden. Der Zuwachs an Einträgen beträgt stattdessen im Mittel 229% mit einer Standartabweichung von 63%-Punkten. Der maximale zu beobachtende Zuwachs betrug 387%, der minimalste Zuwachs betrug 59%. Bei 178 Lexika wurden alle ε-LI erfolgreich entfernt. Die restlichen 46 Lexika behielten nur ein ε-LI, welches von Anfang an keine positiven Merkmale besaß, weswegen der Algorithmus es auch nicht beeinflussen konnte. Des Weiteren war eine Reduktion der durchschnittlichen Anzahl von Merkmalen pro LI von 42% zu beobachten. Dies ist nicht nur mit einer Zunahme von LI mit wenigen Merkmalen zu erklären (Abbildung 2 zeigt die Verteilungen der Merkmale pro Eintrag für einige ausgewählte Lexika vor und nach der Transformation). Die maximale Länge der Merkmalslisten der Lexika sank auch im Mittel um 28%. Jedes involvierte Merkmalspaar in einem Parsierungsverfahren entspricht einem Knoten im späteren Ableitungsbaum, daher sind möglichst kurze Merkmalslisten zu bevorzugen.

Die transformierten Grammatiken konnten nun mit dem verwendeten LC-Parser [7] ohne die oben genannten Probleme geparst werden. Abbildung 3 zeigt die Dauer der Parsierung mit dem LC-Parser für einige Zahlwortlexika und Zahlwörter nach der Transformation. Es ist festzuhalten, dass alle Parsierungen in wenigen (Milli-)Sekunden erfolgreich beendet sind. Selbst

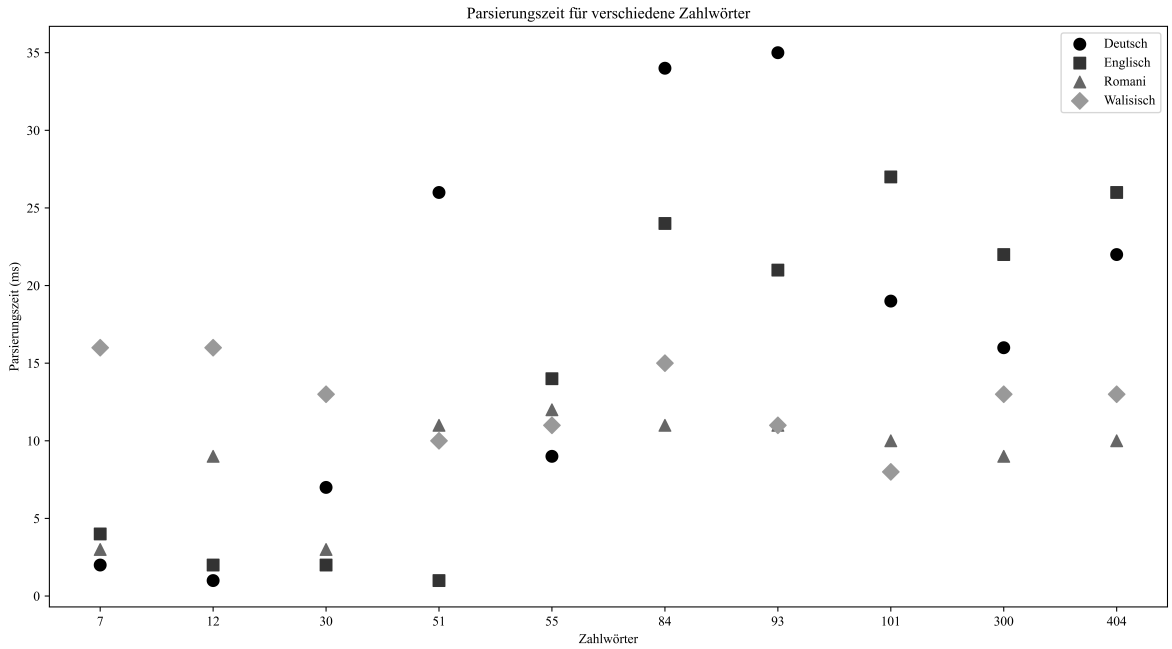

Abbildung 3 – Parsierungszeiten für 4 ausgewählte Sprachen. Die Zahlen waren bei allen Test dieselben (7, 12, 30, 51, 55, 84, 93, 101, 300, 404). Der Parser ist mit SWI-Prolog (Version 9.0.4) implementiert. Der verwendete Computer hat einen Intel Core i7-13700 Prozessor mit 2.10 GHz und 31,7 GB verwendbaren RAM, sowie ein 64-Bit-Betriebssystem.

für ein eigens angefertigtes Zahlwortlexikon für deutsche Zahlwörter mit dem Wertebereich 1 bis 1.000.000 braucht für komplexe Zahlen (d. h. viele involvierte Einträge) nur wenige Sekunden (z. B. für *sechzig, tausend* dauert das Parsen 0,01 Sekunden, während das Parsen für *vier, und, fünf, zig, tausend, drei, hundert, sieben* insgesamt 3,664 Sekunden braucht).

Für die 36 Lexika, welche die Voraussetzung 1 nicht erfüllt haben, wurde der Algorithmus mit der beschriebenen Abänderung angewandt. Die mittlere Anzahl an Einträge Betrug vor der Transformation 80 und stieg im Mittel um 221% durch die Transformation. Der Anstieg ist also vergleichbar mit dem der anderen Lexika. Der Anteil von ε-LI sank im Mittel von 45% auf 4%. Im Mittel haben die neuen Lexika hier 8 ε-LI. Als Beispiel wurde hier Französisch genauer betrachtet (siehe Tabelle 1), mit 27 verbleibenden ε-LI, und unter den gleichen Bedingen wie in Abbildung 3 die Dauer des Parsen gemessen.

Tabelle 1 – Parsierungszeit in *ms* für das französische Zahlwortlexikon unter den gleichen Bedingungen wie in Abbildung 3

Zahlwörter	7	12	30	51	55	84	93	97	101	300	404
Parsierungszeit (ms)	88	62	40	141	168	455	218	10.486	453	160	12.420

Zu erkennen ist die deutlich längeren Parsierungsdauer verglichen mit den Lexika, die nicht Voraussetzung 1 für den Algorithmus verletzten.

5 Ausblick

Der in dieser Arbeit vorgestellte Algorithmus stellt einen ersten Schritt dar, um ein praxistaugliches Parsing mit MGs zu ermöglichen. Dieser Algorithmus zeigt eine Möglichkeit, MGs für die Benutzung durch Parser mittels geeigneter Transformation effizienter in Bezug auf die Laufzeit zu gestalten. Ein Blick auf die Ergebnisse der französischen Zahlwortgrammatik zeigt noch

deutliche Verbesserungsmöglichkeiten für die Verarbeitungsdauer. Es ist zu vermuten, dass die verbleibenden ε-LI hier zu einer längeren Verarbeitungszeit beim Parsen geführt haben könnten, als bei vergleichbaren Lexika ohne ε-LI. Eine Verbesserung des Algorithmus, der auch für derartige Lexika die ε-Einträge weiter herausarbeitet sollte zu Verbesserungen führen. Hier herrscht also noch Forschungsbedarf. Die hier generell erzeugten Lexika sind noch nicht optimal in Bezug auf die Größe und die Parsierungszeit für längere Ausdrücke. Aber sie sind ein weiterer Schritt zur Optimierung von MGs für die Sprachverarbeitung. Weitere Verfahren zur Transformation von MG-Lexika mit Blick auf eine bessere Verarbeitung beim Parsen und Generieren könnten also ebenfalls noch von Interesse sein. Unseres Wissens nach ist diese Arbeit einer der ersten für Minimalistische Grammatiken zu diesem speziellen Thema, abgesehen von nicht veröffentlichten Studentenarbeiten.

Literatur

[1] RÖMER, R., P. BEIM GRABEN, M. HUBER, M. WOLFF, G. WIRSCHING, und I. SCHMITT: *Behavioral control of cognitive agents using database semantics and minimalist grammars*. S. 73–78. 2019. doi:10.1109/CogInfoCom47531.2019.9089947.

[2] BEIM GRABEN, P., , R. RÖMER, W. MEYER, M. HUBER, und M. WOLFF: *Reinforcement learning of minimalist numeral grammars*. In *2019 10th IEEE International Conference on Cognitive Infocommunications (CogInfoCom)*, S. 67–72. 2019. doi:10.1109/CogInfoCom47531.2019.9089924.

[3] BEIM GRABEN, P., W. MEYER, R. RÖMER, und M. WOLFF: *Bidirektionale utterance-meaning-transducer für zahlworte durch kompositionale minimalistische grammatiken*. In P. BIRKHOLZ und S. STONE (Hrsg.), *Studientexte zur Sprachkommunikation: Elektronische Sprachsignalverarbeitung 2019*, S. 76–82. TUDpress, Dresden, 2019. URL `https://www.researchgate.net/publication/330634207`.

[4] MAIER, I. und M. WOLFF: *A decomposition algorithm for numerals based on arithmetics*. *REUNICE Scientific Poster Contest*, 2022. doi:https://doi.org/10.5281/zenodo.7501280.

[5] MAIER, I., J. KUHN, J. BEISEGEL, M. HUBER-LIEBL, und M. WOLFF: *Minimalist grammar: Construction without overgeneration*. *arXiv*, 2023. doi:https://doi.org/10.48550/arXiv.2311.01820.

[6] MAIER, I.: *Algorithm-to-automatically-generate-minimalist-numeral-grammars*. GitHub, 2023. URL `https://github.com/ikmMaierBTUCS/Algorithm-to-Automatically-Generate-Minimalist-Numeral-Grammars`.

[7] STANOJEVIĆ, M. und E. STABLER: *A sound and complete left-corner parsing for minimalist grammars*. In *Proceedings of the Eight Workshop on Cognitive Aspects of Computational Language Learning and Processing*, S. 65–74. 2018. doi:https://doi.org/10.18653/v1/W18-2809.

[8] DUCKHORN, F., M. WOLFF, und R. HOFFMANN: *A new epsilon filter for efficient composition of weighted finite-state transducers*. In *Interspeech*. 2011. URL `https://api.semanticscholar.org/CorpusID:10305242`.

[9] GRAF, T., A. AKSËNOVA, und A. DE SANTO: *A single movement normal form for minimalist grammars*. In A. FORET, G. MORRILL, R. MUSKENS, R. OSSWALD, und S. POGODALLA (Hrsg.), *Formal Grammar*, S. 200–215. Springer Berlin Heidelberg, Berlin, Heidelberg, 2016.

[10] KOBELE, G.: *Minimalist grammars and decomposition. The Cambridge Handbook of Minimalism. Cambridge University Press, Cambridge, to appear*, 2021. URL https://home.uni-leipzig.de/gkobele/files/unpub/Kobele21MGsAndDecomposition.pdf.

[11] ERMOLAEVA, M.: *Deconstructing syntactic generalizations with minimalist grammars*. In *Proceedings of the 25th Conference on Computational Natural Language Learning*, S. 435–444. 2021.

[12] FUJIYOSHI, A.: *Epsilon-free grammars and lexicalized grammars that generate the class of the mildly context-sensitive languages*. In *Proceedings of the 7th International Workshop on Tree Adjoining Grammar and Related Formalisms*, S. 16–23. 2004.

[13] MAIER, I. K., J. KUHN, J. BEISEGEL, M. HUBER-LIEBL, und M. WOLFF: *Minimalist grammar: Construction without overgeneration*. 2023. doi:https://doi.org/10.48550/arXiv.2311.01820. 2311.01820.