Constraint-Erfüllung und -Optimierung mithilfe von endlichen Maschinen in der Anwendung des robusten Dependenzparsings

Dissertation
zur Erlangung des Doktorgrades
am Fachbereich Informatik der Universität Hamburg

vorgelegt von
Jörg Didakowski
aus Berlin

im Jahr 2022

Vorsitzender der Prüfungskommission
Prof. Dr. Chris Biemann, Universität Hamburg

Vertreter des Fach-Promotionsausschusses
Prof. Dr. Matthias Rarey, Universität Hamburg

Erst- und Zweitgutachter
Prof. Dr.-Ing. Wolfgang Menzel, Universität Hamburg
Prof. Dr. Jonas Kuhn, Universität Stuttgart

Datum der Disputation
23. März 2023

Eidesstattliche Versicherung

Hiermit erkläre ich an Eides statt, dass ich die vorliegende Dissertationsschrift selbst verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

Ich versichere außerdem, dass ich die vorliegende Dissertationsschrift nur in diesem und keinem anderen Promotionsverfahren eingereicht habe und dass diesem Promotionsverfahren keine endgültig gescheiterten Promotionsverfahren vorausgegangen sind.

.....	.....
Ort, Datum	Unterschrift

Danksagung

Ich möchte mich bei den zahlreichen Personen bedanken, die direkt oder indirekt an der Erstellung meiner Arbeit beteiligt waren.

Ich danke meinem Betreuer Prof. Dr.-Ing. Wolfgang Menzel, der mich für den Bereich der Constraint-Verarbeitung begeisterte und mich stets unterstützte. Ich danke ihm dafür, dass er mir die Möglichkeit gab, ein Thema anzugehen, das mir sehr am Herzen lag. Mein Dank gilt aber auch meinem Zweitgutachter Prof. Dr. Jonas Kuhn, der sich spontan auf dieses Thema einließ.

Danken möchte ich ebenfalls Prof. Dr. Peter Staudacher und Dr. Thomas Hanneforth, die mich bei den ersten Schritten in der Computerlinguistik begleiteten und die mich neben den logischen Systemen auch für die endlichen Techniken begeisterten.

Großer Dank geht zudem an meine Freundin Claudia Willig, die mich dabei unterstützte, die letzten beschwerlichen Meter meiner Promotion zu gehen, an Jette Klein-Berning, die mir immer mit Rat zur Seite stand, wenn es um linguistische Fragen und Grenzfälle ging, und an meinen Onkel, der mich stets ermutigte weiterzumachen.

Nicht zuletzt danke ich Christine Köhn und Andreas Fritsch für ihr wertvolles Feedback in unseren Doktorarbeitsbesprechungen, meinen Freunden Nadja Radtke, Friederike Bachmann, Susanne Haaf und Angela Hartwig, die mich tatkräftig beim orthografischen Feinschliff meiner Arbeit unterstützten, und Claudia Wolf, die mir am finalen Tag den Doktorhut überreichte.

Abstract

This thesis follows the working hypothesis that the field of constraint processing and that of finite-state techniques have a strong potential to enrich each other with new perspectives and possibilities and that their combination provides a true added value. This is especially true for dependency-based parsing, where a lot of research exists on both sides individually.

The semiring-based constraint satisfaction problem is used to combine these two fields. It is shown that it can be represented and solved in a natural way using finite-state machines. Furthermore, it is extended by three essential aspects: an infinite domain, a simultaneous representation and global constraints. Hence, a unified formalism for finite techniques is available that incorporates soft constraints and propagation techniques. For constraint processing, on the other hand, new possibilities exist to tackle known or recent problems.

Based on this, robust dependency parsing in the style of the Weighted Constraint Dependency Grammar is implemented using an extended finite-state approach: A problem decomposition approach with perfect relaxation is presented, for which different problem decomposition and constraint application strategies are worked out and combined with memoization. It is demonstrated how supra-local constraints can be implemented and how both violations and preferences can be traced during problem solving.

The approach allows the intensive inclusion of non-projective structures and the modelling of complex linguistic phenomena such as structural and coordination ellipses, valences and non-local dependencies. A handwritten, high-coverage grammar for German is presented, which is based on regular predicate logic formulas. On the basis of an evaluation the Grammar achieves a higher parsing quality than that of the Weighted Constraint Dependency Grammar.

Unmanageable intermediate results of enormous size are a fundamental problem in constraint-based parsing with finite-state machines. In contrast, experiments show that a compact representation of intermediate results during problem solving is possible. This is enabled by the problem decomposition and constraint application strategies developed here.

The approach provides many connecting points and wide scope for the inclusion of data-driven and neural methods and complexity-reducing approximation strategies. Thus, a powerful hybrid neurosymbolic parsing system is close at hand.

Zusammenfassung

Die Arbeit folgt der These, dass der Bereich der Constraint-Verarbeitung und der Bereich der endlichen Techniken ein großes Potenzial haben, sich gegenseitig mit neuen Perspektiven und Möglichkeiten zu bereichern und dass deren Kombination einen echten Mehrwert schafft. Dies betrifft insbesondere das dependenzorientierte Parsing, bei dem viele Forschungsarbeiten individuell auf beiden Seiten existieren.

Das Semiring-basierte Constraint-Erfüllungsproblem wird als Bindeglied zwischen den beiden Bereichen herangezogen. Es wird gezeigt, dass es auf natürliche Weise mithilfe von endlichen Maschinen repräsentiert und gelöst werden kann. Zudem wird es um drei wesentliche Aspekte erweitert: um einen unendlichen Wertebereich, um eine simultane Repräsentation und um globale Constraints. Aufseiten der endlichen Techniken steht hierdurch ein einheitlicher Formalismus zur Verfügung, um weiche Beschränkungen und Propagierungstechniken einzubeziehen. Für die Constraint-Verarbeitung existieren hingegen neue Möglichkeiten, bekannte oder neuartige Probleme anzugehen.

Hierauf aufbauend wird robustes Dependenzparsing im Stil der Weighted Constraint Dependency Grammar über einen erweiterten endlichen Ansatz umgesetzt: Es wird ein Problemzerlegungsansatz zur perfekten Relaxierung vorgestellt, für den verschiedene Problemzerlegungs- und Constraint-Anwendungsstrategien erarbeitet und mit Memoisation kombiniert werden. Es wird gezeigt, wie überlokale Constraints umgesetzt und sowohl Verletzungen als auch Präferenzen während des Problemlösens nachverfolgt werden können.

Der Ansatz erlaubt die intensive Einbeziehung nichtprojektiver Strukturen und die Modellierung komplexer linguistischer Phänomene wie Struktur- und Koordinationsellipsen, Valenzen und nichtlokale Abhängigkeiten. Es wird eine handgeschriebene, abdeckungsreiche Grammatik für das Deutsche vorgestellt, die auf regulären prädikatenlogischen Formeln beruht und anhand einer Auswertung eine höhere Parsing-Qualität ermöglicht als die der Weighted Constraint Dependency Grammar.

Unhandhabbare Zwischenergebnisse von enormer Größe sind beim Constraint-basierten Parsing mit endlichen Maschinen ein fundamentales Problem. Über Experimente wird dahingegen gezeigt, dass über die hier entwickelten Problemzerlegungs- und Constraint-Anwendungsstrategien zusammen mit der Memoisation eine kompakte Repräsentation von Zwischenergebnissen während des Problemlösens erreicht werden kann.

Der Ansatz bietet viele Anknüpfungspunkte und viel Spielraum für die Einbeziehung datengetriebener und neuronaler Methoden und komplexitätsreduzierender Approximationsstrategien. In diesem Rahmen ist ein leistungsstarkes hybrides neurosymbolisches Parsing-System somit in greifbarer Nähe.

Inhaltsverzeichnis

Abstract **i**

Zusammenfassung **ii**

Abkürzungsverzeichnis **vii**

1 Einführung **1**
1.1 Gegenstand 1
1.2 Ziele dieser Arbeit 3
1.3 Veröffentlichungen 5
1.4 Überblick 6

2 Formale Grundlagen und Notationen **8**
2.1 c-Semiring 8
2.2 Endliche Maschinen 12
2.2.1 Gewichtete endliche Transduktoren 14
2.2.2 Kompilierung 24
2.3 Constraint-Erfüllung und -Optimierung 34
2.3.1 Constraint-Erfüllungsproblem 35
2.3.2 Weiche Constraints 36
2.3.3 Semiring-basiertes Constraint-Erfüllungsproblem . . . 37
2.4 Dependenzrelationen und -strukturen 48

3 Constraint-Erfüllung und -Optimierung mithilfe von FSMs **50**
3.1 Verwandte Arbeiten 50
3.1.1 CSP mit endlichem Wertebereich 51
3.1.2 CSP mit unendlichem Wertebereich 54
3.1.3 Globale Constraints 56
3.2 Repräsentation eines SCSPs mithilfe von FSMs 60
3.2.1 Constraint-System 60
3.2.2 Constraints 68

3.2.3 Constraint-Problem 73
3.3 Lösen eines SCSPs mithilfe von FSMs 74
3.3.1 Tupel-Projektion 74
3.3.2 Kombination 75
3.3.3 Projektion 77
3.3.4 Lösung 78
3.3.5 Beste Konsistenzstufe und abstrakte Lösung 82
3.3.6 Constraint-Propagierung 83
3.4 Erweiterung des SCSP-Frameworks mithilfe von FSMs 84
3.4.1 Unendlicher Wertebereich 84
3.4.2 Simultane Repräsentation 87
3.4.3 Quantifizierte reguläre Constraints 101

4 Robustes Dependenzparsing als SCSP mithilfe von FSMs 113
4.1 Verwandte Arbeiten 114
4.1.1 Parsing mithilfe von FSMs 114
4.1.2 Parsing als CSP und COP 123
4.2 Naive WDG-Umsetzung 129
4.2.1 Constraint-System 129
4.2.2 Constraints 135
4.2.3 Quantifizierte reguläre Constraints 136
4.2.4 Innewohnende Schwierigkeiten 140
4.3 Problemzerlegung und perfekte Relaxierung 141
4.3.1 Lokalität in der Grammatik 141
4.3.2 Syntaxnahe Zerlegung 142
4.3.3 Zerlegung in Ebenen 147
4.3.4 Perfekte Relaxierung 156
4.4 Perfekte Relaxierung mithilfe von FSMs 159
4.4.1 Repräsentation lokaler Konsistenzregeln 159
4.4.2 Anwendung lokaler Konsistenzregeln 161
4.4.3 Zusammenfassen lokaler Konsistenzregeln 162
4.4.4 Bottom-Up-Lösen 162
4.5 Simultane perfekte Relaxierung mithilfe von FSMs 163
4.5.1 Repräsentation simultaner lokaler Konsistenzregeln . . 164
4.5.2 Zusammenfassen simultaner lokaler Konsistenzregeln . 171
4.5.3 Parsingkomplexität 172
4.5.4 Ambige Eingabe 173
4.5.5 Bottom-Up-Lösen 174
4.5.6 Bottom-Up-Lösen mit Optimalzerlegung 176
4.5.7 Memoisation von Zwischenergebnissen 180
4.5.8 Aufspalten simultaner lokaler Konsistenzregeln 184

4.5.9 Verzögertes Anwenden von Constraints 187
4.5.10 Affixorientierte Zerlegung 195
4.5.11 Weitere Aufspaltung von Teilproblemen 201
4.5.12 Überlokale Constraints 205
4.5.13 Dependenzstruktur und Verletzungsnachverfolgung . . 212

5 Praktische Umsetzung 215
5.1 Endliche-Maschine-Bibliothek 215
5.2 Lexikalische Ressource 216
5.3 Constraint-System 217
5.3.1 Wertebereich 217
5.3.2 Gewichtsstruktur 219
5.3.3 Verknüpfte Informationen 220
5.3.4 Virtuelle Knoten 221
5.4 Grundlegendes Grammatikdesign 222
5.5 Grammatikformat 224
5.5.1 Prädikate 225
5.5.2 Harte und weiche Constraints 225
5.5.3 Hilfsconstraints 228
5.5.4 Dynamische Merkmale 229
5.5.5 Weiter aufgespaltene Teilprobleme 231
5.5.6 Teilausdrücke 232
5.6 Modellierung sprachlicher Phänomene 232
5.6.1 Satzzeichen 233
5.6.2 Appositionen 234
5.6.3 Nichtprojektivität 235
5.6.4 Nebensatzeinleitende Pronomen und Adverbien 240
5.6.5 Koordination 241
5.6.6 Strukturellipsen 245
5.6.7 Koordinationsellipsen 247
5.7 Performanz der Grammatik 249
5.7.1 Vergleich mit der WCDG 250
5.7.2 Betrachtung der einzelnen Dependenz-Label 252

6 Experimente 254
6.1 Problemzerlegung 255
6.1.1 Satzlänge und Zerlegungstiefe 255
6.1.2 Satzlänge und Zustandsanzahl 256
6.1.3 Zerlegungsebenen und Zustandsanzahl 258
6.2 Verzögerte Anwendung von harten Constraints 262
6.2.1 Satzlänge und Zerlegungstiefe 263

6.2.2 Satzlänge und Zustandsanzahl 263
6.2.3 Zerlegungsebenen und Zustandsanzahl 265
6.3 Verzögerte Anwendung von weichen Constraints 267
6.3.1 Satzlänge und Zustandsanzahl 268
6.3.2 Zerlegungsebenen und Zustandsanzahl 268
6.4 Praktische Laufzeit 269
6.4.1 Laufzeit und Satzlänge 270
6.5 Zusammenfassende Ergebnisse 271
6.5.1 Konsequenzen und Möglichkeiten 274

7 Schlussbetrachtung **280**
7.1 Zusammenfassung 281
7.2 Gegenseitige Bereicherung 282
7.2.1 Constraint-Verarbeitung 282
7.2.2 Endliche Techniken 284
7.3 Einordnung 286
7.4 Ausblick 290

Abkürzungsverzeichnis

CDG Constraint Dependency Grammar

CG Constraint Grammar

CSP Constraint-Erfüllungsproblem (Constraint Satisfaction Problem)

DWDS Digitales Wörterbuch der deutschen Sprache

FSA endlicher Automat (Finite-State Automaton)

FSIG Finite-State Intersection Grammar

FST endlicher Transduktor (Finite-State Transducer)

FSM endliche Maschine (Finite-State Machine)

LAS Labeled Attachment Score

MUSE-CSP MUltiply SEgmented Constraint Satisfaction Problem

SCSP Semiring-basiertes Constraint-Erfüllungsproblem (Semiring-based Constraint Satisfaction Problem)

UAS Unlabeled Attachment Score

VCSP bewertetes Constraint-Erfüllungsproblem (Valued Constraint Satisfaction Problem)

WCDG Weighted Constraint Dependency Grammar

WFSA gewichteter endlicher Automat (Weighted Finite-State Automaton)

WFST gewichteter endlicher Transduktor (Weighted Finite-State Transducer)

XDG Extensible Dependency Grammar

Kapitel 1

Einführung

1.1 Gegenstand

Die vorliegende Arbeit folgt der These, dass der Bereich der Constraint-Verarbeitung und der Bereich der endlichen Techniken ein sehr großes Potenzial haben, sich gegenseitig mit neuen Perspektiven und Möglichkeiten zu bereichern und dass deren Kombination einen echten Mehrwert schafft. Dies betrifft insbesondere das dependenzorientierte Parsing, bei dem viele Forschungsarbeiten auf beiden Seiten existieren. Es werden beide Welten zusammengebracht, um von deren jeweiligen Vorteilen zu lernen und zu profitieren. Dies geschieht aufseiten der Constraint-Verarbeitung über das *Constraint-Erfüllungsproblem* (Constraint Satisfaction Problem, CSP, siehe z. B. Dechter, 2003, Kapitel 2) und das *Constraint-Optimierungsproblem* (Constraint Optimization Problem, COP, siehe z. B. Dechter, 2003, Kapitel 13) und aufseiten der endlichen Techniken über die *endlichen Maschinen* (Finite-State Machine, FSM, siehe z. B. Mohri, 1997).

Das CSP ist ein einfaches und mächtiges Werkzeug, das es ermöglicht, Probleme über Constraints und Variablen darzustellen und zu lösen. Für die Beschreibung mancher realitätsnaher Situationen reicht das klassische CSP jedoch nicht aus. Dies sind solche, die Präferenzen, verletzbare Beschränkungen und Unsicherheiten beinhalten. Um diese zu beschreiben, können weiche Constraints herangezogen werden. Dann wird von einem Constraint-Optimierungsproblem gesprochen, bei dem bestimmte Kombinationen von Variablenwerten als besser eingestuft werden als andere. Je nach Problembeschreibung können verschiedene Klassen von Constraints angewendet werden, die mit der Einstufung der Variablenwertkombinationen unterschiedlich umgehen. Über diese können Kosten, Präferenzen, Wahrscheinlichkeiten oder andere Maße behandelt werden. Um sie umzusetzen, existieren

Frameworks wie das *Semiring-basierte Constraint-Erfüllungsproblem* (Semiring-based Constraint Satisfaction Problem, SCSP, Bistarelli et al., 1995).

FSMs stellen ein relativ einfaches Berechnungsmodell auf Zeichenkettenebene dar und bestehen aus Zuständen und Übergängen. Über rationale Operationen ist es möglich, mit ihnen zu rechnen und neue Maschinen zu erstellen, die spezielle Aufgaben bewältigen oder bestimmte Informationen repräsentieren. Die gewichteten Varianten sind zusätzlich mit einer Gewichtsstruktur in Form eines Semirings ausgestattet, über die Kosten, Präferenzen, Wahrscheinlichkeiten oder andere Maße abgebildet werden können. In der natürlichen Sprachverarbeitung werden diese Maschinen für verschiedenste Aufgaben eingesetzt. Die Spanne reicht von der Spracherkennung bis hin zur phonologischen, morphologischen und syntaktischen Analyse.

Der Fokus dieser Arbeit liegt auf der Repräsentation und Lösung des Constraint-Erfüllungs- und Constraint-Optimierungsproblems mithilfe von FSMs. Insbesondere die Umsetzung der Constraint-Optimierung über FSMs wurde in der bisherigen Forschung nur rudimentär angegangen. Als konkreter Anwendungsfall wird die anspruchsvolle realitätsnahe Situation des robusten Dependenzparsings natürlicher Sprache herangezogen. Sie ist sehr komplex und umfangreich und beinhaltet drei wesentliche Herausforderungen:

- Regeln, die nicht hart sind und verletzt werden können und eventuell eine Verletzungsschwere mit sich ziehen
- Präferenzen, die negativ oder positiv sein können
- Unvollständiges, ungenaues oder fehlerbehaftetes Wissen

Ein gewichteter Constraint-basierter Ansatz ist geeignet, diese Herausforderungen anzugehen und liefert darüber hinaus Transparenz und Erklärbarkeit und bietet dadurch einen hohen Grad an Vertrauenswürdigkeit.

Der Weg zu einem leistungsstarken Dependenzparser, der auf FSMs beruht und Verletzbarkeit, Präferenzen und Unsicherheit einbezieht, ist lang. So konzentriert sich die vorliegende Arbeit zunächst auf das exakte Ausrechnen eines komplexen Problems, das Handling von Zwischenergebnissen und die Formulierung von Beschränkungen. Sie kann als Vorarbeit für ein hybrides neurosymbolisches System gesehen werden, das Approximationsstrategien und datengetriebene Methoden einbezieht. Hybriden Systeme sind beispielsweise in Foth (2006b) und Sennrich et al. (2009) zu finden, wo deren Vorzüge unterstrichen werden.

1.2 Ziele dieser Arbeit

Umsetzung des SCSPs mithilfe von FSMs Vempaty (1992) zeigte erstmals, wie ein klassisches CSP mit azyklischen FSMs repräsentiert und gelöst werden kann. Dort wird ein endlicher Wertebereich vorausgesetzt und der Skopus eines Constraints umfasst stets alle Variablen, was die Anwendung von Constraint-Propagierungstechniken ausschließt. Es handelt sich um die einzige Arbeit, die sich mit der generellen Repräsentation und Lösung des CSPs mithilfe von FSMs beschäftigt. Vergleichbare Forschungsarbeiten im Kontext des COPs sind nicht zu finden und es existieren auch keine, die endliche Techniken mit einem Framework wie dem SCSP zusammenbringen. Das ist recht verwunderlich, da die Ergebnisse von Vempaty (1992) relativ lange zurückliegen. So ist ein Ziel der vorliegenden Arbeit zu zeigen, dass sich das SCSP mithilfe von FSMs auf natürliche Weise repräsentieren und lösen lässt und dass über diese Repräsentationsform Propagierungstechniken einbezogen werden können.

Erweiterung des SCSPs mithilfe von FSMs Das SCSP setzt einen endlichen Wertebereich voraus und es existieren keine Arbeiten, die dieses um einen unendlichen ergänzen. In Helzerman und Harper (1996) wird eine Erweiterung des CSPs vorgestellt, die eine simultane Repräsentation und Lösung mehrerer Probleme ermöglicht. Dieser Ansatz ist jedoch auf binäre Instanzen beschränkt und wurde noch nicht auf das SCSP verallgemeinert. Globale Constraints wie z. B. der *Regular-Membership-Constraint* in Pesant (2004) sind für jeden Skopus wohldefiniert. Solche Constraints sind innerhalb des SCSPs jedoch nicht nativ umsetzbar, da dort eine feste Stelligkeit vorausgesetzt wird. Ein weiteres Ziel dieser Arbeit ist es daher zu zeigen, dass die Einbeziehung von endlichen Techniken eine Erweiterung des SCSP-Frameworks auf die oben genannten Punkte ermöglicht und dass FSMs im Rahmen der Constraint-Verarbeitung somit einen echten Mehrwert schafft.

Robustes Dependenzparsing als Anwendungsfall Ausgehend von der Umsetzung und Erweiterung des SCSPs mithilfe von FSMs, soll das Dependenzparsing natürlicher Sprache als Anwendungsfall betrachtet werden. Die *Weighted Constraint Dependency Grammar* (WCDG, siehe Foth et al., 2004) ist ein Constraint-basiertes Framework, das sich dieser anspruchsvollen, realitätsnahen Situation im Rahmen eines binären COPs widmet. Jedoch wird die Berechnung einer nach der Grammatik optimalen Lösung von dem Framework nicht gewährleistet, da eine vollständige Suche zu aufwendig und unpraktikabel wäre. Damit geht das Potenzial völlig erklärbarer und nach-

vollziehbarer Ergebnisse verloren. Zudem wird dort nur ein konkretes Parsingergebnis ermittelt, was den Erhalt von Ambiguitäten für nachgeschaltete Komponenten ausschließt. Ein weiteres Ziel dieser Arbeit ist zu zeigen, dass ein äußerst komplexes, realitätsnahes Problem wie das regelbasierte Dependenzparsing analog zur WCDG im Rahmen des SCSP-Frameworks innerhalb eines erweiterten endlichen Ansatzes umgesetzt werden kann. Über die Propagierungstechnik der perfekten Relaxierung und über Memoisation, einer Technik des dynamischen Programmierens, sollen alle nach der Grammatik optimalen Analysen auch in der Praxis gewährleistet werden. Aufseiten der FSMs ist dies ein ambitioniertes Vorhaben, da dort die Einbeziehung von Memoisations- und Constraint-Propagierungstechniken und die Verwendung von Gewichtungen für die Modellierung von Präferenzen, Verletzungen und Unsicherheiten teilweise Neuland bedeuten und unhandhabbare Zwischenergebnisse ein fundamentales Problem darstellen.

Formulierung und Umsetzung von Beschränkungen Innerhalb der WCDG werden Regeln über universalquantifizierte Formeln ausgedrückt, über die nativ höchstens zwei Dependenzkanten einbezogen werden können. Wenn sich mehrere Beschränkungen auf ein spezielles sprachliches Phänomen beziehen, muss innerhalb dieser stets erneut auf dieses geprüft werden. Es können zwar Hilfsconstraints angewendet werden, diese sind jedoch unär. Somit können Constraint-Checks nur bedingt gekapselt und wiederverwendet werden und es ist nur eingeschränkt möglich, den Sichtbereich einer Regel zu erweitern. Für die Behandlung von Ellipsen wie z. B. Verbauslassungen werden zusätzliche Dependenz-Label herangezogen, mit denen jedoch nicht alle elliptischen Konstruktionen auf natürliche Weise abgebildet werden können. Ein weiteres Ziel dieser Arbeit ist es daher, zu zeigen, dass über den neuen Ansatz eine strukturiertere Beschreibung sprachlicher Phänomene möglich ist: indem innerhalb einer Beschränkung nativ mehr als zwei Kanten einbezogen und existenzielle Quantifizierungen angewendet werden können; indem mehrstellige Hilfsconstraints und dynamische Merkmale möglich sind, um den Sichtbereich einer Regel zu erweitern; indem leere Köpfe angenommen werden können, um Ellipsen zu modellieren. Wenn ein System komplett auf FSMs beruht, muss ein Grammatikschreiber im Rahmen von regulären Ausdrücken eine bestimmte Zeichenkettencodierung der Eingabe mitdenken. Hier soll hingegen gezeigt werden, dass innerhalb der Regeln über *reguläre prädikatenlogische Formeln* (Hulden, 2008) eine volle Abstraktion von der Zeichenkettenebene möglich ist und dass im Rahmen des neuen Ansatzes analog zur WCDG eine abdeckungsreiche Grammatik für das Deutsche umgesetzt werden kann.

1.3 Veröffentlichungen

Grundlegende theoretische Ergebnisse dieser Arbeit sind in dem Festschriftartikel Didakowski (2010) zu finden. Dieser thematisiert erstmals die Umsetzung des SCSPs mit FSMs und betrachtet das Dependenzparsing als Anwendungsfall.

Des Weiteren existieren zwei Beiträge, die sich mit dem regelbasierten dependenzorientierten Parsing beschäftigen, sich aber nicht im Kontext des Constraint-Erfüllungs- und -Optimierungsproblems bewegen. Nichtsdestotrotz liefern sie grundlegende Bausteine für den hier erarbeiteten Ansatz. Zudem betonen deren Folgeveröffentlichungen, die neben dieser Arbeit entstanden, die Relevanz des Parsingthemas für praktische Anwendungen. Die einzelnen Veröffentlichungen sind im Folgenden aufgeführt.

Der Konferenzartikel Didakowski (2008a), der in Vorarbeit zu dieser Arbeit entstand, befasst sich mit der flachen Analyse von Sätzen mithilfe von gewichteten endlichen Transduktoren, einer speziellen Art der FSMs. Es wird ein Verfahren vorgestellt, das eine Mischung aus einem konstruktiven und einem eliminativen Ansatz ist. Es ist der erste Ansatz, der die *Finite-State Intersection Grammar* (FSIG, siehe Abschnitt 4.1.1) um Gewichtungen erweitert. Teile des dort vorgestellten Ansatzes finden sich auch in Beiträgen zur regelbasierten Eigennamenerkennung und -klassifizierung. Siehe hierzu die Veröffentlichungen Didakowski et al. (2006), Didakowski et al. (2007) und Didakowski und Drotschmann (2008). Die Eigennamenerkennung findet unter anderem praktische Anwendung in der Informationsanreicherung von Texten im Projekt *Deutsches Textarchiv* (DTA, `http://www.deutschestextarchiv.de`).

Der Konferenzartikel Didakowski (2008b), der im Rahmen dieser Arbeit erstellt wurde, befasst sich mit einem erweiterten endlichen Ansatz zum regelbasierten projektiven Dependenzparsing, der mithilfe von gewichteten endlichen Transduktoren umgesetzt ist und mit Gewichtungen, einem Fixed-Point-Algorithmus und Memoisation arbeitet. Dieser Ansatz kommt in zwei praktischen Anwendungen im Bereich der Korpuslinguistik zum Einsatz. Dies ist einerseits die Relationsextraktion im Rahmen der vollautomatischen Erstellung einer Kookkurrenzdatenbank, die auf der Projektseite des *Digitalen Wörterbuchs der deutschen Sprache* (DWDS, `http://www.dwds.de`, Geyken et al., 2017) unter dem Namen *DWDS-Wortprofil* abfragbar und insbesondere für Deutsch als Fremdsprache interessant ist. Siehe hierzu die Beiträge Geyken et al. (2008), Geyken et al. (2009) und Didakowski und Geyken (2013). Das DWDS-Wortprofil hat sich als ein sehr nützliches Werkzeug für die maschinell gestützte Ermittlung von Stützverbgefügen erwiesen, siehe hierzu die Veröffentlichung Didakowski und Radtke (2014) und die Veröffentlichungen

Didakowski und Radtke (2018) mit dem Schwerpunkt Textsorten und Didakowski und Radtke (2020) mit dem Fokus auf Adjektive. Andererseits findet der Parsingansatz in der Bewertung der Komplexität und Lesbarkeit von Sätzen im Rahmen der vollautomatischen Extraktion guter Beispielsätze Anwendung. Siehe hierzu den Beitrag Didakowski et al. (2012) und den Beitrag Lemnitzer et al. (2015) mit Einbeziehung von maschinellem Lernen. Die extrahierten Beispielsätze stellen Belegkandidaten für einen Wörterbuchartikel dar und beschleunigen die Recherchearbeit der Lexikografen im DWDS-Projekt. Zudem werden die Sätze auf der DWDS-Projektseite präsentiert, falls zu einem Stichwort noch kein ausgearbeiteter Artikel zur Verfügung steht.

1.4 Überblick

Die Arbeit ist wie folgt strukturiert:

- In Kapitel 1 wurde die Arbeitsthese formuliert und es wurde ein Einstieg in den Gegenstand gegeben. Anschließend wurden die Ziele dieser Arbeit definiert.
- In Kapitel 2 werden die formalen Grundlagen und Notationen gegeben, die für das Verständnis dieser Arbeit notwendig sind.
- In Kapitel 3 wird gezeigt, wie Constraint-Erfüllung und -Optimierung mithilfe von endlichen Techniken umgesetzt werden kann. Es werden relevante Arbeiten zur Constraint-Verarbeitung mit endlichen Techniken vorgestellt. Darauf aufbauend wird die Umsetzung und Erweiterung des SCSP-Frameworks über FSMs erarbeitet.
- In Kapitel 4 wird ein neuer Parsingansatz vorgestellt, der in dem SCSP-Framework eingebettet ist und auf FSMs beruht. Es wird auf verwandte Arbeiten eingegangen, die für das dependenzorientierte Parsing mithilfe von endlichen Techniken und mithilfe der Constraint-Verarbeitung relevant sind. Darauf aufbauend wird der neue Ansatz erarbeitet.
- In Kapitel 5 wird eine Implementierung des neuen Ansatzes vorgestellt. Es wird auf die Codierung des Parsing-Problems als SCSP eingegangen und es wird eine umfangreiche Grammatik für das Deutsche vorgestellt. Deren Qualität wird abschließend evaluiert.
- In Kapitel 6 wird über Experimente untersucht, wie sich verschiedene Problemlösungsstrategien und die Memoisation von FSMs beim Parsing auf die Größe der Zwischenergebnisse auswirken. Daraufhin wird die praktische Laufzeit untersucht und diskutiert.

- Abschließend gibt Kapitel 7 eine Zusammenfassung der Ergebnisse, eine Einordnung in die aktuelle Forschung und einen Ausblick für weiterführende Forschungsfragen.

Kapitel 2

Formale Grundlagen und Notationen

2.1 c-Semiring

Bekannte Strukturen aus der abstrakten Algebra sind Gruppen, Ringe und Körper. Diese beschreiben Operationen auf Mengen und sind axiomatisch und auf abstrakte Weise über bestimmte Eigenschaften definiert. In dieser Arbeit ist die algebraische Struktur des Semirings von zentraler Bedeutung. Dabei handelt es sich um einen (unitären) Ring ohne Subtraktion. Bekannt ist der Semiring-Begriff z. B. aus dem algebraischen Pfadproblem, bei dem Graphen über solch eine Struktur bewertet sind (siehe Hebisch und Weinert, 1998, Abschnitt 4.6). Im Folgenden wird der Begriff des Semirings genauer definiert. Daraufhin wird auf die algebraische Struktur des c-Semirings eingegangen, die eine Semiring-Struktur mit erweiterten Eigenschaften darstellt und für die folgenden Kapitel grundlegend ist.

Ein Semiring ist eine algebraische Struktur, die zwei Operationen auf einer Menge beschreibt, die Addition $\oplus$ und die Multiplikation $\otimes$. Mit diesen beiden sind auf abstrakte Weise bestimmte Eigenschaften verbunden, daher wird auch von der *abstrakten Addition* und der *abstrakten Multiplikation* gesprochen. Der Semiring-Begriff wurde erstmals von Vandiver (1934, Abschnitt 6) eingeführt. Dort und in anderer Literatur wird die Semiring-Struktur über die algebraische Struktur des Monoiden definiert. Dann besteht ein Semiring aus zwei Monoiden, welche bestimmte Eigenschaften (untereinander) erfüllen (siehe z. B. Hebisch und Weinert, 1998, Abschnitt 1.2). Hier wird ein Semiring nicht über diesen Umweg definiert. Die Definition orientiert sich an Bistarelli

et al. (1997, Abschnitt 2):

Definition 1 (Semiring)**.** *Ein Semiring ist ein Quintupel* $(A, \oplus, \otimes, \bar{0}, \bar{1})$*, das (i)-(iv) erfüllt:*

(i) A *ist eine Menge und* $\bar{0}, \bar{1} \in A$*, wobei* $\bar{0}$ *als Nullelement und* $\bar{1}$ *als Einselement des Semirings bezeichnet werden.*

(ii) $\oplus$ *ist eine kommutative (d.h.* $a \oplus b = b \oplus a$ *für alle* $a, b \in A$*), assoziative (d.h.* $(a \oplus b) \oplus c = b \oplus (a \oplus c)$ *für alle* $a, b, c \in A$*) Operation mit* $\bar{0}$ *als neutralem Element (d.h.* $a \oplus \bar{0} = a = \bar{0} \oplus a$ *für alle* $a, b \in A$*).*

(iii) $\otimes$ *ist eine assoziative (d.h.* $(a \otimes b) \otimes c = b \otimes (a \otimes c)$ *für alle* $a, b, c \in A$*) Operation mit* $\bar{1}$ *als neutralem Element (d.h.* $a \otimes \bar{1} = a = \bar{1} \otimes a$ *für alle* $a, b \in A$*) und mit* $\bar{0}$ *als absorbierendem Element (d.h.* $a \otimes \bar{0} = \bar{0} = \bar{0} \otimes a$ *für alle* $a, b \in A$*).*

(iv) $\otimes$ *distribuiert über* $\oplus$ *(d.h.* $a \otimes (b \oplus c) = (a \otimes b) \oplus (a \otimes c)$ *und* $(b \oplus c) \otimes a = (b \otimes a) \oplus (c \otimes a)$ *für alles* $a, b, c \in A$*).*

Über eine Semiring-Struktur können Meta-Frameworks definiert werden, bei denen die Verwendungen verschiedener Semiringe zu unterschiedlichen Instanzen führen. Ein Beispiel hierfür sind gewichtete endliche Transduktoren (Weighted Finite-State Transducer, WFST) oder das Semiring-basierte Constraint-Erfüllungsproblem (Semiring-based Constraint Satisfaction Problem, SCSP). Innerhalb dieser beiden Frameworks wird die Semiring-Algebra beispielsweise dafür verwendet, abhängig vom Semiring mit Präferenzen, Verletzungen oder Wahrscheinlichkeiten umzugehen (siehe Abschnitt 2.2 und 2.3).

Jedoch sind die grundlegenden Eigenschaften eines Semirings nicht immer restriktiv genug. Ein SCSP fordert z. B., dass die abstrakte Addition eine partielle Ordnung definiert und dass mit unendlichen Summen umgegangen werden muss. Aus diesem Grund ist in Bistarelli et al. (1995, Abschnitt 2) ein Semiring mit erweiterten Eigenschaften definiert, der als c-Semiring bezeichnet wird. Hierbei steht das 'c' für Constraint im Sinne eines SCSPs. In der folgenden Definition des c-Semirings wird für die Addition von Zweielementmengen $\oplus$ in Infixnotation und für den generellen Fall $\bigoplus$ in Präfixnotation geschrieben:

Definition 2 (c-Semiring)**.** *Ein c-Semiring ist ein Quintupel* $(A, \oplus, \otimes, \bar{0}, \bar{1})$*, das (i)-(iv) erfüllt:*

(i) A *ist eine Menge und* $\bar{0}, \bar{1} \in A$*.*

(ii) $\oplus$ *ist über Mengen von Elementen aus* A *definiert:*

- *für alle* $a \in A, \bigoplus(\{a\}) = a$*.*
- $\bigoplus(\varnothing) = \bar{0}$ *und* $\bigoplus(A) = \bar{1}$*;*
- $\bigoplus(\bigcup A_i, i \in I) = \bigoplus(\bigoplus(A_i), i \in I)$ *für alle Mengen von Indizes* I*, die die Familien der Teilmengen von* A *indizieren.*

(iii) $\otimes$ *ist eine kommutative (d.h.* $a \otimes b = b \otimes a$ *für alle* $a, b \in A$*) und assoziative Operation mit* $\bar{1}$ *als neutralem Element und mit* $\bar{0}$ *als absorbierendem Element.*

(iv) $\otimes$ *distribuiert über* $\oplus$ *(d.h. für alle* $a \in A$ *und* $B \subseteq A$ *ist* $a \otimes \bigoplus(B) = \bigoplus(\{a \otimes b, b \in B\})$ *und* $\bigoplus(B) \otimes a = \bigoplus(\{b \otimes a, b \in B\})$*).*

Die c-Semiring-Struktur erfüllt alle Eigenschaften eines Semirings. $\bar{0}$ ist das neutrale Element der $\oplus$-Operation, da $\bigoplus(\{a, \bar{0}\}) = \bigoplus(\{a\} \cup \varnothing) = \bigoplus(\{a\})$ für jedes $a \in A$. Zudem ist die $\oplus$-Operation über Mengen definiert und nicht über Paare. Dies macht die Operation automatisch kommutativ und assoziativ.

Ein c-Semiring besitzt aber auch erweiterte Eigenschaften. Da die $\oplus$-Operation über Mengen definiert ist, ist sie idempotent, d.h. $a \oplus a = a$ für alle $a \in A$. Somit bildet ein c-Semiring einen idempotenten Semiring. Außerdem ist die $\oplus$-Operation über jede Teilmenge von A definiert, also ebenfalls über unendliche Mengen. Daraus folgt, dass ein c-Semiring bezüglich unendlicher Summen abgeschlossen ist und dass die Distributivität auch für unendliche Summanden gilt. Ein Semiring mit solch einer Eigenschaft wird als vollständiger Semiring bezeichnet. Das Einselement ist absorbierendes Element der $\oplus$-Operation, da $\bigoplus(\{a, \bar{1}\}) = \bigoplus(\{a\} \cup A) = \bigoplus(A) = \bar{1}$ für jedes $a \in A$. Es wird in diesem Fall auch als Top-Element bezeichnet. Da die $\otimes$-Operation kommutativ ist, gilt für jedes $a, b \in A$, dass $a \oplus (a \otimes b) = a \oplus (b \otimes a) = a$. Ein Semiring mit solchen Eigenschaften wird als absorptiv bezeichnet. Hierbei impliziert Absorptivität Idempotenz. Des Weiteren bildet ein c-Semiring einen kommutativen Semiring, da die $\otimes$-Operation kommutativ ist. Zusammenfassend

ist ein c-Semiring ein kommutativer und absorptiver Semiring (vgl. Bistarelli und Gaducci, 2006, Abschnitt 2). Des Weiteren ist ein c-Semiring ein tropischer Semiring mit Top-Element (siehe Bistarelli und Gaducci, 2006, Abschnitt 2.1.1). Dies ist dahingegen interessant, da viele klassische Kürzeste-Pfad-Algorithmen, aber auch einige Algorithmen über gewichtete FSMs im Kontext des tropischen Semirings definiert sind.

Da ein c-Semiring idempotent ist, lässt sich für $S = (A, \oplus, \otimes, \bar{0}, \bar{1})$ eine Ordnung $a \leq_S b$ definieren, mit $a \leq_S b$ genau dann, wenn $a \oplus b = b$ für alle $a, b \in A$. Der Ordnung können hierbei folgende Eigenschaften zugeschrieben werden (vgl. Bistarelli et al., 1997, Abschnitt 2):

- Die Ordnung $\leq_S$ ist eine partielle Ordnung.
- Die Operationen $\oplus$ und $\otimes$ sind monoton bezüglich $\leq_S$ (d.h. $a \leq_S a'$ impliziert, dass $a \oplus b \leq_S a' \oplus b$ und $a \otimes b \leq_S a' \otimes b$ mit $a, a', b \in A$).
- Das Nullelement $\bar{0}$ ist das Minimum von $\leq_S$ (d.h. $\bar{0} \leq_S a$ für alle $a \in A$) und das Einselement $\bar{1}$ ist das Maximum (d.h. $a \leq_S \bar{1}$ für alle $a \in A$), sodass $(A, \leq_S)$ ein vollständiger Verband ist.
- Für alle $a, b \in A$ ist $a \oplus b$ die kleinste obere Schranke bzw. das Supremum (sup) von a und b, sodass $a \oplus b = sup(a, b)$.
- Wenn die Operation $\otimes$ idempotent ist (d.h. $a \otimes a = a$ für alle $a \in A$), ist für alle $a, b \in A$ die größte untere Schranke bzw. das Infimum (inf) von a und b definiert durch $a \otimes b$, sodass $a \otimes b = inf(a, b)$. Dann distribuiert $\oplus$ über $\otimes$ und $(A, \leq_S)$ ist ein distributiver Verband, wobei $\otimes$ dessen Infimum-Operation ist.

Über die partielle Ordnung ist es möglich, Elemente eines c-Semirings miteinander zu vergleichen. Hierbei meint der Vergleich $a \leq_S b$ intuitiv, dass a 'besser' ist als b und dass sich die $\oplus$-Operation zwischen a und b für b entscheidet. Das Supremum einer Teilmenge von A, die über $a \leq_S b$ geordnet werden kann, kann dann als 'bestes' Element der entsprechenden Teilmenge angesehen werden. In Abbildung 2.1 sind die Ordnungseigenschaften eines c-Semirings grafisch zusammengefasst. Dort zeigt der große Kreis die gesamte Menge A mit dem Nullelement als kleinstes und dem Einselement als größtes Element. Der innere Kreis zeigt eine Teilmenge von A mit dem Infimum als kleinstes und dem Supremum als größtes Element.

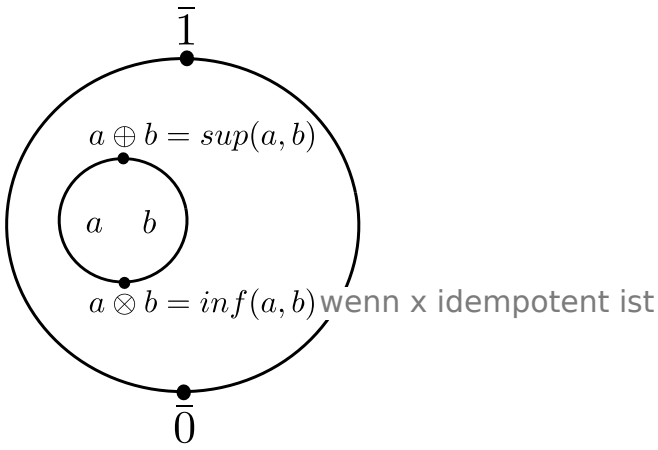


Abbildung 2.1: Ordnungseigenschaften eines c-Semiring.

Von einem c-Semiring wird nicht gefordert, dass über die $\oplus$-Operation eine lineare Ordnung definiert ist. Jedoch sind in Kapitel 4 gerade linear geordnete c-Semiringe relevant. In diesem Fall sind alle Elemente eines c-Semirings miteinander vergleichbar (d.h. für alle $a, b \in A$ gilt $a \leq_S b$ oder $b \leq_S a$). Das einfachste Beispiel für einen linear geordneten c-Semiring ist der Boolesche Semiring $(\{0, 1\}, \vee, \wedge, 0, 1)$, der in der Verbindung von der Theorie der formalen Potenzreihen und der Theorie der formalen Sprachen eine besondere Rolle spielt.

2.2 Endliche Maschinen

Unter dem Framework der *endlichen Maschinen* (Finite-State Machine, FSM) werden in dieser Arbeit folgende Maschinen verstanden: *endliche Automaten* (Finite-State Automaton, FSA), *endliche Transduktoren* (Finite-State Transducer, FST), *gewichtete endliche Automaten* (Weighted Finite-State Automaton, WFSA) und *gewichtete endliche Transduktoren* (Weighted Finite-State Transducer, WFST). Im Folgenden wird auch von ungewichteten und gewichteten FSMs gesprochen. FSMs sind Maschinen auf der Ebene der Zeichenkettenverarbeitung und werden in der natürlichen Sprachverarbeitung für verschiedenste Aufgaben eingesetzt. Die Spanne reicht von der Spracherkennung bis hin zur phonologischen, morphologischen und syntaktischen Analyse (für einen kurzen Überblick siehe z. B. Mohri, 1997). Erstmals beschrieben wurden FSMs im Kontext der mathematischen Formulierung des

Verhaltens von Neuronen in McCulloch und Pitts (1943).

In der Praxis ist ihre Anwendung durch Zeit- und Platzeffizienz motiviert. Ersteres kann durch deterministische Maschinen und Letzteres durch Minimierungsalgorithmen erreicht werden. Ein weiterer und vielleicht noch wichtigerer Grund für ihren praktischen Einsatz ist der Sachverhalt, dass sie sich vielfältig über Automatenoperationen kombinieren lassen. Dies ermöglicht z. B. die modulare und inkrementelle Entwicklung größerer Systeme, bei denen regelbasierte und statistische Paradigmen und große Datenmengen zusammen in einem Formalismus repräsentiert und miteinander verbunden werden können. Die gewichteten FSMs sind zusätzlich mit einer Semiring-Struktur ausgestattet und als eine Generalisierung der ungewichteten FSMs zu verstehen. Sie stellen eine Art Meta-Framework da, bei dem über unterschiedliche Semiringe Wahrscheinlichkeiten, Kosten, Präferenzen oder andere Maße behandelt werden können. Verschiedene Semiringe führen so zu unterschiedlichen Meta-Framework-Instanzen. Ein Beispiel für solch eine Instanz sind probabilistische FSMs, die unter anderem in der Spracherkennung eingesetzt werden (siehe z. B. Mohri et al., 2002). In der folgenden Arbeit sind solche Instanzen im Fokus, die über c-Semiringe gebildet werden können.

Zu den ungewichteten FSMs existieren entsprechende Gegenstücke aus der Theorie der formalen Sprachen. So kann ein FSA eine *rationale Sprache* und ein FST eine *rationale Transduktion*, welche Zeichenketten auf Zeichenketten abbildet, erkennen oder generieren. Zu den gewichteten FSMs existieren hingegen Gegenstücke aus der Theorie der formalen Potenzreihen. So kann ein WFSA eine *rationale Potenzreihe* und ein WFST eine *rationale Transduktion*, welche rationale Potenzreihen auf rationale Potenzreihen abbildet, erkennen oder generieren. Eine rationale Potenzreihe ist hierbei eine Abbildung von der Menge aller Zeichenketten über einem endlichen Alphabet in die Trägermenge eines Semirings. Das grundlegende Konzept der gewichteten Maschinen wurde im Kontext der Theorie der formalen Potenzreihen in Schützenberger (1961) eingeführt. Im Folgenden wird für eine bessere Lesbarkeit eine rationale Transduktion im Kontext der formalen Sprachen als *rationale Zeichenkettentransduktion* und im Kontext der formalen Potenzreihen als *rationale Potenzreihentransduktion* bezeichnet. Es wird verkürzt rationale Transduktion geschrieben, wenn sowohl Zeichenketten- als auch Potenzreihentransduktion gemeint sind.

Viele theoretische Grundlagen der FSMs und ihrer Gegenstücke aus der

Theorie der formalen Sprachen und Potenzreihen sind schon früh entstanden. Bezüglich der FSAs ist John E. Hopcroft (1979), hinsichtlich der FSTs ist Berstel (1979) und in Hinblick auf die gewichteten Maschinen ist Kuich und Salomaa (1985) ein exzellentes Referenzbuch. Eine gute Einführung in FSMs ist im ungewichteten Fall in Roche und Schabes (1997) und im gewichteten Fall in Mohri (2009) zu finden.

Ein WFST kann als eine Generalisierung eines FSA, FST und WFSA angesehen werden. Aus diesem Grund wird in diesem Abschnitt nur die Definition eines WFST gegeben. Es wird aber dessen Bezug zu den anderen Maschinen verdeutlicht.

2.2.1 Gewichtete endliche Transduktoren

Ein WFST realisiert eine Abbildung von Eingabesymbolsequenzen auf Ausgabesymbolsequenzen und weist jedem Paar zusätzlich ein Semiring-Element zu, das als Gewicht bezeichnet wird. Über dieses können Wahrscheinlichkeiten, Bestrafungen, Zeitdauern oder alle andere Maße, die über einen Semiring beschrieben werden können, abgebildet werden:

Definition 3 (WFST)**.** *Ein WFST über einen Semiring* $S = (A, \oplus, \otimes, \bar{0}, \bar{1})$ *ist ein 8-Tupel* $T = (\Sigma, \Delta, Q, q_0, F, E, \lambda, \rho)$ *das (i)-(viii) erfüllt:*

(i) Σ *ist das endliche Eingabealphabet.*

(ii) Δ *ist das endliche Ausgabealphabet.*

(iii) Q *ist eine endliche Menge von Zuständen.*

(iv) $q_0 \in Q$ *ist der Startzustand.*

(v) $F \subseteq Q$ *ist eine Menge von Endzuständen.*

(vi) $E \subseteq Q \times (\Sigma \cup \epsilon) \times (\Delta \cup \epsilon) \times A \times Q$ *ist eine endliche Menge von Übergängen.*

(vii) $\lambda \in A$ *ist das initiale Gewicht, welches dem Startzustand ein Element aus* A *zuweist.*

(viii) $\rho : F \to A$ *ist die finale Gewichtsfunktion, welche Endzustände auf Elemente in* A *abbildet.*

Gegeben einen Übergang $t \in E$, dann bezeichnet $p[t]$ dessen Quellzustand, $n[t]$ dessen Zielzustand, $i[t]$ dessen Eingabezeichen, $o[t]$ dessen Ausgabezeichen und $w[t]$ dessen Gewicht. Ein Übergang mit $i[t] = \epsilon$ konsumiert keine Eingabe und ein Übergang mit $o[t] = \epsilon$ generiert keine Ausgabe. Ein Übergang mit ϵ als Ein- und Ausgabesymbol wird als Epsilon-Übergang bezeichnet. Ein Pfad π in einem WFST ist eine endliche Sequenz $t_1 t_2 .. t_{|\pi|}$ von $|\pi|$ vielen Übergängen, wobei $n[t_i] = p[t_{i+1}]$ für $1 \leq i \leq |\pi|$ gilt. Der Quellzustand eines Pfades wird mit $p[\pi]$ bezeichnet und der Zielzustand mit $n[\pi]$. Ein erfolgreicher Pfad ist ein Pfad vom Startzustand zum Endzustand mit $p[\pi] = q_0$ und $n[\pi] \in F$. Die Eingabezeichenkette $i[\pi]$ zu einem erfolgreichen Pfad π ist das Ergebnis der Konkatenation der Eingabezeichen der aufeinanderfolgenden Übergänge: $i[\pi] = i[e_1]i[e_2]...i[e_{|\pi|}]$. Entsprechend ist $o[\pi]$ für die Ausgabezeichenkette eines erfolgreichen Pfades π definiert: $o[\pi] = o[e_1]o[e_2]...o[e_{|\pi|}]$. Im Folgenden wird davon gesprochen, dass das Zeichenkettenpaar $(i[\pi],o[\pi])$ akzeptiert oder generiert wird, wenn ein erfolgreicher Pfad π existiert. Wenn die Ein- und Ausgabe separat betrachtet wird, wird davon gesprochen, dass die Zeichenkette $i[\pi]$ auf der Eingabeseite und die Zeichenkette $o[\pi]$ auf der Ausgabeseite akzeptiert oder generiert wird. Wenn die Ein- und Ausgabe miteinander in Bezug gesetzt wird, wird davon gesprochen, dass die Zeichenkette $i[\pi]$ in die Zeichenkette $o[\pi]$ übersetzt wird. Das Gewicht eines erfolgreichen Pfades $w[\pi]$ ist das $\otimes$-Produkt des Startzustandsgewichtes, des Gewichts der aufeinanderfolgenden Übergänge und des Endzustandsgewichts: $w[\pi] = \lambda \otimes w[t_1] \otimes ... \otimes w[t_n] \otimes \rho(n[t_n])$. Die Menge aller erfolgreichen Pfade sei mit Π bezeichnet. Dann ist das Gewicht eines WFST T, geschrieben als $w[T]$, das im Kontext der rationalen Potenzreihen Sprachgewicht genannt wird, über die abstrakte Addition der Gewichte aller Pfade in Π definiert:

Definition 4 (Gewicht)**.** *Für einen WFST T ist sein Gewicht definiert als*

$$w[T] = \bigoplus_{\pi \in \Pi} w[\pi]$$

Das Verhalten eines WFST T, bezeichnet durch $[\![T]\!]$, ist die Potenzreihentransduktion, die er erkennt. Sei die Menge aller erfolgreichen Pfade mit der Eingabezeichenkette x und der Ausgabezeichenkette y durch $\Pi(x, y)$ gegeben, dann ist das WFST-Verhalten wie folgt definiert:

Definition 5 (Rationale Potenzreihentransduktion als Verhalten)**.** *Für alle Zeichenkettenpaare* $(x, y) \in \Sigma^* \times \Delta^*$ *ist das Verhalten* $[\![T]\!]$ *eines WFST* T *definiert als*

$$[\![T]\!](x, y) = \begin{cases} \bigoplus_{\pi \in \Pi(x,y)} w(\pi) & \textit{falls } \Pi(x, y) \neq \varnothing \\ \bar{0} & \textit{sonst} \end{cases}$$

Wenn für alle Zeichenkettenpaare $(x, y) \in \Sigma^* \times \Delta^*$ das Gewicht wohldefiniert und in S ist, wird der WFST als *reguliert* (regulated) bezeichnet. Dies ist z. B. der Fall, wenn ein WFST keine ϵ-Schleifen besitzt. Dann ist die Menge an Pfaden endlich und somit können keine unendlichen Summen auftreten. Dies ist aber auch der Fall, wenn der Semiring vollständig ist (siehe Mohri et al., 2008, Abschnitt 28.3.1). Da in dieser Arbeit ausschließlich WFSTs über c-Semiringe betrachtet werden, sind diese stets reguliert.

Im Gegensatz zu dem Verhalten eines WFST bezeichnet das Verhalten eines WFSAs eine rationale Potenzreihe. Ein WFSA kann hierbei als WFST angesehen werden, bei dem die Ausgabe weggelassen ist. Das Verhalten eines WFSA kann demnach über einen WFST definiert werden:

Definition 6 (Rationale Potenzreihe als Verhalten)**.** *Für alle Zeichenketten* $x \in \Sigma^*$ *ist das Verhalten* $[\![T]\!]$ *eines WFST* T *definiert als*

$$[\![T]\!](x) = \begin{cases} \bigoplus_{\pi \in \Pi(x,x)} w(\pi) & \textit{falls } \Pi(x, x) \neq \varnothing \\ \bar{0} & \textit{sonst} \end{cases}$$

In Definition 6 erhalten ausschließlich Pfade der Form $\Pi(x, x)$ ein Gewicht ungleich $\bar{0}$. Demnach sind hier WFSTs interessant, welche der Restriktion der Identitätstransduktion unterliegen. Das bedeutet, dass alle Übergänge identische Eingabe- und Ausgabesymbole besitzen. Auf diese Weise ist es möglich, einen WFSA über einen gewichteten Identitätstransduktor zu repräsentieren. FSA und FST können als WFST angesehen werden, bei dem die Gewichte (zusätzlich) weggelassen sind. Dies korrespondiert mit dem Support einer rationalen Potenzreihe oder rationalen Potenzreihentransduktion. FSA und FST können aber auch alternativ über einen WFST über den Booleschen Semiring repräsentiert werden. Ein WFST über einem Booleschen Semiring

kann wiederum in einen WFST über einen anderen Semiring überführt werden, indem die Semiring-Elemente entsprechend behandelt werden. So kann ein FSA oder FST auch über einen WFST repräsentiert werden, der über einen beliebigen Semiring definiert ist. Das bedeutet, dass sowohl WFSA als auch FSA und FST über einen WFST repräsentiert werden können (vgl. Mohri, 2009, Abschnitt 1.2). Im Folgenden wird aber weiterhin der allgemeine Begriff FSM verwendet, um auf FSA, FST, WFSA und WFST zu referieren. Wenn eine spezielle Art von Maschine gemeint ist, wird dies explizit angegeben.

Übergangsdiagramm

Um einen WFST grafisch darzustellen, kann er in Form eines Übergangsdiagramms visualisiert werden. Bei solch einem Diagramm werden die Zustände als Kreise repräsentiert, welche nach der entsprechenden Zustandsnummerierung gekennzeichnet sind. Die Übergänge werden als Pfeile dargestellt und sind mit Eingabe- und Ausgabesymbol beschriftet, die durch einen Doppelpunkt getrennt sind. Wenn das Ausgabesymbol dem Eingabesymbol gleicht, wird dieses weggelassen. Die Gewichte werden den entsprechenden Komponenten über einen Schrägstrich hinzugeschrieben. Wenn ein Gewicht dem Nullelement des Semirings entspricht, wird dieses weggelassen. Der Startzustand ist mit einem kurzen Pfeil markiert und die Endzustände mit Doppelkreisen. Die Bausteine eines Übergangsdiagramms sind in Abbildung 2.2 zusammengefasst.

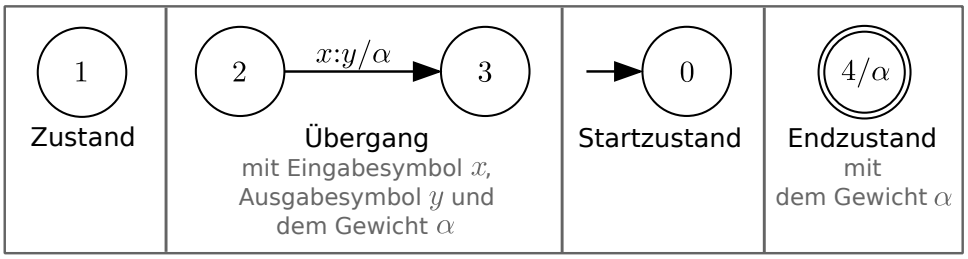


Abbildung 2.2: Bausteine für das Übergangsdiagramm eines WFSTs.

In Abbildung 2.3 ist ein konkretes Beispiel für ein Übergangsdiagramm gegeben. Das Diagramm bezieht sich auf einen WFST über den c-Semiring $(\mathbb{R}_0^+ \cup \{+\infty\}, min, +, +\infty, 0)$. Dieser bildet die Vollform 'Staubecken' auf zwei mögliche Grundformen ab und fügt über Epsilon ein Trennsymbol ('#') ein,

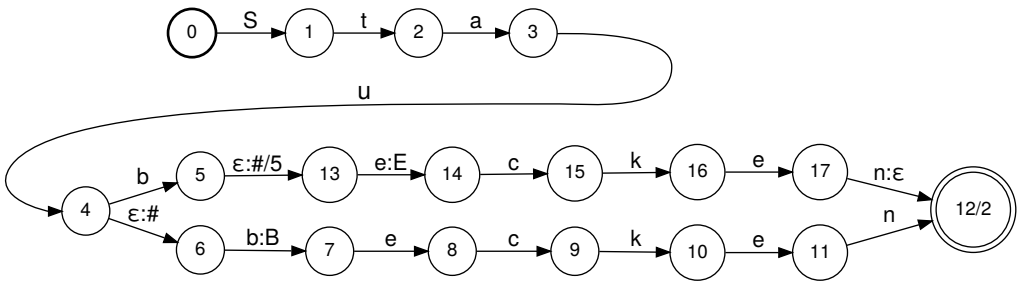


Abbildung 2.3: Übergangsdiagramm zu einem WFST.

um Kompositazerlegungen anzuzeigen. So wird 'Staubecken' einerseits auf 'Stau#Becken' und andererseits auf 'Staub#Ecke' abgebildet. Da eine Zerlegung in die Bestandteile 'Staub' und 'Ecke' recht ungewöhnlich ist wird für diese ein Strafgewicht von 5 vergeben ($\epsilon : \#/5$). Des Weiteren wird beiden Zerlegungen über den Endzustand (zusätzlich) das Strafgewicht 2 zugewiesen. Es ist zu beachten, dass die Zustandsnummerierung in diesem Beispiel beliebig gewählt ist und dass es einen WFST geben kann, der sich von dem Beispiel-WFST nur in der Zustandsnummerierung unterscheidet, aber das gleiche Verhalten aufweist.

Gewichtsbestimmung

Die Bestimmung des Gewichts eines WFST (siehe Definition 4), kann als eine Instanz des *algebraischen Pfadproblems* angesehen werden. Hierbei wird ein WFST wie ein bewerteter gerichteter Graph behandelt. Weil in dieser Arbeit ausschließlich c-Semiringe verwendet werden, ist dieser stets absorptiv bewertet, da der Wert eines Pfades immer $\leq \bar{1}$ ist. Zudem führt die abstrakte Multiplikation der Bewertungen der einzelnen Kanten stets zu Werten, die in der Ordnung weiter unten oder gleichauf liegen, d.h. $a \otimes b \leq a, b$. So kann von der Bestimmung des Wertes des kürzesten Pfades vom Startzustand zu den Endzuständen gesprochen werden (vgl. Hebisch und Weinert, 1998, Abschnitt 4.6). Die Ermittlung dieses Wertes ist ein klassisches *Single-Source-Shortest-Distance-Problem* (siehe Mohri, 2002, Abschnitt 3).

Automatenoperationen

Es existieren verschiedene Automatenoperationen, mit denen regulierte WFSTs manipuliert und kombiniert werden können. Die meisten von diesen sind

Generalisierungen der rationalen Operationen über FSAs. So korrespondieren die WFST-Operationen *Summe*, *Produkt* und *Sternhülle* mit den FSA-Operationen Vereinigung, Konkatenation und Sternhülle. Weitere rationale Operationen sind die *Invertierung*, die *Projektion*, das *Kreuzprodukt* und die *Komposition*. Letztere kann als eine Generalisierung des Hadamard-Produktes von WFSAs angesehen werden, welches mit dem Schnitt von FSAs korrespondiert.

FSTs sind nicht unter der Schnittbildung abgeschlossen. Gleiches gilt für WFSTs bezüglich des Hadamard-Produktes. Die Komplementbildung ist eine Automatenoperation, unter der ausschließlich FSAs abgeschlossen sind. Im Folgenden werden die einzelnen Automatenoperationen definiert und beschrieben, unter denen WFSAs abgeschlossen sind. Für die anderen Operationen wie die Komplementbildung sei auf John E. Hopcroft (1979) verwiesen.

Summe Die Summe zweier WFSTs T_1 und T_2 ergibt einen WFST, der genau die Zeichenkettenpaare erkennt, die von T_1 oder T_2 akzeptiert werden. Wenn ein Zeichenkettenpaar sowohl von T_1 als auch von T_2 erkannt wird, ist dessen Gewicht die abstrakte Addition der Gewichte, die dem Paar über T_1 und T_2 zugewiesen werden. Wenn ein Zeichenkettenpaar hingegen nur von T_1 oder nur von T_2 akzeptiert wird, entspricht dessen Gewicht dem, das dem Paar über T_1 beziehungsweise T_2 zugeordnet wird:

Definition 7 (Summe)**.** *Für alle Paare* $(x, y) \in \Sigma^* \times \Delta^*$ *und für alle regulierten WFSTs* T_1, T_2 *ist die Summe* $T_1 \oplus T_2$ *definiert als*

$$([\![T_1 \oplus T_2]\!])(x, y) = [\![T_1]\!](x, y) \oplus [\![T_2]\!](x, y)$$

Produkt Das Produkt zweier WFSTs T_1 und T_2 ergibt einen WFST, der genau die Zeichenkettenpaare erkennt, die aus der Konkatenation der Zeichenkettenpaare von T_1 mit den Zeichenkettenpaaren aus T_1 gebildet werden können. Die Konkatenation zweier Zeichenkettenpaare meint hierbei, dass ihre Komponenten konkateniert werden und dass so ein neues Zeichenkettenpaar hervorgeht. Das Gewicht, das ein Zeichenkettenpaar über den Ergebnis-WFST erhält, resultiert aus der abstrakten Multiplikation der Gewichte, die dem Teilzeichenkettenpaar über T_1 und T_2 zugewiesen werden. Wenn mehrere Zerlegungen eines Zeichenkettenpaares bezüglich T_1 und T_2 existieren, ist

das Gewicht die abstrakte Addition der Gewichte, die sich über die einzelnen Zerlegungen ergeben:

Definition 8 (Produkt)**.** *Für alle Paare* $(x, y) \in \Sigma^* \times \Delta^*$ *und für alle regulierten WFSTs* T_1, T_2 *ist das Produkt* $T_1 \otimes T_2$ *definiert als*

$$[\![T_1 \otimes T_2]\!](x, y) = \bigoplus_{\substack{x=x_1 x_2 \\ y=y_1 y_2}} [\![T_1]\!](x_1, y_1) \otimes [\![T_2]\!](x_2, y_2)$$

Sternhülle Das n-fache Produkt $T_1 \otimes ... \otimes T_n$ mit $n > 0$ wird mit T^n bezeichnet. T^0 ist der WFST, der dem Zeichenkettenpaar (ϵ, ϵ) das Gewicht $\bar{1}$ zuweist und jedem anderen das Gewicht $\bar{0}$. Die Sternhülle eines WFST ist dessen ungebundene Iteration. Bei dieser Operation muss der Semiring bezüglich unendlicher Summen abgeschlossen sein. Dies ist bei einem c-Semiring der Fall:

Definition 9 (Sternhülle)**.** *Für alle Paare* $(x, y) \in \Sigma^* \times \Delta^*$ *und für alle regulierten WFST* T *über einen vollständigen Semiring ist die Sternhülle* T^* *definiert als*

$$[\![T^*]\!](x, y) = \bigoplus_{n=0}^{+\infty} [\![T]\!]^n(x, y)$$

Invertierung Die Invertierung eines WFST T ergibt einen WFST, bei dem die Ein- und Ausgabe von T vertauscht sind:

Definition 10 (Invertierung)**.** *Für alle Paare* $(x, y) \in \Sigma^* \times \Delta^*$ *und für alle regulierten WFST* T *ist die Invertierung* T^{-1} *definiert als*

$$[\![T^{-1}]\!](x, y) = [\![T]\!](y, x)$$

Projektion Die Projektion eines WFST ergibt einen WFSA, der dessen Eingabe (erste Projektion) oder dessen Ausgabe (zweite Projektion) erkennt. Das Gewicht, das einer Zeichenkette über den WFSA zugewiesen wird, entspricht dem Gewicht, dass dem zugrunde liegenden Zeichenkettenpaar über den WFST zugeteilt wird. Wenn einer Zeichenkette mehrere Zeichenkettenpaare zugrunde liegen, ergibt sich das Gewicht aus der abstrakten Addition der Gewichte, die den einzelnen Zeichenkettenpaaren zugeordnet sind:

Definition 11 (erste Projektion)**.** *Für alle $x \in \Sigma^*$ und für alle regulierten WFST T ist die erste Projektion definiert als*

$$[\![T]\!](x) = \bigoplus_{y \in \Delta^*} [\![T]\!](x, y)$$

Definition 12 (zweite Projektion)**.** *Für alle $x \in \Sigma^*$ und für alle regulierten WFST T ist die zweite Projektion definiert als*

$$[\![T]\!](y) = \bigoplus_{x \in \Delta^*} [\![T]\!](x, y)$$

Kreuzprodukt Das Kreuzprodukt zweier WFSAs T_1 und T_2 ergibt einen WFST, der auf der Eingabeseite genau die Zeichenketten erkennt, die T_1 akzeptiert und der auf der Ausgabeseite genau die Zeichenketten erkennt, die T_2 akzeptiert. Das Gewicht, das einem Zeichenkettenpaar über den WFST zugewiesen wird, resultiert aus der abstrakten Multiplikation der Gewichte, die den Komponenten über T_1 und T_2 zugeordnet werden. Das Kreuzprodukt fordert, dass der Semiring der Operanden kommutativ ist. Da in dieser Arbeit c-Semiringe verwendet werden, ist dies stets Fall. In der folgenden Definition wird ein WFSA durch einen WFST repräsentiert:

Definition 13 (Kreuzprodukt)**.** *Für alle Paare $(x, y) \in \Sigma^* \times \Delta^*$ und für alle regulierten WFSTs T_1, T_2 über einem kommutativen Semiring ist das Kreuzprodukt $T_1 \times T_2$ definiert als*

$$[\![T_1 \times T_2]\!](x, y) = [\![T_1]\!](x) \otimes [\![T_2]\!](y)$$

Komposition Die Komposition zweier WFSTs T_1 und T_2 ergibt einen WFST, der dessen Hintereinanderschaltung realisiert. Wenn T_1 eine Zeichenkette x in die Zeichenkette z übersetzt und T_2 die Zeichenkette z in die Zeichenkette y, genau dann erkennt der Ergebnis-WFST das Zeichenkettenpaar (x, y). Das Gewicht, das einem Zeichenkettenpaar über den Ergebnis-WFST zugewiesen wird, ergibt sich aus der abstrakten Multiplikation von dem Gewicht, das dem Zeichenkettenpaar (x, z) über T_1 zugeteilt wird, mit dem Gewicht, das dem Zeichenkettenpaar (z, y) über T_2 zugeordnet wird. Die Komposition fordert wie auch das Kreuzprodukt, dass der Semiring der Operanden kommutativ ist. Da in dieser Arbeit c-Semiringe verwendet werden, ist dies stets Fall:

Definition 14 (Komposition). *Für alle Paare* $(x, y) \in \Sigma^* \times \Delta^*$ *und für alle regulierten WFSTs* T_1, T_2 *über einem kommutativen Semiring, wobei* T_1 *das Ein- und Ausgabealphabet* Σ *und* Γ *und* T_2 *das Ein- und Ausgabealphabet* Γ *und* Δ *besitzt, ist die Komposition* $T_1 \circ T_2$ *definiert als*

$$[\![T_1 \circ T_2]\!](x, y) = \bigoplus_{z \in \Gamma} [\![T_1]\!](x, z) \otimes [\![T_2]\!](z, y)$$

Wenn zwei WFSAs über zwei Identitätstransduktoren repräsentiert werden, korrespondiert deren Komposition mit dem Hadamard-Produkt über WFSAs. Die Kompositionsoperation ist grundlegend für die Anwendung linguistischer Regeln oder statistischer Modelle. Ein Beispiel hierfür sind Prozess-Pipelines, bei denen verschiedene Analyse-Prozesse hintereinandergeschaltet werden. Im Rahmen der linguistischen Textaufbereitung besteht solch eine Pipeline beispielsweise aus der Hintereinanderschaltung von Texttokenisierung, morphologischer Analyse und Wortartendisambiguierung (siehe z. B. Tzoukerman und Radev, 1999). Solch eine Kaskade kann in einer Offline-Berechnung (teilweise) zusammengefasst oder online nacheinander abgearbeitet werden. Ein anderes Beispiel ist die iterative Komposition eines WFSTs mit seiner eigenen Ausgabe, bei der z. B. syntaktische Strukturen aufgebaut werden können. Diese Iteration kann hierbei gebunden oder im Rahmen eines erweiterten endlichen Ansatzes über einen Fixpunktalgorithmus geregelt sein (siehe z. B. Roche, 1997).

Äquivalenztransformationen

Es ist möglich, WFSTs von Epsilon-Übergängen zu befreien, zu determinisieren und zu minimieren, um primär ihre Zeiteffizienz zu steigern und sekundär ihren Platzbedarf zu verringern. Diese drei Schritte stellen Transformationen dar, die zwar eine FSM verändern, aber dessen Verhalten erhalten. Diese Äquivalenztransformationen werden im Folgenden kurz erläutert.

Nichtdeterministische FSMs und deren Umwandlung in deterministische Maschinen wurden erstmals von Rabin und Scott (1959, Abschnitt 2.5) im Kontext von FSAs thematisiert. Ein WFSA wird als deterministisch bezeichnet, wenn er keine Epsilon-Übergänge enthält und in jedem Zustand für ein Eingabesymbol genau einen Folgezustand existiert. Ein Epsilon-freier nichtdeterministischer WFSA kann in einen deterministischen WFSA überführt werden, der das gleiche Verhalten besitzt (siehe Mohri, 1997, Abschnitt 3.3).

Der resultierende WFSA kann im ungünstigsten Fall eine Zustandsanzahl aufweisen, die exponentiell größer ist. Jedoch kann nicht jede rationale Potenzreihe von einem deterministischen WFSA erkannt oder generiert werden. Demnach kann nicht jeder WFSA determinisiert werden. Dies ist aber dann möglich, wenn er die sogenannte Zwillingseigenschaft besitzt. Ein azyklischer WFSA hat beispielsweise stets diese Eigenschaft (siehe Mohri, 1997, Abschnitt 3.5).

Ab wann ein Transduktor als deterministisch bezeichnet wird, ist in der Literatur unterschiedlich definiert (vgl. Roche und Schabes, 1997, Abschnitt 1.3.8). Hier wird ein WFST deterministisch genannt, wenn er keine Epsilon-Übergänge besitzt und in jedem Zustand für ein Ein- und Ausgabezeichen exakt ein Folgezustand existiert. In anderen Arbeiten wird ein WFST hingegen als deterministisch oder auch als sequenziell bezeichnet, wenn es in jedem Zustand für ein Eingabesymbol genau ein Folgezustand gibt und die Eingabe keine Epsilon-Symbole enthält, also wenn die Eingabe deterministisch ist (siehe Mohri, 1997, Abschnitt 2.1). Diesbezüglich ist schon das Überführen eines nichtdeterministischen FSTs in einen deterministischen beziehungsweise sequenziellen nicht immer möglich (siehe Mohri, 1997, Abschnitt 3.4). Wenn ein WFST als ein WFSA interpretiert wird, bei dem die einzelnen Zeichenpaare als ein Alphabet-Symbol angesehen werden, kann dieser im Sinne eines WFSA determinisiert werden.

Ein deterministischer WFSA wird als minimal bezeichnet, wenn es keinen anderen deterministischen WFSA gibt, der weniger Zuständen besitzt und das gleiche Verhalten aufweist. Dies ist analog zu dem Myhill-Nerodes-Theorem für deterministische endliche Automaten (siehe John E. Hopcroft, 1979, Abschnitt 3.4). Es ist möglich einen deterministischen WFSA in einen minimalen deterministischen WFSA zu überführen (siehe z. B. Eisner, 2003). Wenn ein WFST als ein WFSA interpretiert wird, bei dem die einzelnen Zeichenpaare als ein Alphabet-Symbol angesehen werden, kann dieser im Sinne eines WFSA minimiert werden. So wird ein WFST, der deterministisch im Sinne eines WFSA ist, in dieser Arbeit als minimal bezeichnet, wenn er zudem im Sinne eines WFSA minimal ist. Es ist auch möglich einen sequenziellen WFST, also einen WFST mit deterministischer Eingabe, in einen minimalen sequenziellen WFST zu überführt (siehe Mohri, 1997, Abschnitt 3.7). Diese Arbeit beschränkt sich aber auf WFSTs, die minimal im Sinne eines WFSA sind.

Die Anwendung der drei Schritte Epsilon-Entfernung, Determinisierung und Minimierung wird im Folgenden als *Optimierung* bezeichnet und der resultierende WFSA als *optimal*. Das Ergebnis der Anwendung von rationalen Operationen kann zu WFSTs führen, die nicht optimal sind. Dies ist z. B. bei der klassischen Thompson-Konstruktion der Fall, bei der aus einem regulären Ausdruck mit den Operatoren Konkatenation, Vereinigung und Sternhülle eine FSM berechnet wird (im Kontext der Programmcode-Kompilierung siehe Thompson, 1968). Um einen optimalen WFST zu erhalten, müssen dann entsprechende Äquivalenztransformationen angewendet werden.

2.2.2 Kompilierung

Über FSMs können sowohl statistische als auch regelbasierte Paradigmen umgesetzt und kombiniert werden. Ein Beispiel für einen regelbasierten Ansatz ist die Implementierung von *Kontextbeschränkungsregeln* (context restriction rules) innerhalb des methodologischen Paradigmas der *Constraint-Grammatik* (Constraint Grammar, siehe Karlsson, 1990 und Koskenniemi, 1990). Regelbasierte Ansätze sind gerade deswegen interessant, weil dort die Maschinen ausschließlich über Automatenoperationen kompiliert werden, wie sie in Abschnitt 2.2.1 beschrieben sind. Ein Beispiel für einen statistischen Ansatz ist die Implementierung von N-Gram-Sprachmodellen für den Einsatz in der Spracherkennung mit großem Vokabular (siehe z. B. Mohri et al., 2002). Dort geschieht die Erstellung der Modelle jedoch teilweise über eine direkte Manipulation von Zuständen und Übergängen und nicht über die klassischen Automatenoperationen. In Hanneforth und Würzner (2009) ist ein Vorgehen beschrieben, wie auch N-Gram-Sprachmodelle ausschließlich über die gängigen Operationen erstellt werden können. In dieser Arbeit wird ebenfalls diesem Grundsatz gefolgt. Demnach meint hier die Kompilierung von FSMs die sukzessive Anwendung der klassischen Automatenoperationen wie sie in Abschnitt 2.2 beschrieben sind.

Die Erstellung von FSMs für komplexere Aufgaben ist meist kompliziert und erfordert eine hohe Abstraktion von den zugrunde liegenden Automatenoperationen. Ein Beispiel hierfür ist die *Zwei-Ebenen-Morphologie* (two-level morphology, Koskenniemi, 1983). Dieser Ansatz realisiert eine morphologische Analyse über parallele *Zwei-Ebenen-Regeln* (two-level rules). In der initialen Version wurden Regel-FSMs per Hand direkt über die Zustände und

Übergänge erstellt. Die Semantik der Regeln wahr zwar vollständig definiert, jedoch fehlte eine Methode, um die FSMs automatisch zu kompilieren. Erst einige Jahre später war ein Regel-Compiler entwickelt und verfügbar (Karttunen et al., 1987). Reguläre Ausdrücke sind hierbei der Schlüssel, um FSMs für komplexe Aufgaben zu kompilieren. Erstmals eingeführt wurden sie in Kleene (1956, Abschnitt 7) im Kontext von regulären Mengen. Sie haben eine klare deklarative Semantik und gleichzeitig den Charakter einer höheren Programmiersprache. Schnell hat sich ein eigener Forschungsgegenstand entwickelt, der sich ausschließlich mit der Kompilation von FSMs über reguläre Ausdrücke befasst.

Es existieren zwei grundlegende Mechanismen, die für die Kompilation von Regeln in eine FSM meist unabdingbar sind. Das ist auf der einen Seite die doppelte eingebettete Komplementbildung (Negation), die z. B. bei der Implementierung von Kontextbeschränkungsregeln herangezogen wird. Andererseits ist es die Verwendung von Markierungssymbolen, die zwar während des Kompilierungsvorgangs vorhanden sind, aber in der Ergebnismaschine nicht mehr auftreten. Diese Markierungssymbole werden insbesondere dafür verwendet Überlappungen von Mustern, die innerhalb von Regeln definiert sind, korrekt zu behandeln. Hierfür werden Sie mit der doppelten eingebetteten Komplementbildung verbunden (siehe Yli-Jyrä, 2008). Dies ist z. B. bei der Kompilation von Zwei-Ebenen-Regeln der Fall. Genau dieses Zusammenspiel von Markierungssymbolen und der Komplementbildung macht sich Hulden (2008) zunutze, um reguläre Ausdrücke um reguläre Prädikatenlogik zu erweitern, bei der Propositionen auf Teilzeichenketten angewendet werden. Die dort verwendeten Formeln, die hier als *reguläre prädikatenlogische Formeln* bezeichnet werden, stellen eine weitere Abstraktionsstufe bei der Erstellung von FSMs dar. Der Vorteil dieser Erweiterung ist die einheitliche Notation, die es z. B. möglich macht Zwei-Ebenen-Regeln zu definieren, ohne selbst Markierungssymbole einführen zu müssen.

Der Zusammenhang zwischen den FSMs, den formalen Sprachen und Potenzreihen und den regulären Ausdrücken und regulären prädikatenlogischen Formeln ist in Abbildung 2.4 verdeutlicht. Im Folgenden wird genauer auf die Kompilation von FSMs mithilfe von regulären Ausdrücken und regulären prädikatenlogischen Formeln eingegangen.

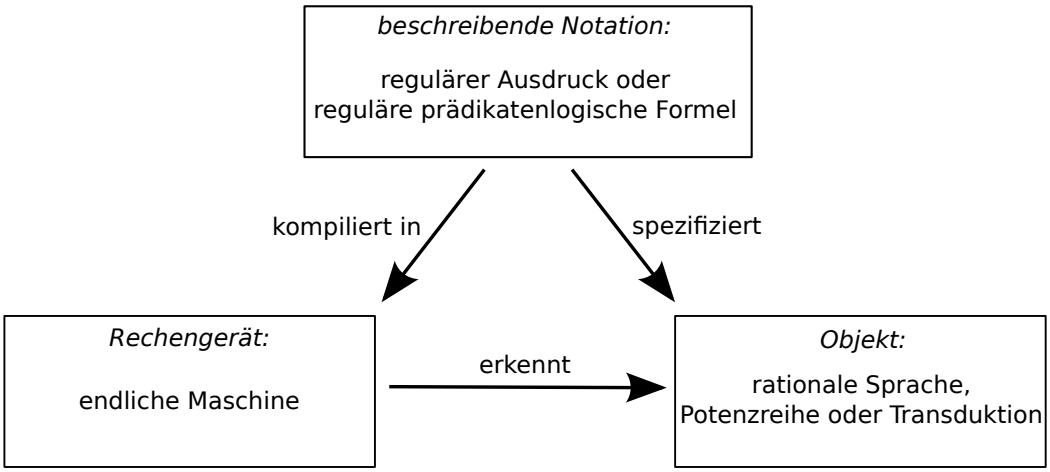


Abbildung 2.4: Spezifizierung und Kompilierung.

Reguläre Ausdrücke

Die Algebra von regulären Ausdrücken besteht aus zwei Teilen: Konstanten und Operatoren. Die Konstanten selbst sind reguläre Ausdrücke und spezifizieren rationale Sprachen, rationale Potenzreihen und rationale Transduktionen und stellen die Grundbausteine dar. Über die Operatoren können aus einfachen Ausdrücken komplexere erstellt werden. Einerseits denotieren sie Operationen auf rationale Sprachen, rationale Potenzreihen oder rationale Transduktionen, andererseits sind sie eine verkürzende Schreibweise. Ein Formalismus für reguläre Ausdrücke, der die Operatoren Konkatenation, Vereinigung und Sternhülle umfasst, reicht aus, um jede rationale Sprache zu spezifizieren und jeden FSA zu kompilieren (siehe Kleene, 1956, Abschnitt 7 oder John E. Hopcroft, 1979, Abschnitt 2.5). Wenn geeignete Konstanten einbezogen werden, gilt Entsprechendes auch für FSTs (siehe Kaplan und Kay, 1994, Abschnitt 3.1) und für gewichtete FSMs (siehe Kuich und Salomaa, 1985, Abschnitt 2.8).

Eine weitverbreitete Notation für reguläre Ausdrücke ist das *Xerox finite state calculus* (für einen guten Überblick siehe Karttunen et al., 1996 und für eine ausführlichere Beschreibung mit Beispielen siehe Beesley und Karttunen, 2003). Dort beziehen sich die Ausdrücke auf FSAs und FSTs beziehungsweise auf rationale Sprachen und rationale Zeichenkettentransduktionen. Die klassischen Operatoren wie Schnitt, Konkatenation und Sternhülle sind um viele andere erweitert, um einen hohen Abstraktionsgrad zu erhalten. Dies

sind z. B. solche, die Kontextbeschränkungen oder Ersetzungen implementieren. Jedoch ist innerhalb der Notation der Bezug von den Konstanten und Operatoren zu der Theorie der formalen Sprachen nicht immer intuitiv. So bezeichnet dort der Operator '&' die Schnittbildung und das Symbol '[]' die leere Sprache. Die Notation für reguläre Ausdrücke, die von dem Compiler *Foma* verwendet wird, erlaubt hingegen den Gebrauch von Unicode-Zeichen für mathematische Operatoren und Symbole (Hulden, 2009, Abschnitt 2). So bezeichnet dort der Operator '$\cap$' die Schnittbildung und die Konstante '$\varnothing$' die leere Sprache. Hierdurch wird eine direkte Verbindung zu der Theorie der formalen Sprachen möglich. Die Notationsform, die in der folgenden Arbeit verwendet wird, folgt diesem Grundsatz, um einen intuitiven Bezug zu der Theorie der formalen Sprachen und Potenzreihen herzustellen.

Verwendete Notation Im Folgenden wird die Notation der regulären Ausdrücke gegeben, die in dieser Arbeit verwendet wird. Diese spezifizieren rationale Sprachen, rationale Potenzreihen oder rationale Transduktionen. Die Möglichkeit des Vermischens der Ausdrücke für die verschiedenen FSM-Typen ist über die Repräsentation als WFST möglich (vgl. Abschnitt 2.2.1). Im Folgenden wird erst auf die Konstanten eingegangen und dann auf die Operatoren. Hierbei beziehen sich die Definitionen auf das Alphabet Σ und auf die Semiring-Trägermenge A.

Ein Ausdruck x mit $x \in \Sigma$ spezifiziert die rationale Sprache $\{x\}$, der Ausdruck ϵ die rationale Sprache $\{\epsilon\}$, der Ausdruck $\langle\alpha\rangle$ mit $\alpha \in A$ die rationale Potenzreihe $\alpha\epsilon$ und der Ausdruck $\varnothing$ die leere Menge. Der Ausdruck $x{:}y$ mit $x \in (\Sigma \cup \epsilon)$ und $y \in (\Sigma \cup \epsilon)$ spezifiziert eine rationale Zeichenkettentransduktion, die über die reguläre Relation (x, y) definiert ist.[1] Innerhalb eines regulären Ausdrucks ist es möglich, ein Alphabet anzugeben. So spezifiziert der Ausdruck Δ die rationale Sprache $\{w|w$ ist eine Zeichenkette der Länge 1 über dem Alphabet $\Delta \subseteq \Sigma\ \}$. Eine Menge von Zeichen kann auch kommagetrennt in geschweiften Klammern geschrieben werden. Für das Alphabet $\Delta = \{a, b, c\}$ kann dann alternativ der Ausdruck $\{a, b, c\}$ notiert werden. Die endlichen Automaten, die über die einzelnen Konstanten kompiliert werden können, sind in Abbildung 2.5 skizziert. Angenommen, das Alphabet Σ enthält alle klein- und großgeschriebenen Buchstaben und angenommen es bein-

[1] Eine reguläre Relation $R \subseteq A^* \times B^*$ definiert eine Zeichenkettentransduktion $\tau : A^* \to B^*$ durch $\tau(x) = \{y \in B^* \mid (x, y) \in R\}$ (siehe Berstel, 1979, Abschnitt 3.4).

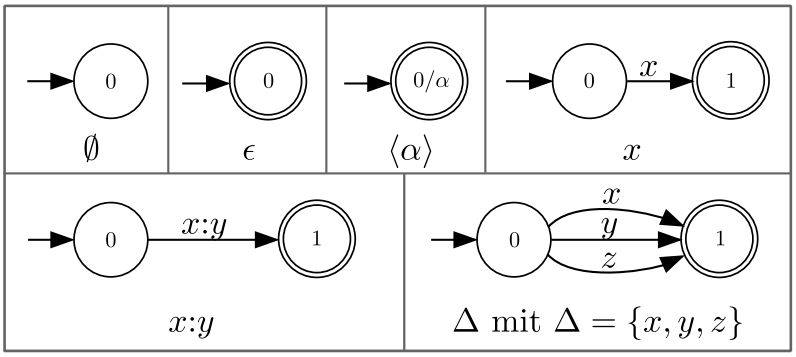


Abbildung 2.5: Konstanten und ihre korrespondierenden FSMs.

haltet auch das Zeichen `SYMBOL`. Dann könnte der reguläre Ausdruck `SYMBOL` sowohl das Zeichen als auch die Konkatenation einzelner Buchstaben bedeuten. Im Folgenden wird über den gegebenen Kontext eindeutig sein, wie solch ein Ausdruck zu interpretieren ist.

Die Operatoren sind in der Tabelle 2.1 zusammengefasst. Sie sind von der höchsten bis zur niedrigsten Präzedenz absteigend geordnet, wobei horizontale Linien die Präzedenzklassen trennen. Die Notation orientiert sich an den rationalen Sprachen und Relationen. Die Subtraktion $A - B$ ist äquivalent zu $A \cap \neg B$. Die rechtsseitige Ersetzung von A durch B, geschrieben als $A \mapsto B$, ist definiert durch $(\neg(\Sigma^* \ A \ \Sigma^*)(A \times B))^* \neg(\Sigma^* \ A \ \Sigma^*)$, die linksseitige Ersetzung $A \leftarrow B$ hingegen durch $(A \mapsto B)^{-1}$. Die rechtsseitige optionale Ersetzung von A durch B, geschrieben als $A(\mapsto)B$, ist definiert durch $(\Sigma^*(A \times B))^*\Sigma^*$, die linksseitige optionale Ersetzung $A(\leftarrow)B$ hingegen durch $(A(\mapsto)B)^{-1}$. Ein Ausdruck der Form $Range(A \circ B)$ wird auch als die Anwendung von B auf A bezeichnet. Gegeben einen Ausdruck $A \cup B$, dann wird im Fall von rationalen Sprachen oder rationalen Zeichenkettentransduktionen von der Vereinigung von A und B gesprochen und bei rationalen Potenzreihen oder rationalen Potenzreihentransduktionen von der der Summe von A und B. Entsprechendes gilt für die Ausdrücke $A \ B$ und $A \cap B$. Nun ist es möglich, für den WFST aus der Beispielabbildung 2.3 einen regulären Ausdruck anzugeben:

$$\texttt{Stau (b } \epsilon\texttt{:\# } \langle 5 \rangle \texttt{ e:E cke n:}\epsilon \cup \epsilon\texttt{:\# b:B ecken) } \langle 2 \rangle \tag{2.1}$$

Im Folgenden wird für eine rationale Sprache, rationalen Potenzreihe oder rationale Transduktion, die von einem regulären Ausdruck r spezifiziert wird, $L(r)$ geschrieben, für eine FSM, die aus einem Ausdruck r kompiliert wird,

$Dom(A)$	erste Projektion/Definitionsbereich
$Range(A)$	zweite Projektion/Wertebereich
$Support(A)$	Support
$Best(A)$	Zeichenketten mit dem besten Gewicht
A^{-1}	Invertierung
A^*	Sternhülle
A^+	Plushülle
A^n	n-fache Iteration
$\neg A$	Komplementbildung
$A\ B$	Konkatenation/Produkt
$A \cup B$	Vereinigung/Summe
$A \cap B$	Schnitt/Hadamard-Produkt
$A - B$	Subtraktion
$A \times B$	Kreuzprodukt
$A \mapsto B$	rechtsseitige Ersetzung
$A \leftarrowtail B$	linksseitige Ersetzung
$A(\mapsto)B$	rechtsseitige optionale Ersetzung
$A(\leftarrowtail)B$	linksseitige optionale Ersetzung
$A \circ B$	Komposition

Tabelle 2.1: Operatoren.

hingegen $A(r)$. Wenn es für einen gegeben Ausdruck r in einem gegebenen Kontext eindeutig ist, dass $L(r)$ oder $A(r)$ gemeint ist, wird verkürzt r notiert.

Im Folgenden ist die Einbeziehung von rationalen Sprachen, rationalen Potenzreihen, rationalen Transduktionen oder FSMs in reguläre Ausdrücke erlaubt. Wenn $\mathcal{A}_1$ und $\mathcal{A}_2$ zwei FSMs sind, kann z. B. der Ausdruck $(\mathcal{A}_1 \cup \mathcal{A}_2)^*$ notiert werden. Korrespondierend hierzu kann der Ausdruck $(\mathcal{L}_1 \cup \mathcal{L}_2)^*$ geschrieben werden, wenn $\mathcal{L}_1$ und $\mathcal{L}_2$ für zwei rationale Sprachen, rationale Potenzreihen oder Transduktionen stehen.

Wenn ein beliebiges Konstrukt α über eine rationale Sprache, rationale Potenzreihe oder rationale Transduktion beschrieben wird, wird im Folgenden $L(\alpha)$ geschrieben. Wenn α hingegen über eine FSM repräsentiert wird, wird $A(\alpha)$ notiert. Wenn eine Beschreibung $L(\alpha)$ existiert, dann gibt es auch eine Repräsentation $A(\alpha)$ und andersherum. Hierbei erkennt oder generiert die FSM $A(\alpha)$ die rationale Sprache, rationale Potenzreihe oder rationale Transduktion $L(\alpha)$. Demnach kann z. B. für $(L(\alpha) \cup L(\alpha))^*$ auch

$(A(\alpha) \cup A(\alpha))^*$ geschrieben werden.

Generalisiertes Übergangsdiagramm

Bei dem generalisierten Übergangsdiagramm sind im Gegensatz zum normalen Übergangsdiagramm die Pfeile mit regulären Ausdrücken statt mit einzelnen Symbolen oder Symbolpaaren versehen (vgl. Abschnitt 2.2.1). Die Gewichtungen sind nicht über den Schrägstrich den Komponenten hinzugefügt, da diese aus den regulären Ausdrücken resultieren. Zudem ist die Zustandsbenennung weggelassen. Solch ein Diagramm wird z. B. in Kapitel 3 verwendet, um die Umsetzung von Constraint-Erfüllung und -Optimierung mit FSMs zu verdeutlichen. In Abbildung 2.6 ist ein Beispiel für solch ein generalisiertes Übergangsdiagramm gegeben. Dieses bezieht sich auf das Übergangsdiagramm in Abbildung 2.3 aus Abschnitt 2.2.1 und auf den regulären Ausdruck in 2.1 und kombiniert beides.

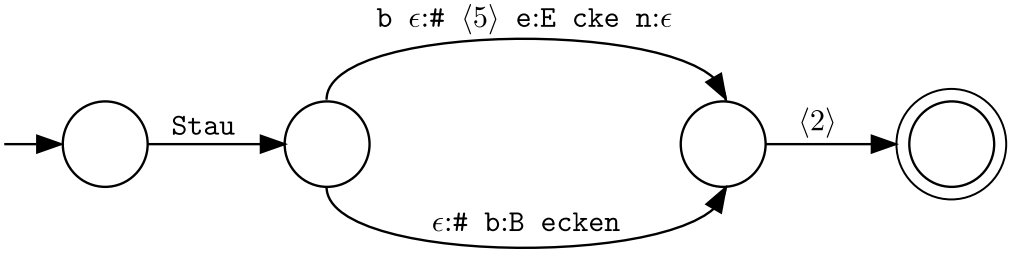


Abbildung 2.6: Generalisiertes Übergangsdiagramm.

Reguläre prädikatenlogische Formeln

Über reguläre Ausdrücke können Quantifizierungen, wie sie in Tabelle 2.2 aufgeführt sind, ausgedrückt werden. Reguläre prädikatenlogische Formeln

$\Sigma^*\texttt{ab}\Sigma^*$	es existiert ein `a` direkt gefolgt von einem `b`
$\neg(\Sigma^*\texttt{ab}\Sigma^*)$	es existiert keine `ab`-Sequenz
$\Sigma^*\texttt{a}\neg(\texttt{b}\Sigma^*)$	es existiert ein `a` dem etwas anderen folgt als ein `b`
$\neg(\Sigma^*\texttt{a}\neg(\texttt{b}\Sigma^*))$	jedes `a` wird von einem `b` gefolgt

Tabelle 2.2: Quantifizierte Ausdrücke.

sind als eine Erweiterung der regulären Ausdrücke zu verstehen, über die auch komplexere Quantifizierungen auf höherer Abstraktionsebene umgesetzt

werden können. Hierbei werden Existenz- und Allquantoren und Junktoren verwenden, um Bedingungen zu formulieren, wobei Propositionen auf Teilzeichenketten angewendet werden und mehrfaches Quantifizieren möglich ist. Über die Formeln können ausschließlich rationale Sprachen spezifiziert werden. Über eine bestimmte Symbolcodierung ist es aber auch möglich reguläre Relationen zu behandeln, um beispielsweise Zwei-Ebenen-Regeln zu kompilieren. In der folgenden Arbeit werden solche Symbolcodierungen jedoch nicht verwendet.

Der Compiler Foma erlaubt es, reguläre prädikatenlogische Formeln innerhalb von regulären Ausdrücken zu verwenden (Hulden, 2009, Abschnitt 2). Über Unicode-Zeichen ist dort eine Notation umgesetzt, die nahe an den klassischen prädikatenlogischen Formeln ist. Die Notationsform, die in der folgenden Arbeit verwendet wird, folgt diesem Grundsatz, um einen intuitiven Bezug zu Prädikatenlogik herzustellen. Sie orientiert sich an Kutschera und Breitkopf (2000).

Im Folgenden wird eine Schreibweise aus Kutschera und Breitkopf (2000, Abschnitt 9.1) herangezogen, die für die darauffolgenden Definitionen relevant ist und die sich auf die Ersetzung von Teilausdrücken bezieht:

Definition 15 (Ersetzung)**.** *Wenn $A[*]$ für einen Ausdruck steht, ist der Ausdruck $A[\alpha]$ derjenige Ausdruck, der aus $A[*]$ entsteht, wenn man in $A[*]$ das Symbol '$*$' an jeder Stelle durch einen Ausdruck α ersetzt. Entsprechend wird die Ersetzung bezüglich der Ausdrücke $A[*_1, ..., *_n]$ und $A[\alpha_1, ..., \alpha_n]$ verstanden.*

Neben dem Existenzquantor und dem Allquantor werden auch der Eindeutigkeitsquantor (siehe z. B. Kutschera und Breitkopf, 2000, Abschnitt 13.1) und der Optionalitätsquantor verwendet, die die klassischen prädikatenlogischen Quantoren erweitern. Diese sind wie folgt definiert:

Definition 16 (Eindeutigkeitsquantor)**.** *Der Eindeutigkeitsquantor, geschrieben $\exists!$, sagt aus, dass genau ein Ding mit einer bestimmten Eigenschaft existiert:*

$$\exists! x\ A[x] \ :=\ \exists x \forall y\ A[y] \leftrightarrow x = y$$

Definition 17 (Optionalitätsquantor)**.** *Der Optionalitätsquantor, geschrieben $\exists?$, sagt aus, dass es genau ein Ding mit einer bestimmten Eigenschaft gibt oder kein solches:*

$$\exists ?x\ A[x] \ :=\ (\neg \exists x\ A[x]) \vee (\exists !x\ A[x])$$

Verwendete Notation Im Folgenden wird die Notation der regulären prädikatenlogischen Formeln gegeben, die in dieser Arbeit verwendet wird. Erst wird auf die Variablen und Wahrheitswerte eingegangen und dann auf die logischen Operatoren und Prädikate und zum Schluss auf die Kapselung von Teilausdrücken.

Logische Variablen werden mit Kleinbuchstaben notiert ($a - z$). Das Zeichen $\top$ spezifiziert den Wahrheitswert 'wahr' und das Zeichen $\bot$ den Wahrheitswert 'falsch'. Die runden Klammern werden verwendet, um Ausdrücke zu gruppieren. In Tabelle 2.3 sind die einzelnen logischen Operatoren zusammengefasst. Diese sind von der höchsten bis zur niedrigsten Präzedenz absteigend geordnet, wobei horizontale Linien die Präzedenzklassen trennen. In Tabelle 2.4 sind die logischen Prädikate gegeben, die grundlegend unterstützt werden.

$\exists$	Existenzquantor
$\forall$	Allquantor
$\exists !$	Eindeutigkeitsquantor
$\exists ?$	Nichtexistenz-Eindeutigkeitsquantor
$\neg\alpha$	logische Negation
$\alpha \wedge \beta$	logisches Und
$\alpha \vee \beta$	logisches Oder
$\alpha \veebar \beta$	logisches exklusives Oder
$\alpha \rightarrow \beta$	logische Implikation
$\alpha \leftrightarrow \beta$	logische Äquivalenz

Tabelle 2.3: Logische Operatoren.

$x \in A$	Sprachzugehörigkeit zu der rationalen Sprache L(A)
$x = y$	Identität (Positionsäquivalenz)
$x \neq y$	Nichtidentität (Positionsinäquivalenz)
$x \prec y$	x führt y (Positionsabfolge)
$x \succ y$	x folgt y (Positionsabfolge)
$x \mid y$	x und y sind konvex (direkter Positionsanschluss)

Tabelle 2.4: Logische Prädikate.

Bei mehrfachem Quantifizieren kann für $\exists x \exists y$ vereinfacht $\exists xy$ geschrieben werden und für $\forall x \forall y$ der Ausdruck $\forall xy$. Für $x \prec y \wedge y \prec z$ kann vereinfacht $x \prec y \prec z$ notiert werden und für $x \succ y \wedge y \succ z$ der Ausdruck $x \succ y \succ z$. Für $x = y \wedge y = z$ kann verkürzt $x = y = z$ geschrieben werden.

Es soll ein Beispiel für eine reguläre prädikatenlogische Formel gegeben werden. Hierzu wird an das Kompositumzerlegungsbeispiel aus Abschnitt 2.2.1 angeknüpft. Die folgende prädikatenlogische Formel drückt aus, dass das Symbol # nur dann auftreten darf, wenn es zwischen den Teilzeichenketten '`Stau`' und '`Becken`' steht:

$$\forall x \; x \in \texttt{\#} \rightarrow \exists yz \; y \in \texttt{Stau} \wedge z \in \texttt{Becken} \wedge y \prec x \prec z \wedge x \mid y \wedge x \mid z \tag{2.2}$$

Nach dieser Formel ist das Wort '`Stau#Becken`' erlaubt aber das Wort '`Staub#Ecke`' nicht. Da die Restriktion für Teilzeichenketten formuliert ist, ist aber auch ein Wort wie '`xxxStau#Beckenxxx`' möglich. Die oben definierte Einschränkung kann alternativ auch mithilfe eines regulären Ausdrucks umgesetzt werden:

$$\neg((\neg(\Sigma^* \texttt{ Stau } \texttt{\#} \; \Sigma^*) \cup (\Sigma^* \texttt{ \# } \neg(\texttt{ Becken } \Sigma^*))) \tag{2.3}$$

Der Ausdruck enthält mehrere Komplementbildungen und ist schwierig zu durchschauen. Im *Xerox finite state calculus* ist der sogenannte *Kontextbeschränkungsoperator* (context restriction operator, Koskenniemi, 1983, Abschnitt 2.3.8) definiert um solche Einschränkungen auf einfache Weise zu formulieren. Jedoch ist dieser Operator nur genau für diese Art von Beschränkung konzipiert.

Mithilfe des Prädikats $\in /2$ ist es möglich, einen regulären Ausdruck in eine prädikatenlogische Formel einzubetten und eine bestimmte Sprachzugehörigkeit zu spezifizieren. Auf diese Weise können weitere Prädikate definiert werden, die die Notation der regulären prädikatenlogischen Formeln erweitern. So lässt sich z. B. das einstellige Prädikat $Umlaut/1$ angeben, welches die Eigenschaft bezeichnet, die Teilzeichenkette `ä`, `ü` oder `ö` zu sein, also ein kleingeschriebener Umlaut zu sein. $Umlaut(\alpha)$ ist hierbei definiert durch $\alpha \in$ `ä` $\cup$ `ü` $\cup$ `ö`. Angenommen, es existiert ein einstelliges Prädikat $Letter/1$, das die Eigenschaft bezeichnet, ein Buchstabe zu sein, dann lässt sich über eine reguläre prädikatenlogische Formel formulieren, dass jedes Zeichen ein kleingeschriebener Umlaut ist: $\forall x \; Letter(x) \rightarrow Umlaut(x)$. Diese Formel spe-

zifiziert demnach eine rationale Sprache, die außer den kleingeschriebenen Umlauten keine weiteren Buchstaben enthält. Die so definierten Prädikate werden in Kapitel 4 ausgiebig verwendet, um syntaktische Beschränkungen zu formulieren.

Um innerhalb von regulären prädikatenlogischen Formeln Teilausdrücke zu kapseln, wird von der Ersetzungsschreibweise aus Definition 15 Gebrauch gemacht. Wenn beispielsweise $Final[*]$ durch den Ausdruck $\neg\exists z\ Letter(z) \wedge * \prec z$ definiert ist, dann steht $Final[x]$ für den Ausdruck $\neg\exists z\ Letter(z) \wedge x \prec z$. So spezifiziert die reguläre prädikatenlogische Formel $\exists x\ Umlaut(x) \wedge Final[x]$ eine rationale Sprache, bei der alle Zeichenketten auf einen Umlaut enden.

2.3 Constraint-Erfüllung und -Optimierung

Für viele Aufgaben können Instruktionen vorgegeben werden, um zu einer Lösung zu kommen. Wenn es jedoch schwer ist, den Weg zu einer Lösung anzugeben, aber dahingegen leicht ist, das Ziel zu spezifizieren, sollte eine andere Herangehensweise gewählt werden. Das *Constraint-Erfüllungsproblem* (Constraint Satisfaction Problem, CSP) ist ein Framework, welches genau solche Situationen adressiert, seien es *realitätsnahe Situationen* (real-life scenarios) oder abstrakte mathematische Probleme. Es erlaubt, diese auf rein deklarative Weise anzugeben. Eine wichtige Rolle spielen CSPs in der diskreten Komplexitätstheorie, in der sie prototypische vollständige Probleme für verschiedene Komplexitätsklassen bereitstellen. Dieser Forschungsgegenstand ist jedoch nicht Teil dieser Arbeit. Hier liegt der Fokus auf der Beschreibung und Lösung von Aufgaben oder Problemen, die über wirklichkeitsnahe Situationen gegeben sind und eine starke Flexibilität in ihrer Darstellung benötigen. In diesem Fall ist die Situationsbeschreibung meist eine Vereinfachung der echten Gegebenheiten, da nicht unbedingt alle Faktoren lösungsrelevant sind.

Im Folgenden wird genauer auf das CSP und seine Grenzen in der Beschreibungsfähigkeit eingegangen. Daraufhin wird über den Begriff des weichen Constraints (soft Constraint) das *Semiring-basierte Constraint-Erfüllungsproblem* (Semiring-based Constraint Satisfaction Problem, SCSP, Bistarelli et al., 1995) eingeführt, über welches realitätsnahe Situationen adäquat abgebildet werden können.

2.3.1 Constraint-Erfüllungsproblem

Über das Constraint-Erfüllungsproblem (Constraint Satisfaction Problem, CSP, siehe z. B. Dechter, 2003, Kapitel 2) können viele Situationen durch eine endliche Menge von Variablen zusammen mit einer endlichen Menge von Constraints beschrieben werden. Die Variablen stehen für Objekte, die verschiedenste Werte annehmen können. Constraints hingegen sind Relationen, die Einschränkungen auf die Werte bestimmter Variablen oder Variablenkombinationen fordern. Es gibt zwei Blickwinkel: Entweder spezifiziert ein Constraint mögliche Wertebelegungen oder er spezifiziert unmögliche. Die potenziell unendliche Menge aller Werte, die die Variablen eines Constraint-Problems annehmen können, bilden den Wertebereich. Wenn eine Belegung aller Variablen allen Constraints genügt, wird sie als konsistent bezeichnet, ansonsten, wenn mindestens ein Constraint gebrochen ist, als inkonsistent. Die Constraints können also nur ausschließlich erfüllt oder verletzt werden, daher nennt man sie auch *harte Constraints* (hard constraints). Eine Lösung eines CSPs ist eine Belegung aller Variablen, die konsistent ist. Ein CSP erlaubt es so, Situationen auf deklarative Weise unabhängig vom Lösungsansatz zu beschreiben. Unter Umständen kann eine Situation auf mehrere Arten formuliert werden. Dann können alternative Formulierungen zu einer unterschiedlichen Anzahl von Variablen und Constraints von divergierender Komplexität führen. Somit kann es auch zu Unterschieden in der Laufzeit und dem Speicherbedarf kommen (siehe Barták et al., 2004, Abschnitt 2.2.2).

Für Probleme mit endlichem Wertebereich existieren nur endlich viele Wertebelegungen. In diesem Fall liegt ein endlicher Lösungsraum vor und es wird von einem *endlichen CSP* gesprochen. Dieses besitzt zwar eine geringere Mächtigkeit, ist dafür aber leichter zu handhaben und gleichzeitig in der Lage, viele Situationen zu beschreiben.

Ein CSP, das keine Lösung besitzt, also über die Constraints zu stark eingeschränkt ist, wird als *überbedingt* bezeichnet, ein CSP mit mehr als einer Lösung hingegen als *unterbedingt*. Eine zu große Menge an Lösungen oder die Erkenntnis über die leere Lösungsmenge ist jedoch nicht immer zufriedenstellend. Für die betreffenden Situationen muss hinterfragt werden, ob eine adäquate Beschreibung des echten Problems vorliegt und ob für ihre Darstellung das CSP die geeignete Wahl ist. Dies ist beispielsweise dann der Fall, wenn eine realitätsnahe Situation nicht wirklichkeitsgetreu über harte Constraints abgebildet werden kann, da dem CSP-Formalismus eine gewisse

Flexibilität fehlt (siehe auch Barták, 2002). Die Defizite des CSPs zeigen sich in drei hervorzuhebenden Punkten:

- **Verletzbarkeit:** In vielen realitätsnahen Situationen existieren Regeln, die nicht hart sind und verletzt werden können und eventuell eine Verletzungsschwere mit sich ziehen. Die Umsetzung solcher Regeln über harte Constraints würde zu einem überbedingten CSP führen, ein Ignorieren zu einem unterbedingten. Harte Constraints sind also nur dann ein adäquates Mittel für die Beschreibung von realitätsnahen Situationen, wenn unanfechtbar ist, was verboten und was erlaubt ist.

- **Präferenzen:** In vielen realitätsnahen Situationen spielen Präferenzen eine große Rolle. Diese können z. B. Zufriedenheit, Zurückweisung oder Kosten ausdrücken und können negativ oder positiv sein. Es kann etwas mehr oder weniger präferiert sein als anderes. Harte Constraints kennen nur erfüllbar und unerfüllbar. So können Präferenzen nicht über harte Constraints beschrieben werden und ein Ignorieren würde zu einem unterbedingten CSP führen.

- **Unsicherheit:** Für die Behandlung von realitätsnahen Situationen existiert unter Umständen nur unvollständiges, ungenaues oder fehlerbehaftetes Wissen und trotz resultierender Unsicherheit müssen Entscheidungen getroffen werden. So können manche Situationen nicht perfekt vorhergesagt, aber beispielsweise über Defaultwerte, Default-Strategien oder Wahrscheinlichkeiten behandelt werden. Harte Constraints sind zu starr, um solche flexiblen Mechanismen umzusetzen.

Die syntaktische Analyse geschriebener natürlicher Sprache ist z. B. eine realitätsnahe Situation, die alle der drei oben genannten Punkte umfasst (vgl. Schröder, 2002, Abschnitt 1.1).

2.3.2 Weiche Constraints

Die mangelnde Flexibilität des CSPs ist der Grund für den Übergang von harten zu weichen Constraints, welche mehr als nur zwei Grade der Erfüllbarkeit besitzen (siehe z. B. Barták, 2002). Ein weicher Constraint realisiert eine Funktion, die bestimmten Variablenbelegungen Elemente zuweist, über

die Präferenzen, Verletzungen, Wahrscheinlichkeiten oder Ähnliches beschrieben werden kann. Über die Elemente ist eine partielle oder totale Ordnung definiert, sodass eine Variablenbelegung als 'besser' angesehen werden kann als eine andere. Auf diese Weise können Problemlösungen miteinander verglichen werden. Dann ist der Übergang von harten zu weichen Constraints ein Wechsel von einem Erfüllungsproblem zu einem Optimierungsproblem, bei dem die Aufgabe darin besteht, eine konsistente Variablenbelegung zu finden, die bezüglich der zugewiesenen Elemente die beste Lösung darstellt. Über weiche Constraints können Verletzbarkeit, Präferenzen und Unsicherheit behandelt werden. Jedoch lassen sich nur quantitative und keine qualitativen oder konditionalen Präferenzen abbilden. Es ist zwar möglich, über weiche Constraints die folgende quantitative Präferenz auszudrücken: „Ich mag A mit einer Präferenz von 10". Es ist aber nicht möglich direkt und auf natürliche Weise eine qualitative Präferenz wie „Ich mag A lieber als B" zu beschreiben. Die Ordnung beruht hier auf unabhängigen paarweisen Vergleichen. Um diese umzusetzen, müssten im Gegensatz zur Optimierung von globalen Präferenzwerten Variablenbelegungen paarweise verglichen und unabhängig geordnet werden. Das ist mithilfe von weichen Constraints nicht ohne Weiteres möglich. Dies gilt auch für konditionale Präferenzen wie „Wenn A geschieht, mag ich lieber B als C" (vgl. Rossi et al., 2008). Um qualitative und konditionale Präferenzen zu behandeln, wird in Boutilier et al. (2004) das *konditionale Präferenznetzwerk* (conditional preference network) vorgeschlagen, welches (konditionale) Ceteris-Paribus-Präferenzen umsetzt. In Domshlak et al. (2003) wird gezeigt, dass diese Netzwerke über weiche Constraints approximiert werden können. Die folgende Arbeit beschränkt sich jedoch auf quantitative Präferenzen.

2.3.3 Semiring-basiertes Constraint-Erfüllungsproblem

Für weiche Constraints mit endlichem Wertebereich existieren zwei grundlegende Meta-Frameworks, die zeitgleich eingeführt wurden. Das ist einerseits das *Semiring-basierte Constraint-Erfüllungsproblem* (Semiring-based Constraint Satisfaction Problem, SCSP, Bistarelli et al., 1995) und andererseits das *bewertete Constraint-Erfüllungsproblem* (Valued Constraint Satisfaction Problem, VCSP, Schiex et al., 1995). Beide Frameworks generalisieren endliche CSPs und beruhen auf einer abstrakten algebraischen Struktur, über

welche der Grad der Erfüllbarkeit angegeben und behandelt werden kann. Konkret basiert das SCSP auf einem c-Semiring und das VCSP auf einem linear geordneten und kommutativen Monoiden (bezüglich des VCSP siehe Bistarelli et al., 1999, Abschnitt 3). Über diese beiden Strukturen ist es möglich, verschiedene Framework-Instanzen zu realisieren, die auf Wahrscheinlichkeiten, Präferenzen, Kosten oder auf anderen Maßen beruhen.

Beim VCSP sind die Bewertungselemente den Constraints zugeordnet, beim SCSP hingegen den einzelnen Wertetupeln. Zudem muss beim VCSP der Grad der Erfüllbarkeit linear geordnet sein. Das SCSP kann hingegen auch mit einer partiellen, nichtlinearen Ordnung umgehen. Dies kann z. B. nützlich sein, wenn bestimmte Situationen nur über Multikriterien beschrieben werden können. Dann muss mit einer Pareto-Ordnung umgegangen werden. In der folgenden Arbeit spielen nichtlineare Ordnungen jedoch keine Rolle. Aber dennoch liegt der Fokus auf dem SCSP-Framework, da dort wie auch bei den gewichteten FSMs eine Semiring-Struktur einbezogen wird. Über diese wird es möglich, Instanzen eines SCSPs auf natürliche Weise mithilfe von FSMs zu repräsentieren und zu lösen (siehe Kapitel 3). In Abbildung 2.7 ist der Unterschied zwischen den beiden Frameworks grafisch dargestellt. Eine lineare Ordnung vorausgesetzt lassen sich das SCSP und das VCSP gegenseitig ineinander überführen (siehe Bistarelli et al., 1999, Abschnitt 4).

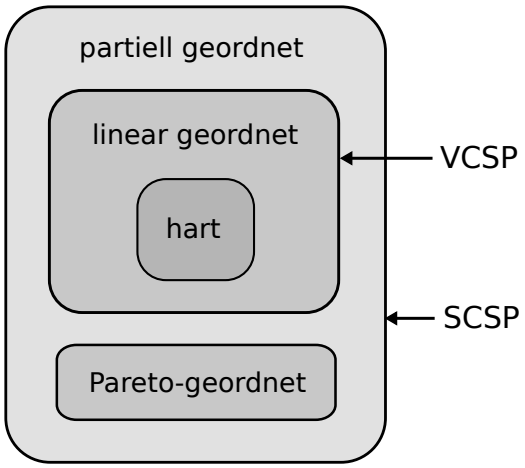


Abbildung 2.7: Der grundlegende Unterschied zwischen dem SCSP und dem VCSP.

Repräsentation eines SCSPs

Grundsätzlich besteht ein SCSP aus drei Teilen: einem Constraint-System, Constraints und einem Constraint-Problem. In diesem Abschnitt wird die Definition dieser drei Bestandteile nach Bistarelli et al. (1995, Abschnitt 3) gegeben und es werden die einzelnen Teile beschrieben.

Constraint-System Ein Constraint-System spezifiziert den zu verwendenden c-Semiring zusammen mit einer Menge von Variablen und ihrem Wertebereich. Jede Instanz eines SCSPs korrespondiert mit der Wahl des Constraint-Systems und somit mit der des c-Semirings:

Definition 18 (Constraint-System)**.** *Ein Constraint-System über einem c-Semiring* $S = (A, \oplus, \otimes, \bar{0}, \bar{1})$ *ist ein Tripel* (S, D, V)*, wobei* V *eine linear geordnete endliche Menge von Variablen und* D *eine endliche Menge von Werten für diese Variablen (der Wertebereich der Variablen) ist.*

Über den c-Semiring erhält ein Constraint-System spezielle Eigenschaften, die für das Lösen eines SCSPs relevant sind und die die Wahl der möglichen Lösungsalgorithmen oder Strategien mitbestimmen.

Constraints Ein Constraint über einem Constraint-System spezifiziert die involvierten Variablen zusammen mit einer Funktion, die allen möglichen Tupeln von Werten für diese Variablen ein c-Semiring-Element zuweist:

Definition 19 (Constraint)**.** *Gegeben ein Constraint-System* $CS = (S, D, V)$ *mit* $S = (A, \oplus, \otimes, \bar{0}, \bar{1})$*, dann ist ein Constraint über* CS *ein Paar* (def, con) *mit* $con \subseteq V$ *und* $def : D^{|con|} \to A$*, wobei con als Typ und def als Wert des Constraints bezeichnet wird.*

Der Typ eines Constraints gibt dessen Skopus an, und der Constraint-Wert realisiert Bedingungen über die involvierten Variablen. Die Menge an Constraints, die über einem Constraint-System $CS = (S,D,V)$ mit $S = (A,\oplus,\otimes,\bar{0},\bar{1})$ formuliert werden können, ergibt sich durch: $\zeta_{CS} = \bigcup_{con \subseteq V} \{(def, con) | def : D^{|con|} \to A\}$. Über die Trennung von Typ und Wert ist es analog zum klassischen CSP möglich, die Struktur eines SCSPs alleine über die Constraint-Typen zu erfassen. Bestimmte Struktur-Eigenschaften können hierbei zu effizienten Lösungsstrategien oder Algorithmen führen. Es ist zu beachten, dass der Wert eines Constraints eine totale Funktion ist und ausnahmslos jedem

Wertetupel ein Element aus S zuweist. Dementsprechend sind die Begriffe Verletzbarkeit und Erfüllbarkeit anders zu interpretieren als bei einem harten Constraint im Rahmen eines klassischen CSPs. Gegeben den Typ con, dann wird im Folgenden ein unerfüllbarer Constraint durch $(\bar{0}, con)$ und ein Universal-Constraint durch $(\bar{1}_{con}, con)$ bezeichnet.

Constraint-Problem Ein Constraint-Problem über einem Constraint-System besitzt eine Menge von Constraints zusammen mit einer Variablenmenge, die als Typ des Constraint-Problems bezeichnet wird:

Definition 20 (Constraint-Problem)**.** *Gegeben ein Constraint-System $CS = (S, D, V)$, dann ist ein Constraint-Problem über CS ein Paar (C, con) mit $con \subseteq V$, wobei C eine endliche Menge von Constraints über CS ist. Hierbei wird angenommen, dass $(def_1, con') \in C$ und $(def_1, con') \in C$ impliziert, dass $def_1 = def_2$.*

Die Menge aller möglichen Constraint-Probleme, die über einem Constraint-System $CS = (S, D, V)$ gebildet werden können, ergibt sich durch: $\mathcal{P}_{\mathcal{CS}} = \{(C, con) | C \subseteq \zeta_{CS}, con \subseteq V(C)\}$. Hierbei bezeichnet ζ_{CS} die Menge aller möglichen Constraints über CS und $V(C) = \bigcup_{def, con' \in C} con'$ den Typ der Constraint-Menge C. Es wird angenommen, dass die Problemvariablenmenge eine Teilmenge von $V(C)$ ist, ansonsten gäbe es kein Problem zu lösen.

Von einem Constraint-Problem wird gefordert, dass keine zwei Constraints mit dem gleichen Typ existieren. Solch eine Einschränkung ist auch bei der Definition von CSPs zu finden (siehe z. B. Dechter, 2003, Abschnitt 2.1.1). Daher wird angenommen, dass Constraints mit dem gleichen Typ zu einem zusammengefasst sind. Angenommen, es existieren zu einem Typ con' mehrere Constraints mit den Werten $def_1, def_2, ..., def_n$, dann können diese zu einem Constraint (con', def) verknüpft werden mit $def(t) = def_1(t) \otimes def_2(t) \otimes ... \otimes def_n(t)$ für alle Tupel t.

Lösen eines SCSPs

Zusammenfassend bedeutet das Lösen eines SCSPs Folgendes: Ausgehend von einem Constraint-Problem werden die c-Semiring-Elemente, die innerhalb der einzelnen Constraints den Tupeln zugewiesen werden, dafür verwendet, die c-Semiring-Elemente der Tupel zu berechnen, die den Typ der gesamten Constraint-Menge abdecken. Hieraus resultiert ein neuer Constraint,

auf dessen Basis der Wert eines Constraints ermittelt wird, der den Typ des Constraint-Problems besitzt und die Lösung des Constraint-Problems darstellt.

Das Lösen eines Constraint-Problems basiert hierbei auf zwei Operationen über Constraints. Das ist einerseits die Kombination ($\otimes$) und andererseits die Projektion ($\Downarrow$). Diese beiden Constraint-Operationen werden dafür verwendet, aus Constraints neue Constraints zu berechnen. Beide Operationen basieren auf der Tupel-Projektion.

Im Folgenden werden zunächst die Tupel-Projektion und darauf aufbauend die beiden Constraint-Operationen Kombination und Projektion definiert. Daraufhin wird über die beiden Constraint-Operationen die Lösung eines Constraint-Problems erläutert und es wird auf die Begriffe *beste Konsistenzstufe* und *abstrakte Lösung* eingegangen. Auch hier wird den Definitionen aus Bistarelli et al. (1995, Abschnitt 3) gefolgt.

Tupel-Projektion Die Tupel-Projektion bildet einen Wertetupel, der für eine bestimmte Variablenmenge definiert ist, auf einen Wertetupel ab, der für eine Teilmenge dieser Menge gilt. Bei dem neuen Tupel sind einige Komponenten des Ursprungstupels entfernt, nämlich genau die, die mit den Variablen korrespondieren, die nicht mehr in der Teilmenge enthalten sind. Die Tupel-Projektion ist demnach wie folgt definiert:

Definition 21 (Tupel-Projektion)**.** *Gegeben ein Constraint-System* $CS = (S, D, V)$*, wobei* V *über* $\preceq$ *vollständig geordnet ist. Betrachtet man nun ein beliebiges k-Tupel* $t = (t_1, t_2, ..., t_k)$ *aus Werten von* D *und zwei Mengen* $W = \{w_1, w_2, ..., w_k\}$ *und* $W' = \{w'_1, w'_2, ..., w'_m\}$ *mit* $W' \subseteq W \subseteq V$*. Des Weiteren sei* $w_i \preceq w_j$*, wenn* $i \leq j$ *und* $w_i \preceq w_j$*, wenn* $i \leq j$*, dann ist die Projektion* $t \downarrow^W_{W'}$ *des Tupels* t *von* W *nach* W' *definiert als der Tupel* $t' = (t'_1, t'_2, ..., t'_m)$ *mit* $t'_i = t_j$*, wenn* $w'_i = w_j$*.*

Kombination Die Kombination ($\otimes$) zweier Constraints ergibt einen neuen Constraint, dessen Typ alle Variablen der Typen der Original-Constraints umfasst. Der Wert des Constraints ist mithilfe der Tupel-Projektion und der abstrakten Multiplikation des c-Semirings definiert:

Definition 22 (Kombination)**.** *Gegeben ein Constraint-System* $CS = (S, D, V)$ *mit* $S = (A, \oplus, \otimes, \bar{0}, \bar{1})$ *und zwei Constraints* $c_1 = (def_1, con_1)$ *und* $c_2 =$

(def_2, con_2) *über* CS*, dann ist die Kombination beider Constraints, die* $c_1 \otimes c_2$ *geschrieben wird, der Constraint* $c = (def, con)$ *mit* $con = con_1 \cup con_2$ *und* $def(t) = def_1(t \downarrow_{con_1}^{con}) \otimes def_2(t \downarrow_{con_2}^{con})$.

Wenn die Kombination-Operation auf eine Menge von Constraints $C = \{c_1, c_2, ..., c_n\}$ angewendet wird, wird $\bigotimes C$ für $c_1 \otimes c_2 \otimes ... \otimes c_n$ geschrieben. Dies ist möglich, da die abstrakte Multiplikation des c-Semirings kommutativ und assoziativ ist. Aus der Definition der Constraint-Kombination wird ersichtlich, dass über ein SCSP keine positiven Präferenzen abgebildet werden können, denn die abstrakte Multiplikation des c-Semirings führt zu Elementen, die in der Ordnung weiter unten oder gleichauf liegen, d.h. $a \otimes b \leq a, b$. In Bistarelli et al. (2005, Abschnitt 4) wird eine Struktur vorgeschlagen, die auch mit positiven Präferenzen umgehen kann, aber keinen Semiring bildet. In der vorliegenden Arbeit wird weiterhin an der c-Semiring-Struktur festgehalten.

Projektion Über die Projektion ($\Downarrow$) kann ein Constraint mithilfe der Tupel-Projektion über eine Menge von Variablen projiziert werden. Hieraus entsteht ein neuer Constraint, der diese Variablenmenge als Typ besitzt. Wenn ein Wertetupel des neuen Constraints über verschiedene Wertetupel aus dem Original-Constraint projiziert werden kann, werden die entsprechenden c-Semiring-Elemente abstrakt addiert:

Definition 23 (Projektion)**.** *Gegeben ein Constraint-System* $CS = (S, D, V)$ *mit* $S = (A, \oplus, \otimes, \bar{0}, \bar{1})$*, ein Constraint* $c = (def, con)$ *über* CS *und eine Menge von Variablen* I *(*$I \subseteq V$*), dann ist die Projektion von* c *über* I*, die als* $c \Downarrow_I$ *geschrieben wird, der Constraint* (def', con') *über* CS *mit* $con' = I \cap con$ *und* $def'(t') = \bigoplus_{\{t | t \downarrow_{con'}^{con} = t'\}} def(t)$.

Lösung Die Lösung eines Constraint-Problems ist ein Constraint, der aus der Anwendung der Constraint-Kombination und Constraint-Projektion entsteht. Im ersten Schritt werden alle Constraints des Constraint-Problems mithilfe der Constraint-Kombination zu einem neuen Constraint verrechnet. Im zweiten Schritt wird der neue Constraint auf den Typ des Constraint-Problems projiziert:

Definition 24 (Lösung)**.** *Gegeben ein Constraint Problem* $P = (C, con)$ *über einem Constraint-System* CS*, dann ist die Lösung von* P *ein Constraint, definiert als* $Sol(P) = (\bigoplus C) \Downarrow_{con}$.

Den einzelnen Lösungen sind c-Semiring-Elemente zugewiesen. Über diese ist die Lösungsmenge partiell geordnet. Hierbei ist zu beachten, dass jedem möglichen Wertetupel für die Problemvariablen ein c-Semiring-Element zugewiesen ist. So ist jeder Tupel eine potenzielle Lösung für das Problem.

Beste Konsistenzstufe Das beste c-Semiring-Element in der Ordnung einer Lösung wird als beste Konsistenzstufe bezeichnet. Dieses Element wird berechnet, indem zuerst alle Constraints eines Constraint-Problems kombiniert werden. Daraufhin wird der resultierende Constraint auf die leere Variablenmenge projiziert. Der Wert des Ergebnis-Constraints stellt dann die beste Konsistenzstufe dar, über die auch der Begriff der Konsistenz definiert ist:

Definition 25 (Beste Konsistenzstufe)**.** *Gegeben ein Constraint-Problem* $P = (C, con)$*, dann ist* $blevel(P) \in S$ *definiert durch* $(blevel(P), \varnothing) = (\bigoplus C) \Downarrow_{\varnothing}$*. Hierbei wird* P *als konsistent bezeichnet, wenn* $\bar{0} <_S blevel(P)$ *und als* α*-konsistent, wenn* $belevel(P) = \alpha$*.*

Wenn allen Lösungen eines Constraint-Problems das Nullelement des c-Semirings zugewiesen wird, kann dieses als inkonsistent bezeichnet werden. Unter dieser Betrachtungsweise kann ein Constraint, der seine Wertetupel ausschließlich auf das Null- oder Einselement des c-Semirings abbildet, als hart interpretiert werden. Ein Constraint-System über den Booleschen Semiring $(\{0,1\}, \vee, \wedge, 0, 1)$ führt demnach zu einer Instanz des SCSPs, die das klassische CSP implementiert.

Abstrakte Lösung Wenn die Lösungen eines Constraint-Problems von Interesse sind, bei denen die zugewiesenen c-Semiring-Elemente nicht 'schlechter' sind als die c-Semiring-Elemente anderer Lösungen, kann die abstrakte Lösung berechnet werden:

Definition 26 (Abstrakte Lösung)**.** *Gegeben ein Constraint Problem* $P = (C, con)$*, dann ist unter der Betrachtung von* $Sol(P) = (def, con)$ *die abstrakte Lösung von* P *die Menge* $ASol(P) = \{(t, v) | def(t) = v$ *und es existiert kein* t'*, sodass* $v \leq_S def(t')\}$*.*

Wenn $\leq_S$ eine totale Ordnung definiert, enthält die abstrakte Lösung ausschließlich Tupel, deren c-Semiring-Elemente gleich der besten Konsistenz-

stufe des Constraint-Problems sind. Wenn ein Constraint-Problem konsistent ist und die abstrakte Lösung mehrere Tupel besitzt, kann es sein, dass c-Semiring-Elemente existieren, die nicht über $\leq_S$ verglichen werden können. In diesem Fall stellt die beste Konsistenzstufe eine oberste Schranke für die zugeordneten c-Semiring-Elemente dar. Wenn $\leq_S$ eine totale Ordnung definiert und das Constraint-Problem konsistent ist und wenn die abstrakte Lösung mehrere Lösungen enthält, kann das Constraint-Problem hingegen als unterbedingt angesehen werden.

In dieser Arbeit kann mit der abstrakten Lösung je nach Kontext auch ein Constraint gemeint sein, was in Bistarelli et al. (1995) nicht der Fall ist, aber hier in späteren Kapiteln notwendig sein wird:

Definition 27 (Abstrakte Lösung als Constraint)**.** *Gegeben die abstrakte Lösung $ASol(P)$ zu dem Constraint-Problem $P = (C, con)$, dann ist der Constraints, der die abstrakte Lösung darstellt, definiert als (def, con) mit*

$$def(t) = \begin{cases} a & falls\ (t,a) \in ASol(P) \\ \bar{0} & sonst \end{cases}$$

Constraint-Propagierung

In Bistarelli et al. (1997, Abschnitt 4) werden lokale Konsistenzregeln vorgeschlagen, um Constraint-Propagierungstechniken auf das SCSP zu verallgemeinern. Sie werden im Verlauf einer Constraint-Propagierung iterativ angewendet, um sukzessive Subprobleme zu lösen und c-Semiring-Elemente zu verändern. Wenn eine Menge von Constraints existiert und wenn über diese ein beliebiges Constraint-Problem definiert ist, das α-konsistent ist, dann gilt für ein β-konsistentes Subproblem über einer Teilmenge dieser Constraints stets $\alpha \leq_S \beta$. Die c-Semiring Element können während der Constraint-Propagierung also nur gleich bleiben oder sich 'verschlechtern'. Das heißt, die Elemente verändern sich immer in Richtung des Nullelements. Ein Ansatz zur lokalen Konsistenz mit iterativer Regelanwendung wurde erstmals in Ugo Montanari (1991) im Kontext des CSPs unter dem Namen *relaxion rule* formalisiert. Im Folgenden wird sich den Definitionen in Bistarelli et al. (1997, Abschnitt 4) bedient.

Konkret hat eine lokale Konsistenzregel die Form $l \leftarrow L$, wobei l eine typisierte Position und L eine Menge von typisierten Positionen ist und $l \notin L$ gilt. Eine typisierte Position bezeichnet hierbei eine Menge von Variablen.

Über diese Positionen können Constraints anhand ihrer Typen identifiziert werden. So ist für ein gegebenes Constraint-Problem der Wert einer typisierten Position ein Constraint des gleichen Typs, falls er existiert. Andernfalls entspricht der Wert einem Universal-Constraint mit entsprechendem Typ.

Die Anwendung einer lokalen Konsistenzregel auf ein Constraint-Problem $P(C, con)$, geschrieben $[l \leftarrow L](P)$, ergibt ein neues Constraint-Problem. Zuerst wird die Lösung für ein Subproblem mit dem Typ l berechnet, welches Constraints aus C beinhaltet, deren Typen in der Menge $L \cup \{l\}$ enthalten sind. Daraufhin wird, in dem Constraint-Problem P der Constraint mit dem Typ l durch den entsprechenden Lösungs-Constraint ersetzt. Wenn solch ein Constraint nicht existiert, wird er der Constraint-Menge hinzugefügt. Das bedeutet, dass die Anwendung von lokalen Konsistenzregeln als eine Funktion von Constraint-Problemen auf Constraint-Probleme angesehen werden kann. So lässt sich eine Sequenz solcher Regeln mit $[r_1; S](P) = [S]([r_1](P))$ auf ein Constraint-Problem anwenden.

Ein *lokaler Konsistenzalgorithmus* (local consistency algorithm) ist die Anwendung einer unendlichen Sequenz von lokalen Konsistenzregeln auf ein Constraint-Problem: $[r_1; r_2; ...](P)$. Die Menge an Regeln ist endlich und jede Regel muss unendlich Mal in der Sequenz vorkommen. Wenn das Constraint-Problem bezüglich der lokalen Konsistenzregeln stabil ist, kann der Algorithmus gestoppt werden. Das bedeutet, dass für jede Regel $[r](P) = P$ gilt. Wenn die abstrakte Multiplikation des c-Semirings idempotent ist, dann ist hier die Eingabe des lokalen Konsistenzalgorithmus gleich der Ausgabe und der Algorithmus terminiert. Wenn die abstrakte Multiplikation hingegen nicht idempotent ist, ist eine Terminierung nicht gewährleistet. Dies kann nur dann garantiert werden, wenn die Ergebnisse der abstrakten c-Semiring-Operationen zusammen mit den c-Semiring-Elementen der Constraints in einer endlichen Menge liegen.

Über den Begriff der lokalen Konsistenz lässt sich der Grad an Explizitheit definieren, den ein Constraint-Problem besitzen kann. Solch ein Grad ist beispielsweise die Kantenkonsistenz. In Bistarelli und Rossi (1998) wird für das SCSP die *Semiring-basierte Kantenkonsistenz* vorgeschlagen. Gegeben ein Constraint-Problem, sodass für jede Variable x ein unärer Constraint c_x und für jedes Paar von Variablen x und y ein binärer Constraint mit c_{xy} existiert. Dann ist das Constraint-Problem kantenkonsistent, wenn für alle beliebigen Variablen x und y gilt, dass $c_x = (\bigotimes\{c_y, c_{xy}, c_x\}) \Downarrow_x$. Für eine effiziente

Propagierung wird vorgeschlagen ein SCSP, dass nicht über den Booleschen c-Semiring definiert ist, auf den Booleschen c-Semiring abzubilden. Hierbei muss die abstrakte Multiplikation des ursprünglichen c-Semirings nicht zwingend idempotent sein. Über das resultierende SCSP kann dann die klassische Kantenkonsistenz berechnet werden. Später kann das SCSP wieder in den ursprünglichen c-Semiring überführt werden. Dieses Verfahren ist insbesondere dann interessant, wenn man ausschließlich an der Eliminierung lokaler Inkonsistenzen interessiert ist. Alternativ ist es aber auch möglich, ein SCSP über einem c-Semiring mit nichtidempotenter abstrakter Multiplikation auf einen nicht Booleschen c-Semiring abzubilden, dessen abstrakte Multiplikation idempotent ist. Dann kann über diesen c-Semiring die Semiring-basierte Kantenkonsistenz berechnet werden. Solch eine Abbildung wird in Bistarelli et al. (2000) als Abstraktion und ein entsprechendes SCSP als abstraktes SCSP bezeichnet.

Perfekte Relaxierung

Constraint-Propagierung überführt ein CSP in ein äquivalentes CSP, welches leichter zu handhaben ist. Über Constraint-Propagierung alleine ist es sogar möglich, ein CSP vollständig zu lösen. In Ugo Montanari (1991) werden im Kontext des CSPs mit endlichem Wertebereich *Relaxierungsregeln* (relaxion rules) vorgeschlagen, um innerhalb einer *perfekten Relaxierung* (perfect Relaxation) die Lösung eines Problems zu berechnen. Hierbei wird jede Regel genau ein Mal angewendet und die Anwendung folgt einer *perfekten Strategie* (perfect strategy), die eine bestimmte vorgegebene Regelanwendungsabfolge darstellt, die sich an einer gegebenen Problemzerlegung orientiert. Solch eine perfekte Relaxierung ist beispielsweise dann möglich, wenn eine baumartige Zerlegung des Problems vorliegt. In Bistarelli et al. (1997) wird die perfekte Relaxierung auf das SCSP-Framework übertragen. Es werden lokale Konsistenzregeln herangezogen, die nacheinander angewendet werden, um ein baumartiges Bottom-Up-Lösen eines Problems mit baumartiger Zerlegung zu ermöglichen. Im Folgenden wird sich den Definitionen aus Bistarelli et al. (1997, Abschnitt 5) bedient.

Für ein SCSP mit gegebener Baumzerlegung wird ein *Parse-Baum* (parsing tree) angenommen, der durch eine Sequenz von lokalen Konsistenzregeln definiert ist und bestimmten Regularien unterliegt:

Definition 28 (Parse-Baum). *Gegeben ein SCSP $P = (C, con)$, dann ist ein Parse-Baum von P eine Sequenz von lokalen Konsistenzregeln $S = r_1; r_2; ...; r_n$, wobei $r_i = (l_i \leftarrow L_i)$. Sei*

- $L = \{l_1, i = 1, \ldots, n\} \cup \bigcup_{i=1,\ldots,n} L_1$ *(L ist die Menge aller Positionen innerhalb der Regeln von S)*
- $l_1 \prec l$ *gdw.* $l \in L_i$ *(Die Regel r_i generiert die Position l_i);*
- $l_i \prec v$ *gdw.* $v \notin l_i$ *und* $v \in l'$ *mit* $l' \in L_i$ *(Die Regel r_i generiert die Variable v_i);*
- *für jede Position oder Variable x:* $prec(x) = \{l | l \prec x\}$.

Dann wird gefordert, dass

- *für jede Position oder Variable x gilt, dass wenn $x = l_n$ oder $x \in l_n$, dann ist* $prec(x) = \varnothing$, *ansonsten ist* $prec(x)$ *eine einelementige Menge (baumartig);*
- $con = l_n$ *und* $(def, con') \in C$ *impliziert, dass* $con' \in L$ *(abdeckend);*
- $l_i \prec l_j$ *impliziert, dass* $j < i$ *(bottom-up).*

Solch ein Parse-Baum kann als eine perfekte Strategie angesehen werden. Jede Regel tritt in der Sequenz genau ein Mal auf, die Regeln umfassen das gesamte Problem und sie sind baumartig organisiert. Des Weiteren gibt die Regelsequenz eine Bottom-Up-Durchwanderung der baumartigen Problemzerlegung vor. Der Lösungsalgorithmus für einen Parse-Baum ist analog zu einem lokalen Konsistenzalgorithmus definiert. Gegeben ein Constraint-Problem $P(C, con)$ und einen Parse-Baum $S = r_1; r_2; \ldots; r_n$, dann besteht der Lösungsalgorithmus aus der Berechnung von $[S](P)$, wobei die Lösung von P dem Constraint mit dem Typ *con* entspricht. Im Folgenden wird der Begriff perfekte Relaxierung verwendet, wenn ein SCSP anhand eines Parse-Baums gelöst wird.

2.4 Dependenzrelationen und -strukturen

Innerhalb einer Dependenzstruktur werden keine phrasalen Einheiten angenommen, sondern Dependenzrelationen (siehe Mel'čuk, 1988). Über diese werden Wörter anderen Wörtern untergeordnet. Die Dependenzrelationen sind anti-symmetrisch, anti-reflexiv und anti-transitiv, sodass eine Unterordnung nur direkt in eine Richtung erlaubt ist. In dieser Arbeit wird das übergeordnete Wort als *Governor* und das untergeordnete als *Modifier* bezeichnet. In der Literatur existieren alternative Bezeichnungen. So wird dort das übergeordnete Wort beispielsweise *Regent* oder *Kopf* und das untergeordnete auch *Dependent* genannt. Es können zudem Dependenz-Label eingebunden werden, die die Art der Unterordnungsbeziehungen angeben. Wenn ein Wort keinem anderen untergeordnet ist, wird es als *unabhängig* bezeichnet und es kann für einen Satz ein *Dummy-Wurzelwort* angenommen werden, dass unabhängig und nicht Teil des Satzes ist.

Eine Dependenzstruktur kann als ein (gelabelter) gerichteter Graph dargestellt werden (siehe Mel'čuk, 1988, Abschnitt 1. III.). Hierbei werden die Wörter des Satzes und das Dummy-Wurzelwort über die Knoten repräsentiert und die Dependenzrelationen zwischen den Wörtern über die (gelabelten) Kanten. Hierbei zeigt eine Kante stets von einem Governor zu seinem Modifier. Des Weiteren werden in dieser Arbeit Beschränkungen für eine Dependenzstruktur angenommen, die im Rahmen des Dependenzparsing üblich sind. So müssen Dependenzstrukturen zusammenhängend und azyklisch sein und sie müssen genau einen ausgezeichneten Wurzelknoten besitzen. In diesem Fall wird von einem gewurzelten Dependenzbaum gesprochen.

In Abbildung 2.8 ist ein Dependenzbaum für den Satz „Die Tiere hat er im Bärengraben gesehen." dargestellt. Die schwarzen Linien geben die Kanten an, die über die Label `S`, `DET`, `AUX`, `OBJA`, `SUBJ`, `PP`, `PN` und über das leere Label ausgezeichnet sind. Die Bedeutung der Dependenz-Label kann der Tabelle 5.1 aus Abschnitt 5.3.1 entnommen werden. Der Governor einer Kante ist als schwarzer Punkt stets höher eingezeichnet als sein Modifier. Des Weiteren sind für die Kanten, die an das Dummy-Wurzelwort angebunden sind, die schwarzen Linien gerade nach oben gezogen. Die grauen senkrechten Linien geben die Projektionen der Wörter an. Aus Gründen der Lesbarkeit sind für die Terminalknoten keine Projektionslinien eingezeichnet. Der Baum ist nicht projektiv, da die `SUBJ`-Kante zwei Linien kreuzt, wobei man sich hier

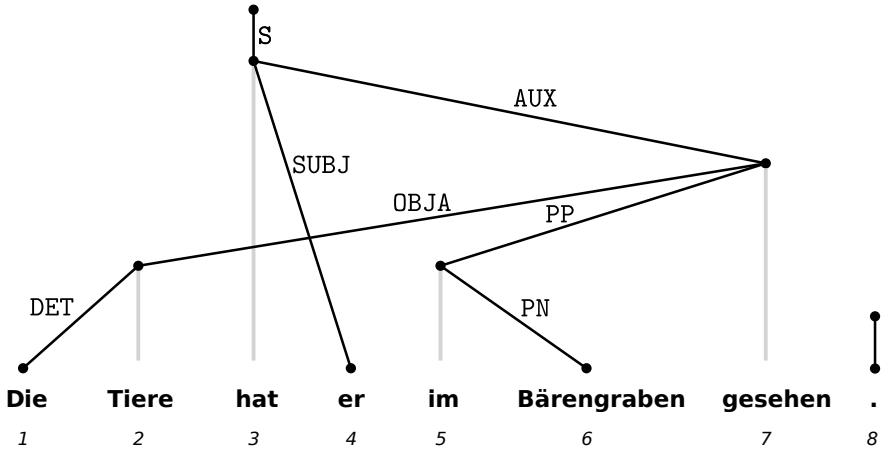


Abbildung 2.8: Dependenzbaum.

bei den Terminalknoten eine Linie hinzudenken muss.

Beim Dependenzparsing wird die Projektivität oft als grundlegende Beschränkung angenommen. Bei projektiven Dependenzbäumen kommt es zu keinen Überkreuzungen und sie können in Phrasenstrukturbäume mit kontinuierlichen Konstituenten übersetzt werden (siehe Gaifman, 1965, Abschnitt III). In der vorliegenden Arbeit wird angenommen, dass auch Satzzeichen in einer Dependenzstruktur involviert sein können. So werden diese im Folgenden auch als Wörter bezeichnet.

Kapitel 3

Constraint-Erfüllung und -Optimierung mithilfe von FSMs

In diesem Kapitel wird beschrieben, wie ein SCSP mithilfe von FSMs repräsentiert und gelöst werden kann, und es werden Erweiterungen des SCSPs vorgestellt, die über FSMs realisiert werden können. Der vorgestellte Ansatz basiert auf der Veröffentlichung Didakowski (2010, Abschnitt 3 und 4), die im Rahmen der vorliegenden Arbeit entstand. Es ist die erste Forschungsarbeit, die die Umsetzung eines SCSPs mit FSMs thematisiert. Auch für konkrete CSP-Erweiterungen wie gewichtete CSPs, fuzzy CSPs oder probabilistische CSPs existieren keine vergleichbaren Publikationen.

In Abschnitt 3.1 werden Arbeiten vorgestellt, die sich mit der Umsetzung von CSPs, Constraints, Wertetupeln oder Wertebereichen mithilfe von FSMs beschäftigen. Anschließend wird in den Abschnitten 3.2 und 3.3 gezeigt, wie das SCSP über FSMs repräsentiert und gelöst werden kann. Abschließend werden in Abschnitt 3.4 Erweiterungen des SCSP-Frameworks vorgestellt: ein abzählbarer unendlicher Wertebereich, die simultane Repräsentation von Constraint-Systemen und -Problemen und quantifizierte reguläre Constraints.

3.1 Verwandte Arbeiten

Arbeiten, die sich mit der Umsetzung von CSPs, Constraints, Wertetupeln oder Wertebereichen mithilfe von FSMs beschäftigen, lassen sich in drei Gruppen einteilen, die in den folgenden Abschnitten nacheinander vorgestellt

werden. Die erste basiert auf dem Ansatz von Vempaty (1992), bei dem FSAs vorgeschlagen werden, um endliche CSPs zu repräsentieren und zu lösen. Die zweite beruht auf der Arbeit von van Hoeve et al. (2004), bei dem globale Constraints mithilfe von FSAs implementiert werden. Die dritte befasst sich mit der Umsetzung eines unendlichen Wertebereichs über FSAs.

3.1.1 CSP mit endlichem Wertebereich

In Vempaty (1992) wird gezeigt, wie CSPs mithilfe von FSAs repräsentiert und gelöst werden können, wobei vorausgesetzt wird, dass der Wertebereich endlich ist. Dies ist die erste Arbeit, die die Repräsentation und Lösung eines CSP mit FSMs thematisiert. Um ein CSP über FSAs repräsentieren zu können, wird angenommen, dass eine lineare Ordnung über den Variablen definiert ist und dass sich alle Variablen im Skopus eines Constraints befinden und dass die Abfolge der Komponenten der Wertetupel der Variablenordnung folgt.

Ein Wertetupel wird als Zeichenkette über dem Alphabet des CSPs repräsentiert, deren Länge durch die Größe der Variablenmenge bestimmt ist. Ein FSA erkennt genau die Zeichenketten, die als Wertetupel den Constraint erfüllen. In dieser Repräsentationsform wird der unerfüllbare Constraint durch einen FSA umgesetzt, der die leere Sprache erkennt. Constraints sind somit ausschließlich über ihre Wertetupel definiert.

Ein Beispieldiagramm für einen FSA, der einen Constraint repräsentiert, ist in Abbildung 3.1 in Form eines generalisierten Übergangsdiagramms gegeben. Das Beispiel bezieht sich auf ein CSP mit der Variablenmenge

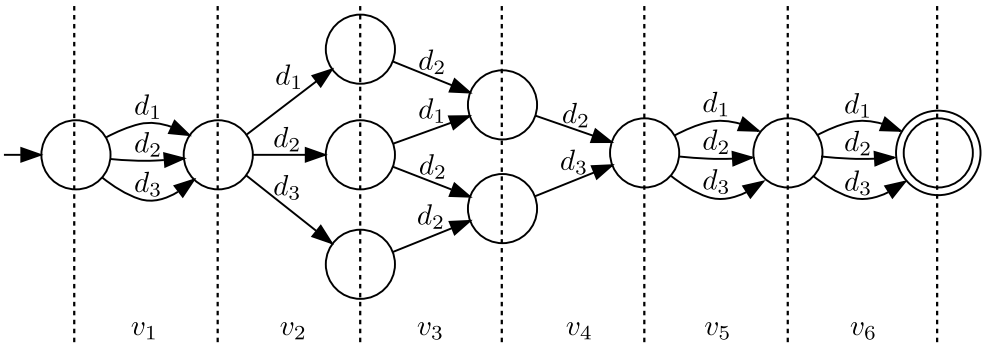


Abbildung 3.1: Constraint als FSA.

$V = \{v_1, v_2, v_3, v_4, v_5, v_6\}$ mit der Ordnung $v_1 \prec v_2 \prec v_3 \prec v_4 \prec v_5 \prec v_6$ und

dem Wertebereich $D = \{d_1, d_2, d_3\}$. Das Diagramm ist in mehrere Ebenen unterteilt, die durch vertikale gestrichelte Linien angezeigt werden. Die Wertetupelkomponenten sind über die Ebenen den einzelnen Variablen zugeordnet. Dieser FSA akzeptiert 36 verschiedene Wertetupel, wie z. B. den Tupel '$(d_1, d_1, d_2, d_2, d_1, d_1)$', der in seiner Zeichenkettenrepräsentation die Form '$d_1d_1d_2d_2d_1d_1$' besitzt. Im Folgenden wird keine strikte Unterscheidung zwischen den Wertetupeln und ihrer Repräsentation auf Zeichenkettenebene gemacht. So kann gesagt werden, dass der FSA den Wertetupel '$(d_1, d_1, d_2, d_2, d_1, d_1)$' erkennt oder generiert. Da hier der Skopus eines Constraints stets alle Variablen umfasst, kann in Abbildung 3.1 der zugrunde liegende Constraint als ein Constraint über die Variablen v_2, v_3 und v_4 interpretiert werden. Dieser wurde dann um die Variablen v_1, v_5 und v_6 erweitert.

Es werden die rationalen Automatenoperationen Vereinigung und Schnitt vorgeschlagen, um die logischen Operationen 'ODER' und 'UND' umzusetzen. So kann über die Vereinigung (ODER) aus zwei FSA-Constraints ein FSA konstruiert werden, der genau die Wertetupel erkennt, die entweder den einen oder den anderen Constraint erfüllen. Über die Schnittoperation (UND) ist es hingegen möglich, aus zwei FSA-Constraints einen FSA zu erstellen, der exakt die Wertetupel akzeptiert, die beiden Constraints genügen. Des Weiteren wird vorgeschlagen, die logische Operation 'DIFF' über das relative Komplement zweier FSAs umzusetzen. Die Lösung eines CSP ist dann die Verrechnung aller FSAs mithilfe dieser Automatenoperationen zu einem FSA, der die CSP-Lösung repräsentiert. Bei diesem Ansatz werden alle Wertebelegungen berechnet, die keinen Constraint verletzen. Wenn der Lösungs-FSA die leere Sprache akzeptiert, also wenn die Lösung ein unerfüllbarer Constraint ist, ist das CSP inkonsistent. Die Effizienz des Ansatzes hängt stark von der Größe der FSAs ab, die während des Problemlösens entstehen. Um die Zwischenergebnisse klein zu halten, werden minimale deterministische Maschinen herangezogen. So wird nach jeder Automatenoperation, wenn notwendig, determinisiert und minimiert. Allgemein gilt für diesen Ansatz, dass eine Lösung effizient berechnet werden kann, wenn es möglich ist, die Wertetupel während des Problemlösens kompakt zu repräsentieren.

Amilhastre et al. (2001) knüpfen an den Ansatz von Vempaty (1992) an und schlagen nichtdeterministische FSAs vor, um die Wertetupel eines Constraints kompakt zu repräsentieren. Die Idee dahinter ist, dass ein nichtdeterministischer FSA exponentiell weniger Zustände besitzen kann als ein

deterministischer (vgl. Rabin und Scott, 1959, Abschnitt 2). Es wird ein Minimierungsalgorithmus vorgestellt, der aus einem FSA-Constraint einen minimalen FSA konstruiert. Der Minimierungsbegriff bezieht sich hierbei auf FSAs im Allgemeinen und nicht nur auf die deterministischen Varianten. Ein FSA ist in diesem Sinne minimal, wenn es keinen anderen FSA mit weniger Zuständen gibt, der das äquivalente Verhalten aufweist. Für eine effiziente Berechnung wird ausgenutzt, dass ein FSA-Constraint eine endliche rationale Sprache erkennt, die aus Wörtern gleicher Länge besteht.

In Amilhastre et al. (2002) wird die Repräsentationsform, wie sie von Vempaty (1992) vorgeschlagen wird, innerhalb eines Ansatzes zur interaktiven Spezifikation und Lösung eines CSPs verwendet. Der Ansatz geht davon aus, dass die harten Constraints eines CSPs, wie sie in Vempaty (1992) beschrieben sind, vorweg offline zu einem Constraint zusammengefasst sind. Das CSP wird also kompiliert. Zu diesem werden dann online dynamisch weiche unäre Constraints hinzugefügt, die einen bestimmten Relevanzgrad besitzen und einzelne Benutzerspezifikationen repräsentieren. Hierzu wird das kompilierte CSP in einen WFSA überführt und es wird über die Gewichte Verletzbarkeit implementiert, wobei eine Verletzung eine gewisse Schwere besitzen kann. Innerhalb des Ansatzes kann ermittelt werden, ob eine gegebene Spezifikation einen der dynamisch ergänzten Constraints bricht. Es können die Lösungen berechnet werden, die keinen weichen Constraint verletzen oder die den weichen Constraints am meisten genügen. Es kann aber auch die Constraint-Menge bestimmt werden, die ohne Verletzungen beibehalten werden kann, wenn ein bestimmter weicher Constraint nicht gebrochen werden soll.

Hadzic et al. (2008) beschäftigen sich mit der Kompilierung von CSPs und vergleichen minimale deterministische FSAs, *Multi-Valued-Decission-Diagramme* (MDD) und *binäre Decission-Diagramme* (BDD) miteinander. Für die FSA-Repräsentation ist die Form gewählt, wie sie in Vempaty (1992) vorgeschlagen wird. Diese entspricht demnach der Repräsentation eines kompilierten CSPs in Amilhastre et al. (2002). Die drei oben genannten Repräsentationsformen teilen die Struktur eines azyklischen gerichteten Graphen. Es wird gezeigt, dass ein nicht reduziertes MDD, bei dem isomorphe Knoten zusammengelegt sind, strukturell identisch zu einem minimalen deterministischen FSA ist, der dieselbe Lösungsmenge repräsentiert. Des Weiteren wird demonstriert, dass BDDs, eine entsprechende Variablenordnung vorausgesetzt, die gleiche zugrunde liegende Struktur Teilen wie MDDs oder

minimale deterministische FSAs. Es wird aber auch gezeigt, dass reduzierte MDDs, eine etwas kompaktere Repräsentation ermöglichen als minimale deterministische FSAs. Reduzierte MDDs sind hierbei MDDs, bei denen isomorphe Knoten zusammengelegt und zusätzlich redundante Knoten entfernt sind.

3.1.2 CSP mit unendlichem Wertebereich

In Golden und Pang (2006) steht ein CSP im Fokus, bei dem die Wertebereiche der Variablen aus Zeichenketten bestehen und unendlich sein können. Es wird vorgeschlagen, die Wertebereiche über FSAs zu repräsentieren, um Constraints über einen unendlichen Wertebereich realisieren zu können.

Innerhalb des Ansatzes können verschiedene Arten von regulären *Zeichenketten-Constraints* (string constraints) formuliert werden: Es kann über einstellige Constraints ausgedrückt werden, dass jeder Wert zu einer Variable gegen einen gegebenen regulären Ausdruck matcht. Es kann eine untere oder obere Schranke für die Zeichenkettenlänge angegeben werden und es kann formuliert werden, dass der Wert einer Variable an einer bestimmten Position ein bestimmtes Zeichen besitzen muss. Des Weiteren können zweistellige Constraints ausgedrückt werden, über die beispielsweise gefordert werden kann, dass der Wert einer Variable innerhalb des Wertes einer anderen Variable auftritt. Es können aber auch n-stellige Constraint formuliert werden, die besagen, dass der Wert einer Variable das Ergebnis der Konkatenation der Werte bestimmter Variablen sein muss.

Ein exhaustives Auflisten eines unendlichen Wertebereichs ist nicht möglich, was eine Umsetzung von Suchalgorithmen schwierig macht. Es wird daher vorgeschlagen, initial generalisierte Kantenkonsistenz herzustellen, um die Wertebereiche auf eine endliche Menge herunterzubrechen. Um einen entsprechenden Algorithmus umzusetzen, sind für die Wertebereiche über rationale Operationen die Prüfung auf Gleichheit und auf eine Teilmengenrelation implementiert. Um den Zustand des Wertebereichs einer Variable zu prüfen, kann dessen Größe berechnet werden, die entweder null, eins oder unendlich sein kann oder einer anderen konkreten Größe entspricht. Besitzt der Wertebereich einer Variable die Größe null, ist das CSP inkonsistent. Wenn hingegen alle Variablen einen Wertebereich der Größe eins besitzen, ist das CSP gelöst.

Hansen und Andersen (2007) beschäftigen sich mit der interaktiven Spezifikation und Lösung eines CSPs mit einem unendlichen Wertebereich, der aus Zeichenketten besteht. Es wird ein einstelliger regulärer Zeichenketten-Constraint vorgeschlagen, der prüft, ob die Werte einer Variable gegen einen regulären Ausdruck matchen. Solche Constraints können mit logischen Operationen wie UND, ODER und NICHT kombiniert werden, die dort als *logische Constraints* bezeichnet werden. Es wird vorgeschlagen, die Zeichenketten-Constraints und die logischen Constraints in einen FSA zu kompilieren, über den geprüft werden kann, ob eine Wertebelegung, die durch eine bestimmte interaktive Spezifikation gegeben ist, zu der Lösung eines CSPs gehört. Es wird jedoch nicht die Repräsentationsform gewählt, wie sie in Vempaty (1992) vorgeschlagen wird. Dies ist ohnehin nicht möglich, da mit einem unendlichen Wertebereich und somit mit unendlichen rationalen Sprachen umgegangen werden muss. So ist ein Mapping von den Werten zu den Variablen über Zeichenkettenpositionen nicht mehr möglich. Es wird dahingegen vorgeschlagen, die Zuordnung über ein komplexes Alphabet Σ_C zu realisieren, das aus Tupeln der Form $\Sigma_C \subset (\Sigma \cup \{\epsilon\})^n$ besteht, wenn n Variablen betrachtet werden. Zudem wird angenommen, dass ein Tupel nur eine Komponente ungleich ϵ beinhaltet. Eine lineare Variablenordnung vorausgesetzt, sind die einzelnen Tupel-Komponenten hierüber bestimmten Variablen zugeordnet: Eine Zeichenkette, die aus der Konkatenation der Zeichen der k-ten Komponenten einer Tupelabfolge entsteht, ist ein Wert für die Variable, die mit der k-ten Komponentenposition assoziiert ist.

Es soll ein Beispiel gegeben werden: Wenn ein CSP mit der Variablenmenge $\{x_1, x_2\}$ mit der Ordnung $x_1 \prec x_2$ angenommen wird und ein Alphabet $\Sigma = \{a, b\}$ existiert, dann ist $\Sigma_C = \{(a, \epsilon), (b, \epsilon), (\epsilon, a), (\epsilon, b)\}$. So führt ein Wort $(a, \epsilon)(\epsilon, b)(a, \epsilon)$ zu der Zuordnung 'aa' zu x_1 und 'b' zu x_2. Zu der gleichen Paarung führen aber auch die Wörter $(b, \epsilon)(\epsilon, a)(a, \epsilon)$ und $(a, \epsilon)(\epsilon, a)(b, \epsilon)$. Über diese Art der Zuordnung ist es möglich, einen unendlichen Wertebereich eines CSP innerhalb einer FSA-Repräsentation zu behandeln. Jedoch wird dort angemerkt, dass diese Repräsentationsform für die Kompilierung eines CSP mit Zeichenketten-Constraints und logischen Constraints in einen FSA nicht praktikabel ist, da der FSA schnell zu groß und unhandhabbar wird. Grund hierfür ist, dass mehrere Wörter dieselbe Variablenbelegung repräsentieren können, je nachdem, wie die Tupel-Zeichen arrangiert sind. Für solch eine Variabilität in der Abfolge ist ein FSA nicht

die geeignete Repräsentationsform. Es wird daher vorgeschlagen, die einstelligen Zeichenketten-Constraints in eine Maschine zu überführen, die ähnlich zu einer Moore-Maschine eine Ausgabe über die Zustände generiert (vgl. John E. Hopcroft, 1979, Abschnitt 2.7). Über die Ausgabe kann zurückverfolgt werden, welche von den zugrunde liegenden Zeichenketten-Constraints erfüllt sind. Diese Repräsentationsform wird dann mit einem BDD verknüpft, welches die logischen Constraints implementiert. Der Ansatz ermöglicht es, Variablen mit unendlichem Wertebereich mit Variablen zu kombinieren, die einen endlichen Wertebereich besitzen, der nicht unbedingt aus Zeichenketten bestehen muss. In Hansen und Tiedemann (2008) wird eine Weiterentwicklung der logischen Constraints vorgestellt, der auf eine Steigerung der Zeiteffizienz abzielt.

3.1.3 Globale Constraints

Globale Constraints repräsentieren Substrukturen, die in vielen verschiedenen Problemen auftreten können und sie stellen für diese Strukturen in der Regel effiziente Propagierungsalgorithmen bereit. Typischerweise sind diese Constraints für jeden Skopus wohldefiniert (siehe z. B. Dechter, 2003, Abschnitt 3.5.2).

In Pesant (2004) wird der *Regular-Membership-Constraint* als ein spezieller Typ eines globalen Constraints eingeführt. Gegeben eine Sequenz von Variablen und einen deterministischen FSA, dann besagt der Regular-Membership-Constraint, dass jede Variablenbelegung von dem gegebenen FSA akzeptiert werden muss. Hierzu werden die Wertesequenzen wie in Vempaty (1992) als Zeichenketten repräsentiert, wobei das Alphabet dem Wertebereich des CSPs entspricht. Ein Beispiel für solch einen FSA ist in Abbildung 3.2 auf der linken Seite für den Wertebereich $\{d_1, d_2, d_3\}$ gegeben. Das Verhalten des FSAs ist eine rationale Sprache, die das Auftreten des Wertes d_3 folgendermaßen einschränkt: Der Wert d_3 muss von dem Wert d_2 gefolgt sein. So ist bei einer Sequenz der vier Variablen (x_1, x_2, x_3, x_4) der Wertetupel (d_1, d_3, d_2, d_1) gültig, aber der Wertetupel (d_1, d_2, d_3, d_1) hingegen nicht.

Für den Regular-Membership-Constraint wird ein Propagierungsalgorithmus vorgestellt, der *generalisierte Kantenkonsistenz* (generalized arc-consistency)[1] herstellt und eine effiziente Umsetzung der Kantenkonsistenzwieder-

[1]Die generalisierte Kantenkonsistenz bezeichnet die Erweiterung der Kantenkonsistenz

herstellung ermöglicht, wenn beispielsweise Variablenwerte im Verlauf einer Constraint-Propagierung entfernt wurden. Als Grundlage für den Algorithmus wird im ersten Schritt anhand eines Regular-Membership-Constraints ein azyklischer geschichteter gerichteter Multi-Graph erstellt. Dieser Graph korrespondiert mit einem azyklischen FSA, der die Wertetupel akzeptiert, die auch von dem FSA des Regular-Membership-Constraints erkannt werden. Diese Repräsentationsform ist der eines Constraint in Vempaty (1992) sehr ähnlich, jedoch sind hier nicht zwingend alle Variablen des CSPs involviert. Die Erstellung solch eines Graphen aus einem gegebenen FSA wird als Entfaltung und der entsprechende azyklische geschichtete gerichtete Multi-Graph als entfalteter FSA bezeichnet. In Abbildung 3.2 ist zur Verdeutlichung auf der linken Seite ein nichtentfalteter und auf der rechten Seite ein korrespondierender entfalteter FSA abgebildet. Die Ecken des entfalteten FSAs sind in

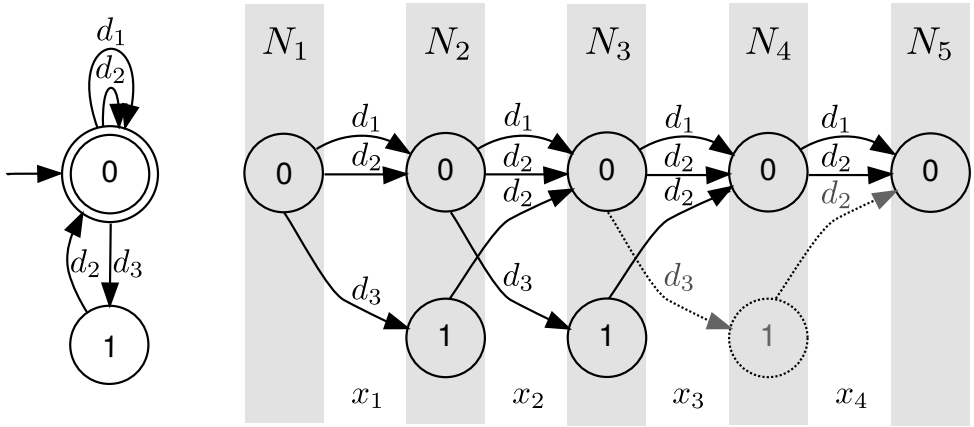


Abbildung 3.2: Ein nichtentfalteter FSA (links) und ein korrespondierender entfalteter FSA (rechts).

die Schichten $N_1, ..., N_5$ eingeteilt. Die Beschriftung der Ecken orientiert sich an der Zustandsbenennung des nichtentfalteten FSAs. Die Kanten zwischen den Schichten sind den einzelnen Variablen zugeordnet. Die grundlegende Idee ist hierbei, dass ein Wert zu einer Variable durch eine Kante gestützt

für binäre Constraints auf die Kantenkonsistenz für n-stellige Constraints. Diese wurde von Mackworth, 1977 eingeführt. Ein Constraint mit dem Skopus $v_1, .., v_k$ wird als generalisiert kantenkonsistent (oder auch als wertebereichkonsistent) bezeichnet, wenn für jedes i mit $1 \leq i \leq k$ und für jeden Wert a aus dem Wertebereich des CSPs ein Wertetupel $(d_1, .., d_k) \in C$ existiert, sodass $d_i = a$. Ein CSP ist generalisiert kantenkonsistent, wenn all seine Constraints generalisiert kantenkonsistent sind.

wird, wenn ein entsprechender Pfad im Graphen existiert. In dem entfalteten FSA in Abbildung 3.2 gilt dies initial für alle Variablenwerte. Er kann nun aktualisiert werden, wenn bestimmte Werte für eine Variable nicht mehr möglich sind. Dies ist hier durch die gestrichelten Kanten angezeigt, die den Variablen x_3 und x_4 zugeordnet sind. Diese wurden entfernt, nachdem der Wert d_2 für die Variable x_4 während einer fortschreitenden Constraint-Propagierung ausgeschlossen wurde. Für die Variable x_3 wird hierdurch der Wert d_3 nicht mehr durch einen Pfad im Graphen gestützt. Es wird angemerkt, dass die Verwendung eines minimalen deterministischen FSA als Eingabe des Propagierungsalgorithmus zu präferieren ist. Hierdurch können redundante Ecken im Graphen vermieden und der Graph effizienter gehandhabt werden.

In Richaud et al. (2006) wird vorgeschlagen, sogenannte *Nogoods* kompakt über einen FSA zu repräsentieren. Ein Nogood ist eine Menge von Wertezuweisungen für eine bestimmte Menge von Variablen, welche zu keiner Lösung eines CSP führen. Nogoods werden im Allgemeinen dazu verwendet, Suchalgorithmen zu beschleunigen. Die FSA-Repräsentation ähnelt in diesem Ansatz der Repräsentationsform eines Constraint in Vempaty (1992). Der einzige Unterschied ist, dass hier nicht zwingend alle Variablen des CSPs involviert sein müssen. Ein entsprechender FSA erkennt genau die Wertetupel zu einer gegebenen Menge von Variablen, die keine Nogoods darstellen. Wenn dem FSA ein neuer Nogood hinzugefügt wird, wird das relative Komplement aus den bestehenden FSA und dem neuen Nogood berechnet und das Ergebnis in einen minimalen deterministischen FSA überführt. Es wird vorgeschlagen, den Propagierungsalgorithmus von Pesant (2004) anzuwenden, um generalisierte Kantenkonsistenz bezüglich der Nogoods herzustellen.

In van Hoeve et al. (2004) wird im Kontext von überbedingten CSPs der *Soft-Regular-Membership-Constraint* eingeführt. Dieser ist eine Erweiterung des Regular-Membership-Constraint und ermöglicht zusätzlich eine Relaxierung. Der Relaxierungsgrad wird über ein Distanzmaß wie der *Hamming-Distanz* oder der *Levenshtein-Distanz* (Editierabstand) angegeben. Innerhalb dieses Ansatzes werden einem bestehenden entfalteten FSA zusätzliche Kanten hinzugefügt, die bestimmte Kosten verursachen. Hieraus entsteht ein bewerteter entfalteter FSA, der je nach Distanzmaß zyklisch oder azyklisch sein kann.[2] Der Grad einer minimalen Relaxierung kann über kürzeste Dis-

[2]Dieses Vorgehen hat Ähnlichkeit zur Variantengenerierung bei der Berechnung des Editierabstandes zwischen zwei rationalen Sprachen mithilfe von WFSAs (siehe Mohri,

tanzen berechnet werden, und es kann eine obere Schranke für die Relaxierungskosten angegeben werden. Der Propagierungsalgorithmus des Regular-Membership-Constraints ist auch für die weiche Variante erweitert. So stützt ein Pfad nur dann einen Wert, wenn er bestimmte Relaxierungskosten nicht übersteigt.

In Demassey et al. (2006) wird eine weiche Variante des Regular-Membership-Constraint vorgestellt, die *Cost-Relular* (cost-regular) genannt wird. Dieser Constraint berücksichtigt zusätzlich eine Kostenmatrix, die den möglichen Variablenwerten und somit den Kanten eines entfalteten FSAs bestimmte Kosten zuordnet. Für die Kosten eines Pfades kann eine untere und obere Schranke angegeben werden, die über die Berechnung kürzester und längster Pfade geprüft werden können. Der Propagierungsalgorithmus des Regular-Membership-Constraint ist für Cost-Regular entsprechend erweitert. Die obere und untere Schranke für die Kosten werden dazu verwendet, den Lösungsraum früh zu beschneiden. Zudem wird vorgeschlagen, die Kosteninformation als Heuristik während des Problemlösens heranzuziehen, um einen Backtracking-Lösungsalgorithmus zu leiten.

In Menana und Demassey (2009) wird *Multicost-Regular* (multicost-regular) als eine Generalisierung von Cost-Regular vorgestellt. Multicost-Regular ermöglicht es, verschiedene kumulative Kosten simultan zu behandeln. Das Prüfen der untersten Schranke ist nun ein Resource-Constrained-Shortest-Path-Problem, das ähnlich zu einem Multi-Criteria-Shortest-Path-Problem ist. Das Testen der obersten Schranke ist hingegen ein Resource-Constrained-Longest-Path-Problem, analog zu einem Multi-Criteria-Longest-Path-Problem.

In Lagerkvist (2008, Abschnitt 3.1) wird angemerkt, dass der Propagierungsalgorithmus von Pesant (2004) ohne größere Veränderungen auch dann verwendet werden kann, wenn der zugrunde liegende FSA nichtdeterministisch ist. Die Verwendung eines nichtdeterministischen FSA ist hierbei dadurch motiviert, dass er exponentiell weniger Zustände besitzen kann als ein deterministischer FSA (vgl. Rabin und Scott, 1959, Abschnitt 2) und somit unter Umständen effizienter zu handhaben ist.

Cheng et al. (2012) beschäftigt sich mit dem Regular-Membership-Constraint und dem *MDDC-Constraint* (mddc Constraint), einem globalen Constraint, der auf MDDs beruht (siehe Cheng und Yap, 2010). Er geht davon

2003).

aus, dass über verschiedenen Repräsentationsformen dieselben gültigen Wertetupel definiert werden können. Es werden nichtdeterministische FSAs, deterministische FSAs und reduzierte Multi-Valued-Decission-Diagramme (reduzierte MDDs) miteinander verglichen und es wird ein generalisierter Kantenkonsistenz-Algorithmus vorgestellt, der mit allen drei Darstellungsformen als Eingabe arbeitet. Es wird gezeigt, dass die Verwendung von nichtdeterministischen FSAs zum geringsten Platzbedarf und insgesamt zur schnellsten Laufzeiten führt. Des Weiteren wird ein Algorithmus präsentiert, der es ermöglicht, aus einem nichtdeterministischen FSA direkt ein reduziertes MDD zu erstellen.

3.2 Repräsentation eines SCSPs mithilfe von FSMs

In diesem Abschnitt wird gezeigt, wie ein SCSP mithilfe von FSMs repräsentiert werden kann. Der Ansatz ist dem von Vempaty (1992) in den Grundzügen sehr ähnlich. Jedoch ist dieser hier um zwei grundlegende Aspekte erweitert: einerseits durch die Einbeziehung weicher Constraints, mit denen es möglich ist, Verletzbarkeit, Präferenzen und Unsicherheit abzubilden und andererseits durch die Trennung zwischen dem Typ und dem Wert eines Constraints, was mit der Separation zwischen dem Skopus und der Relation eines Constraints innerhalb eines CSP korrespondiert. Durch Letzteres wird es ermöglicht, Constraint-Propagierungstechniken einzubeziehen.

Ein SCSP besteht aus einem Constraint-System, aus Constraints und einem Constraint-Problem. Im Folgenden wird gezeigt, wie diese drei Bestandteile mithilfe von FSMs repräsentiert werden können. Der Ansatz wird primär aus dem Blickwinkel der formalen Sprachen und Potenzreihen erarbeitet und zur Exemplifizierung werden FSMs herangezogen. Diese müssen hierbei nicht zwingend deterministisch und minimal sein. Um den Ansatz zu erarbeiten, wird auf die Definitionen eines SCSPs aus Abschnitt 2.3.3 zurückgegriffen.

3.2.1 Constraint-System

Formal ist ein Constraint-System $CS = (S, D, V)$ ein 3-Tupel, wobei S der verwendete c-Semiring, V eine geordnete Menge von Variablen und D der Wertebereich ist (siehe Abschnitt 2.3.3 Definition 18). In diesem Abschnitt

wird gezeigt, wie der Wertebereich D, die Variablenmenge V und Teilmengen aus V mithilfe von FSMs repräsentiert werden können.

Wertebereich

Der Wertebereich D kann über eine endliche rationale Sprache repräsentiert werden, wenn ein Wert des Wertebereichs als ein Zeichen im Alphabet der rationalen Sprache dargestellt wird. Dies ist möglich, weil der Wertebereich D wie auch das Alphabet einer rationalen Sprache endlich ist. Solch eine Repräsentationsform wird z. B. in Vempaty (1992) und in Pesant (2004) vorgeschlagen.

In dieser Arbeit werden die Werte des Wertebereichs D wie in Golden und Pang (2006) als Zeichenketten und der Wertebereich selbst als rationale Sprache repräsentiert, wobei sich zunächst auf einen endlichen Wertebereich beschränkt wird und somit auf eine endliche rationale Sprache. Ein unendlicher Wertebereich und dessen Repräsentation als unendliche rationale Sprache wird in Abschnitt 3.4.1 behandelt.

Es sei der Wertebereich $D = \{d_1, d_2, ..., d_m\}$ und eine bijektive Abbildung $\varphi : D \to \Delta^*$ mit $\Delta \subset \Sigma$ gegeben, die Werte aus D auf Zeichenketten über dem Alphabet Δ abbildet. Σ bezeichnet hierbei das Alphabet der rationalen Sprachen und rationalen Potenzreihen und die Abbildung φ ist definiert durch $\varphi(d_r) = d'_r$ mit $d'_r \in \Delta^*$ für jedes r mit $1 \leq r \leq m$. Mithilfe dieser Abbildung kann ein Wert d_r durch eine einelementige rationale Sprache repräsentiert werden:

$$L(d_r) = \{\varphi(d_r)\} = \{d'_r\} \tag{3.1}$$

Hier ist zu beachten, dass ein Wert auch durch eine einelementige rationale Sprache dargestellt werden kann, die das leeren Wort enthält.

Der Wertebereich D kann durch eine rationale Sprache repräsentiert werden, die aus der Vereinigung der einelementigen rationalen Sprachen resultiert, die die einzelnen Werte umsetzen:

$$L(D) = L(d_1) \cup L(d_2) \cup ... \cup L(d_m) \tag{3.2}$$

Wenn ausschließlich Zeichenketten der Länge 1 betrachtet werden, entspricht diese Repräsentationsform der in Vempaty (1992) und Pesant (2004).

Es soll ein Beispiel anhand des Wertebereichs $D = \{0, 1, ..., 9999\}$, der aus natürlichen Zahlen von 0 bis 9999 besteht, gegeben werden. Jede dieser

Zahlen wird durch eine Zeichenkette repräsentiert, die die Länge vier besitzt und der Konkatenation der einzelnen Ziffern entspricht und eventuell am Anfang mit Nullen aufgefüllt ist. Der minimale deterministische FSA, der diesen Wertebereich darstellt, ist in Abbildung 3.3 gegeben. Gegeben eine

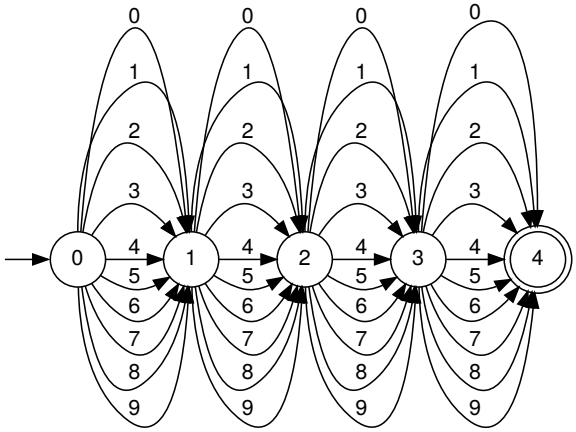


Abbildung 3.3: Ein FSA, der den Wertebereich $D = \{0, 1, ..., 9999\}$ repräsentiert.

Zeichenkette $d_3d_2d_1d_0$ mit $d_0, d_1, d_2, d_3 \in \{0, 1, ..., 9\}$, die von dem FSA erkannt wird, dann ist die korrespondierende natürliche Zahl n wie folgt definiert: $n = \sum_{i=0}^{3} 10^i d_i = d_0 + 10d_1 + 10^2d_2 + 10^3d_3$. Der FSA besitzt vier Zustände und 40 Übergänge und das Alphabet Δ besteht aus zehn Zeichen. Wenn in diesem Beispiel jedoch ein Wert des Wertebereichs als ein Symbol in Δ repräsentiert werden würde, hätte der entsprechende minimale deterministische FSA zwei Zustände und 10.000 Übergänge und das Alphabet Δ bestände aus 10.000 Zeichen.

Ein anderes Beispiel ist ein Wertebereich über endliche Mengen von natürlichen Zahlen. Angenommen, der Wertebereich besteht aus allen Mengen, die über die natürlichen Zahlen von 1 bis 5 gebildet werden können, also $D = \mathcal{P}(\{1, 2, 3, 4, 5\})$. Dann kann dieser kompakt durch den FSA in Abbildung 3.4 repräsentiert werden. Innerhalb der Repräsentation ist die übliche lineare Ordnung über den natürlichen Zahlen angenommen. Gegeben eine Zeichenkette $d_1d_2...d_n$, die der FSA akzeptiert, mit $d_1, d_2, ..., d_n \in \{1, 2, 3, 4, 5\}$ und $d_1 < d_2 < ... < d_n$ und $n <= 5$, dann ist die korrespondierende Menge an natürlichen Zahlen gegeben durch $\{d_1, d_2, ..., d_n\}$. Die leere Zeichenkette stellt hierbei die leere Menge dar. Der FSA hat 15 Übergänge und 6 Zustände und repräsentiert insgesamt 32 Mengen. Wenn alle Mengen berücksichtigt wür-

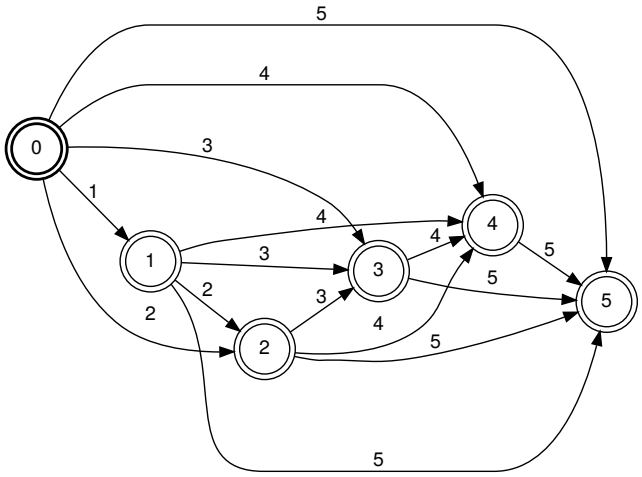


Abbildung 3.4: Ein FSA, der den Wertebereich $D = \mathcal{P}(\{1, 2, 3, 4, 5\})$ repräsentiert.

den, die über die Zahlen von 1 bis 10 gebildet werden können, hätte der FSA 55 Übergänge und 11 Zustände und würde bereits 1024 Mengen abbilden.

Über eine geeignete Zeichenkettenrepräsentation ist es demnach möglich, den Wertebereich D kompakt zu repräsentieren. Dies ist dann der Fall, wenn die einzelnen Werte des Wertebereichs in ihrer Zeichenkettenrepräsentationsform viele Präfixe und Suffixe teilen. Zudem kann das Alphabet der FSMs kleiner gehalten werden, dessen Größe beispielsweise Einfluss auf die Laufzeit der Automatenminimierung hat (siehe Almeida et al., 2008).

Variablen

Jede Variable $v_k \in V$ wird extensional über die rationale Sprache dargestellt, die den Wertebereich D umsetzt:

$$L(v_k) = L(D) \tag{3.3}$$

Eine Variable lässt sich auch kompakter repräsentieren, wenn die genauere Struktur des Wertebereichs D ignoriert wird. Dann wird eine Variable $v_k \in V$ durch die unendliche rationale Sprache Δ^* abgebildet:

$$L(v_k) = \Delta^* \tag{3.4}$$

Der Vorteil dieser Repräsentationsform ist, dass die FSAs in der Regel weniger Zustände und Übergänge besitzen. Im Folgenden wird jedoch an der Repräsentation einer Variable über die Werte des Wertebereichs D festhalten. Die kompaktere Darstellung bleibt aber als alternative möglich und ist für die praktische Anwendung interessant.

Die Variablenmenge $V = \{v_1, v_2, ..., v_n\}$ mit $v_1 \prec v_2 \prec ... \prec v_n$ kann als rationale Sprache repräsentiert werden, die das Ergebnis der Konkatenation der rationalen Sprachen ist, die die einzelnen Variablen aus V darstellen. Hierbei sind die Variablen zusätzlich mit den Markierungszeichen $\triangleleft$ und $\triangleright$ mit $\triangleleft, \triangleright \in \Sigma - \Delta$ eingefasst, die deren Anfang und Ende markieren:

$$L(V) \;=\; \triangleleft\, L(v_1) \triangleright\triangleleft L(v_2) \triangleright \cdots \triangleleft L(v_n) \triangleright \tag{3.5}$$

Die Markierungssymbole sind notwendig, um eine eindeutige Zuordnung der Werte zu den einzelnen Variablen zu gewährleisten. Ohne diese Symbole wäre es nicht möglich, die Grenze zwischen adjazenten Variablen zu bestimmen, wenn der Wertebereich aus Werten bestünde, die nicht die gleiche Länge besitzen. Der FSA, der die Variablenmenge V repräsentiert, ist in Abbildung 3.5 skizziert. Er ist in verschiedene Ebenen eingeteilt, über die die einzelnen

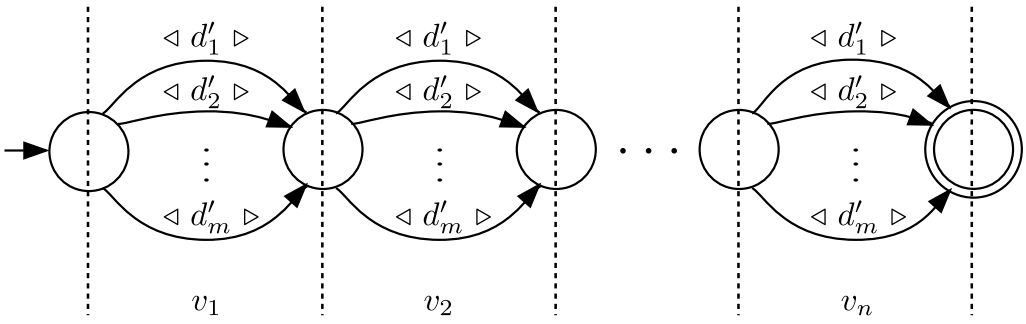


Abbildung 3.5: Variablenmenge V repräsentiert als FSA.

Werte den Variablen zugeordnet werden können. Dieser FSA hat Ähnlichkeit mit einem Universal-Constraint in Vempaty (1992), also einem Constraint der nicht verletzt werden kann, da er alle möglichen Wertetupel akzeptiert.

Variablenteilmengen

Die Typen der Constraints und der Constraint-Probleme bezeichnen Teilmengen von V (siehe Definition 19 und 20 aus Abschnitt 2.3.3).

Um diese zu repräsentieren, werden spezielle Markierungszeichen verwendet: Ausgehend von der rationalen Sprache $L(V)$ werden alle Variablen, welche nicht in einer entsprechenden Teilmenge enthalten sind, statt mit den Zeichen $\triangleleft$ und $\triangleright$ mit den Zeichen $\blacktriangleleft$ und $\blacktriangleright$ mit $\blacktriangleleft, \blacktriangleright \in \Sigma - \Delta$ markiert. Die Potenzmenge von V, geschrieben $P(V)$, kann dann als endliche rationale Sprache repräsentiert werden, die alle möglichen Markierungsvarianten mit den Markierungszeichen $\triangleleft$ und $\triangleright$ sowie $\blacktriangleleft$ und $\blacktriangleright$ beinhaltet:

$$L(\mathcal{P}(V)) = \left(\begin{array}{c} \triangleleft L(v_1) \triangleright \\ \cup \\ \blacktriangleleft L(v_1) \blacktriangleright \end{array} \right) \left(\begin{array}{c} \triangleleft L(v_2) \triangleright \\ \cup \\ \blacktriangleleft L(v_2) \blacktriangleright \end{array} \right) \cdots \left(\begin{array}{c} \triangleleft L(v_n) \triangleright \\ \cup \\ \blacktriangleleft L(v_n) \blacktriangleright \end{array} \right) \tag{3.6}$$

Zur Verdeutlichung ist der FSA $A(P(V))$ in Abbildung 3.6 zu sehen. Über

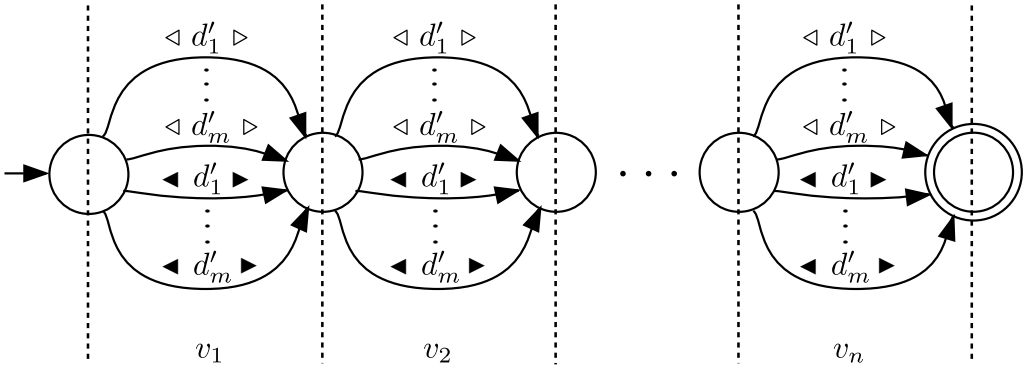


Abbildung 3.6: Potenzmenge von V repräsentiert als FSA.

die einzelnen Ebenen des FSAs und über die Zeichen $\triangleleft$ und $\triangleright$ sowie $\blacktriangleleft$ und $\blacktriangleright$ ist den Variablen die zusätzliche Information über die Teilmengenzugehörigkeit gegeben. Diese bleibt jedoch unspezifiziert, da über die Markierungszeichen sowohl eine Zugehörigkeit als auch eine Nichtzugehörigkeit möglich ist. Gemäß dieser Teilmengenrepräsentation ist die leere Menge $\varnothing$ durch die folgende endliche rationale Sprache $L(\varnothing) \subset L(P(V))$ repräsentiert:

$$L(\varnothing) = \blacktriangleleft L(v_1) \blacktriangleright \blacktriangleleft L(v_2) \blacktriangleright \cdots \blacktriangleleft L(v_n) \blacktriangleright \tag{3.7}$$

Rationale Sprachen, die Teilmengen von V darstellen, sind echte Teilmengen der rationalen Sprache $L(P(V))$. Es existieren aber auch echte Teilmengen von $L(P(V))$, die in unserem Sinne keine Teilmengen von V repräsentieren.

Dies ist der Fall, wenn nicht alle Werte des Wertebereichs einer Variable zugeordnet sind. Die Repräsentationsform ist also stets im Kontext der Repräsentation von Teilmengen von V zu verstehen.

Es soll ein konkretes Beispiel gegeben werden, welches sich im Bereich der syntaktischen Analyse bewegt. Hierzu wird angenommen, dass die einzelnen Variablen des SCSPs für natürlichsprachliche Wörter stehen, die bestimmte grammatische Funktionen innerhalb eines Satzes einnehmen können. In Kapitel 4 wird dieses realitätsnahe Problem im Detail behandelt. In diesem Beispiel wird sich auf die folgenden drei Satzfunktionen beschränkt: Wurzelknoten eines Satzes (`S`), Subjekt (`SUBJ`) und Akkusativobjekt (`OBJA`). Es wird angenommen, dass der Wertebereich des Constraint-Systems $CS = (S, D, V)$ aus diesen drei Satzfunktionen besteht und durch $D = \{\texttt{S}, \texttt{SUBJ}, \texttt{OBJA}\}$ gegeben ist und dass $V = \{v_1, v_2, v_3, v_4\}$ mit $v_1 \prec v_2 \prec v_3 \prec v_4$, wobei die Ordnung der Variablen der Abfolge der Wörter im Satz folgt. Die Werte aus D werden durch einelementige Zeichenketten repräsentiert, deren Zeichen den Werten in D entsprechen, wobei $\Delta = D$ ist.

Der Beispiel-FSA, der die Teilmenge $I \subset V$ mit $I = \{v_1, v_4\}$ repräsentiert, ist in Abbildung 3.7 dargestellt.

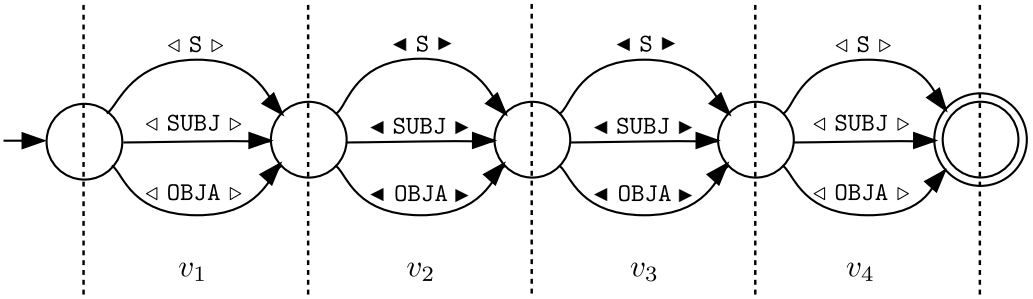


Abbildung 3.7: Beispielvariablenteilmenge als FSA.

Im Kontext des Lösens eines Constraint-Problems wird es notwendig, Teilmengen von V zu vereinigen. Gegeben zwei Teilmengen $I_1 \subset V$ und $I_2 \subset V$, dann ist die Variablenmengenvereinigung von $L(I_1)$ und $L(I_2)$ mit $I_1, I_2 \subset$

$L(P(V))$, geschrieben $L(I_1) \sqcup L(I_2)$, definiert durch:

$$L(I_1) \sqcup L(I_2) =$$

$$Range\left(\begin{array}{c} Range\left(\begin{array}{c} L(I_1) \\ \circ \\ \lhd\,(\mapsto)\lhd_2 \\ \circ \\ \rhd(\mapsto)\,\rhd_2 \\ \circ \\ \blacktriangleleft\,(\mapsto)\lhd_2 \\ \circ \\ \blacktriangleright(\mapsto)\,\rhd_2 \\ \circ \\ \lhd \mapsto \lhd_1 \\ \circ \\ \rhd \mapsto \rhd_1 \end{array}\right) \cap Range\left(\begin{array}{c} L(I_2) \\ \circ \\ \blacktriangleleft\,(\mapsto)\lhd_1 \\ \circ \\ \blacktriangleright(\mapsto)\,\rhd_1 \\ \circ \\ \lhd \mapsto \lhd_2 \\ \circ \\ \rhd \mapsto \rhd_2 \end{array}\right) \circ \left(\begin{array}{c} \lhd_1 \mapsto \lhd \\ \circ \\ \rhd_1 \mapsto \rhd \\ \circ \\ \lhd_2 \mapsto \lhd \\ \circ \\ \rhd_2 \mapsto \rhd \end{array}\right) \end{array}\right) \tag{3.8}$$

wobei die rationale Sprache $L(I_1) \sqcup L(I_2)$ der rationalen Sprache $L(I_1 \cup I_2)$ entspricht.

Der reguläre Ausdruck in 3.8 vollführt im ersten Schritt Umschreibungen von Symbolen bezüglich $L(I_1)$ und $L(I_2)$. Diese sind nötig, um eine Markierung der Variablen zu berechnen, die die Vereinigung beider Variablenmengen repräsentiert. Die Zeichen $\lhd$ und $\rhd$ werden bezüglich $L(I_1)$ auf $\lhd_1$ und $\rhd_1$ abgebildet. Hierdurch ist es möglich, die Menge an Variablen in I_1 durch die Variablen in I_2 zu erweitern, indem hinsichtlich $L(I_2)$ die Zeichen $\blacktriangleleft$ und $\blacktriangleright$ optional in $\lhd_1$ und $\rhd_1$ umgeschrieben werden. Gleichzeitig werden die Zeichen $\lhd$ und $\rhd$ bezüglich $L(I_2)$ auf $\lhd_2$ und $\rhd_2$ abgebildet. Hierdurch ist es möglich, die Menge an Variablen in I_2 durch die Variablen in I_1 zu erweitern, indem hinsichtlich $L(I_1)$ die Zeichen $\blacktriangleleft$ und $\blacktriangleright$ optional in $\lhd_2$ und $\rhd_2$ umgeschrieben werden. Gemeinsame Variablen in I_1 und I_2 werden über die optionale Abbildung von $\lhd$ und $\rhd$ auf $\lhd_2$ und $\rhd_2$ innerhalb von $L(I_1)$ behandelt. Daraufhin wird der Schnitt der Ergebnisse dieser Umschreibungen berechnet. Danach markieren die Zeichen $\lhd_2$ und $\rhd_2$ die Variablen, die nicht in der Menge I_1 aber in der Menge I_2 vorhanden sind und die Zeichen $\lhd_1$ und $\rhd_1$ die Variablen, die nicht in der Menge I_2 aber in der Menge I_1

enthalten sind. Zuletzt werden die Zeichen $\triangleleft_1$ und $\triangleright_1$ und die Zeichen $\triangleleft_2$ und $\triangleright_2$ wieder auf $\triangleleft$ und $\triangleright$ abgebildet. Die rationale Sprache, die die Menge $I_1 \cup I_2$ repräsentiert, kann entweder unmittelbar durch $L(I_1 \cup I_2)$ erstellt oder über $L(I_1) \sqcup L(I_2)$ berechnet werden. Hierbei dürfte die direkte Konstruktion in der Praxis effizienter zu realisieren sein. Die Berechnung über $L(I_1) \sqcup L(I_2)$ macht hingegen deutlich, dass über die Automatenoperationen mit Teilmengen von V gerechnet werden kann. In Kapitel 4 können im Kontext von lokalen Konsistenzregeln Variablenteilmengen, die als rationale Sprache dargestellt sind, in eine kombinierte Repräsentation einfließen. Dann sind die Teilmengen nicht mehr separat repräsentiert, und es muss auf eine Teilmengenvereinigung zurückgegriffen werden, die auf Automatenoperationen beruht.

3.2.2 Constraints

Gegeben ein Constraint-System $CS = (S, D, V)$, dann ist ein Constraint formal ein Paar (def, con), wobei $con \subseteq V$ dessen Typ und $def : D^{|con|} \to S$ dessen Wert ist. Der Wert eines Constraints bildet hierbei Wertetupel aus con auf Elemente in S ab (siehe Abschnitt 2.3.3 Definition 19). In diesem Abschnitt wird gezeigt, wie ein Constraint mithilfe von FSMs repräsentiert werden kann. Dies beinhaltet die Repräsentation des Typs, der Wertetupel und des Wertes eines Constraints.

Constraint-Typ

Der Typ eines Constraints ist eine Teilmenge von V und kann daher als rationale Sprache repräsentiert werden, die eine echte Teilmenge von $L(P(V))$ ist.

Wertetupel

Ein Wertetupel t kann als eine Zeichenkette repräsentiert werden mit $L(t) \in (\triangleleft\, L(D)\, \triangleright)^{|con|}$, wobei die Markierungszeichen $\triangleleft$ und $\triangleright$ verwendet werden, um den Anfangs- und Endpunkt der einzelnen Werte zu markieren. Die Markierungszeichenverwendung ist hierbei analog zu der Repräsentation der Variablenmengen. Im Folgenden wird kein notationeller Unterschied zwischen einer Zeichenkette und einer einelementigen rationalen Sprache gemacht. So

ist mit $L(t)$ je nach Kontext eine Zeichenkette oder eine einelementige rationale Sprache gemeint, die einen Wertetupel repräsentiert. Gegeben ein Wertetupel $t = (d_1, d_2, ..., d_{|con|})$, dann ist die korrespondierende Zeichenkette definiert durch:

$$L(t) \;=\; \triangleleft\, L(d_1) \triangleright\!\triangleleft L(d_2) \triangleright \cdots \triangleleft\, L(d_{|con|}) \triangleright \tag{3.9}$$

Der leere Wertetupel wird in dieser Repräsentation durch die leere Zeichenkette repräsentiert. Im weiteren Verlauf wird keine strikte Unterscheidung zwischen den Wertetupeln und ihrer Zeichenkettenrepräsentation gemacht. So kann z. B. formuliert werden, dass eine FSM den Wertetupel t = (d_1,d_2, ...,$d_{|con|}$) erkennt oder generiert. Zudem kann für das Semiring-Element, das einem Wertetupel t über eine rationale Potenzreihe $\mathcal{L}$ zugewiesen wird, $\mathcal{L}(t)$ geschrieben werden.

Constraint-Wert

Der Wert eines Constraints kann als eine rationale Potenzreihe repräsentiert werden, die Wertetupel aus $(\triangleleft\, L(D)\, \triangleright)^{|con|}$ auf Elemente in S abbildet und alle anderen Zeichenketten auf das Nullelement des c-Semirings. So ist der Support der rationalen Potenzreihe $L(def)$ eine Teilmenge der folgenden rationalen Sprache:

$$L(def) \subseteq (\triangleleft L(D) \triangleright)^{|con|} \tag{3.10}$$

wobei $L(def)(t) = def(t)$ für alle Wertetupel t.

Die Repräsentation der Variablenmenge V und die des Wertes eines Constraints liegen sehr nahe beieinander. Im Fall des Universal-Constraints (def, con) mit $con = V$ entspricht der Support der rationalen Potenzreihe $L(def)$ sogar der rationalen Sprache $L(con)$ und ist somit gleich der rationalen Sprache $L(V)$. Der Wert eines Universal-Constraints wird durch eine rationale Potenzreihe dargestellt, die alle möglichen Wertetupel für con auf das neutrale Element in S abbildet. Der WFSA, der den Wert eines Universal-Constraint mit dem Typ $con = \{k_1, k_2, ..., k_n\}$ mit $k_1 \prec k_2 \prec ... \prec k_n$ repräsentiert, ist in Abbildung 3.8 skizziert. Über die einzelnen Ebenen des WFSA sind die Werte den Variablen in con zugeordnet. Die Zuordnung der Werte zu den Variablen erfolgt über deren Abfolge und nicht über direkte Zeichenkettenpositionen wie in Vempaty (1992). Dies ist möglich, da die Werte durch die Markierungssymbole eindeutig voneinander getrennt sind. Der Support einer

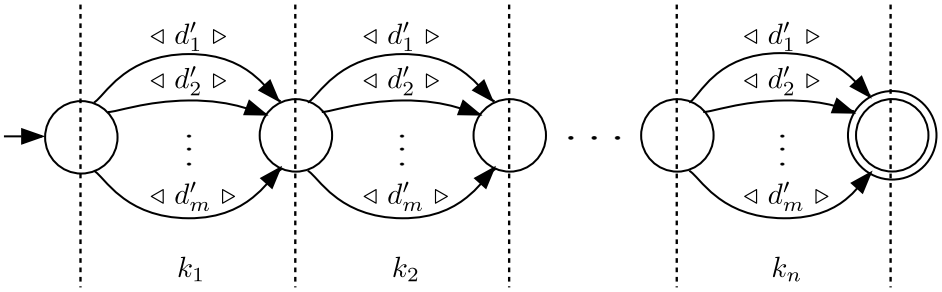


Abbildung 3.8: Repräsentation des Wertes eines Universal-Constraints als WFSA.

rationalen Potenzreihe, die den Wert eines beliebigen Constraints repräsentiert, ist hierbei eine Teilmenge des Supports der rationalen Potenzreihe, die den Wert eines Universal-Constraints darstellt, der den gleichen Typ besitzt.

Der unerfüllbare Constraint wird durch eine rationale Potenzreihe umgesetzt, die die leere Sprache als Support besitzt und die alle Wertetupel auf das Nullelement des c-Semirings abgebildet. Dies ist analog zu dem unerfüllbaren Constraint in Vempaty (1992), der durch die leere Sprache repräsentiert wird.

Um den Wert eines Constraints in die Automatenrepräsentation zu überführen, können im ersten Schritt die einzelnen Wertetupel in ihre Zeichenkettenrepräsentation gebracht werden. Ausgehend hiervon können rationale Potenzreihen konstruiert werden, die den einzelnen Zeichenketten ein c-Semiring-Element zuweisen. Diese können dann über die Summenoperation zu einer rationalen Potenzreihe zusammengefasst werden. Gegeben einen Constraint (def,con), dann ist für alle Wertetupel $t_1, t_2, ..., t_n$ über con die rationale Potenzreihe $L(def)$ definiert als:

$$L(def) = L(t_1)\langle def(t_1)\rangle \cup L(t_2)\langle def(t_2)\rangle \cup ... \cup L(t_n)\langle def(t_n)\rangle \qquad (3.11)$$

wobei $L(def)(L(t_r)) = def(t_r)$ für alle t_r mit $1 \leq r \leq n$. Bei der Konstruktion von $L(def)$ müssen aus praktischer Sicht nur Wertetupel t mit $def(t) \neq \bar{0}$ berücksichtigt werden, da Wertetupel mit $def(t) = \bar{0}$ durch rationale Potenzreihen repräsentiert werden, deren Supports automatisch der leeren Sprache entsprechen.

Es existieren verschiedene Algorithmen, mit denen eine Wertetupelliste in einen FSA kompiliert werden kann. Es kann beispielsweise im ersten Schritt

ein Trie erstellt und anschließend ein Minimierungsalgorithmus für azyklische FSAs angewendet werden. Jedoch ist dieses Vorgehen unter Umständen sehr speicherintensiv. Als alternative existieren inkrementelle Algorithmen, die sofort minimieren, wenn dem Automaten ein neuer Wertetupel hinzugefügt wird. Das Ergebnis ist dann ein minimaler deterministischer FSA (siehe z. B. Daciuk et al., 2005). Es existieren aber auch inkrementelle Algorithmen, die nichtdeterministische FSAs erzeugen, um eine kompaktere Darstellung zu ermöglichen (siehe z. B. Sgarbas et al., 2001). Im Kontext des SCSPs hat man es jedoch mit gewichteten Wertetupeln und somit mit WFSAs zu tun. Hier muss der Aspekt des Gewicht-Pushing mit einbezogen werden (vgl. Mohri, 2009, Abschnitt 6.3).

Constraint

Ein Constraint (*def*,*con*) kann durch ein Paar dargestellt werden, das aus einer rationalen Potenzreihe, die den Wert *def* repräsentiert, und einer rationalen Sprache, die den Typ *con* umsetzt, besteht.

Es soll ein Beispiel für diese Repräsentationsform gegeben werden. Hierzu wird an das Beispiel zur syntaktischen Analyse aus Abschnitt 3.2.1 angeknüpft. Dort ist ein Constraint-System $CS = (S, D, V)$ mit $D = \{$`S`,`SUBJ`, `OBJA`$\}$ gegeben, und es existieren vier Variablen $V = \{v_1,v_2,v_3,v_4\}$ mit $v_1 \prec v_2 \prec v_3 \prec v_4$. Es wird angenommen, dass das Constraint-System über den c-Semiring $S = (\mathbb{R}_0^+ \cup \{+\infty\}, min, +, +\infty, 0)$ definiert ist und ein *gewichtetes CSP* (weighted CSP) implementiert (vgl. Bistarelli et al., 1997, Abschnitt 6.4) und dass ein Constraint $c = (def,con)$ existiert mit $con = \{v_1,v_4\}$. Dieser Constraint fordert, dass alle Variablen in *con* eine unterschiedliche grammatische Funktion als Belegung besitzen müssen. Falls dies nicht der Fall ist, wird ein Strafgewicht von 5 vergeben. Der FSA, der den Typ des Constraints darstellt, ist in Abbildung 3.7 aus Abschnitt 3.2.1 gegeben, der WFSA, der den Wert des Constraints umsetzt, hingegen in Abbildung 3.9.

Es ist auch möglich, den Typ und den Wert eines Constraints und somit den gesamten Constraint als eine rationale Potenzreihe zu repräsentieren. Gegeben ein Constraint (*def*,*con*) und die rationale Sprache $L(def)$ und die rationale Potenzreihe $L(con)$, dann können $L(def)$ und $L(con)$ zu einer rationalen Potenzreihe zusammengefasst werden:

$$L((def, con)) = Dom(L(con) \circ \blacktriangleleft \Delta^* \blacktriangleright \mapsto \epsilon \circ L(def)) \tag{3.12}$$

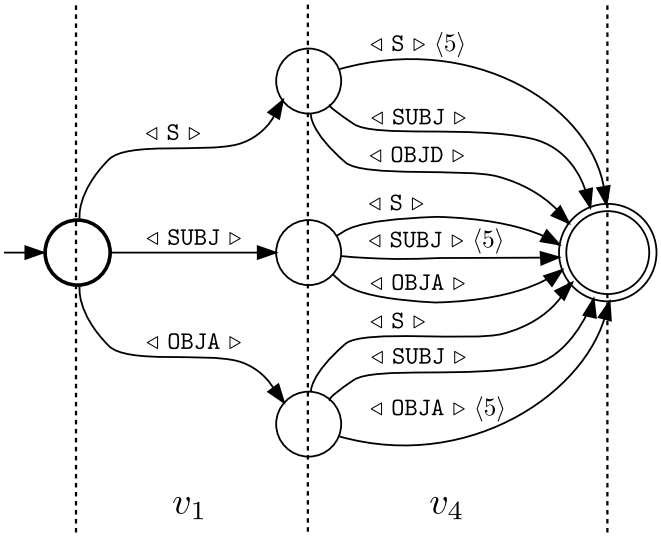


Abbildung 3.9: Beispiel-Constraint-Wert als WFSA.

Über den regulären Ausdruck werden bezüglich $L(con)$ die Variablen, die nicht der Variablenmenge *con* zugeordnet sind, auf ϵ abgebildet. Über diese Umschreibung können $L(con)$ und $L(def)$ mithilfe der Kompositionsoperation verrechnen werden. Dies ist möglich, da die Repräsentation der Variablenmengen und die der Wertetupel kompatibel sind. Der Wertebereich der resultierenden rationalen Potenzreihentransduktion ist dann eine rationale Potenzreihe, die sowohl den Typ als auch den Wert des Constraints repräsentiert. Als Beispiel für diese kombinierte Repräsentation ist in Abbildung 3.10 ein WFSA gegeben, der aus dem Typ in Abbildung 3.7 aus Abschnitt 3.2.1 und dem Wert aus Abbildung 3.9 resultiert. Obwohl v_2 und v_3 nicht im Skopus des Constraints sind, sind die Werte für diese Variablen wiederholt im WFSA repräsentiert. Demnach können bei solch einer kombinierten Repräsentationsform viele Redundanzen auftreten und ein entsprechender WFSA kann sich bei komplexeren Constraints in seiner Zustandsanzahl stark aufblähen. Zudem ist der Typ nicht mehr sauber vom Wert getrennt. Im Folgenden wird dieser kombinierten Repräsentationsform nicht weiter nachgegangen. Sie macht aber deutlich, welchen Vorteil eine Trennung von Typ und Wert in der Repräsentation eines Constraints mit sich bringt.

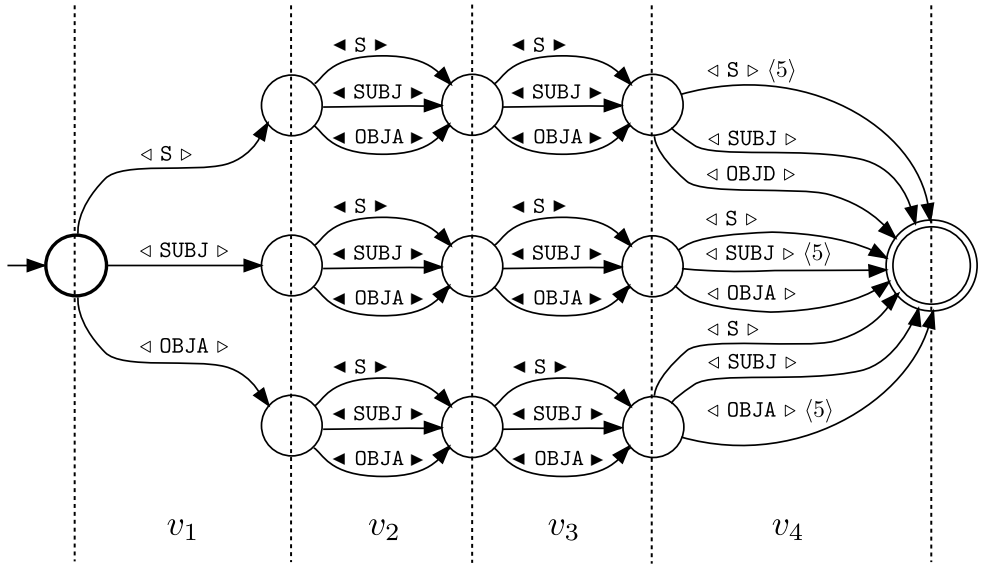


Abbildung 3.10: Beispiel-Constraint-Wert als WFSA mit Redundanzen.

3.2.3 Constraint-Problem

Formal ist ein Constraint-Problem über einem Constraint-System $CS = (S, D, V)$ ein Paar $P = (C, con)$, wobei C eine Menge von Constraints und $con \subseteq V$ dessen Typ ist (siehe Abschnitt 2.3.3 Definition 20).

Der Typ *con* ist eine Teilmenge von V und kann somit als eine echte Teilmenge von $L(P(V))$ repräsentiert werden (vgl. Abschnitt 3.2.1). Ein Constraint in C kann als ein Paar aus einer rationalen Potenzreihe und einer rationalen Sprache dargestellt werden, die dessen Wert und Typ repräsentieren (vgl. Abschnitt 3.2.2).

Innerhalb eines Constraint-Problems sind keine zwei Constraints mit dem gleichen Typ erlaubt. Daher müssen die Werte entsprechender Constraints vorab zu einem Wert zusammengefasst werden. Dieser bildet dann einen Wertetupel auf ein c-Semiring-Element ab, das der abstrakten Multiplikation der c-Semiring-Elemente entspricht, die von den zugrunde liegenden Werten zugewiesen werden. Wenn zu einem Typ con' mehrere Werte def_1,def_2,...,def_n existieren, die durch die rationalen Potenzreihen $L(def_1)$,$L(def_2)$,...,$L(def_n)$ repräsentiert sind, können diese über das Hadamard-Produkt zu einer rationalen Potenzreihe $L(def)$ zusammengefasst werden:

$$L(def) = L(def_1) \cap L(def_2) \cap ... \cap L(def_n) \tag{3.13}$$

wobei $L(def)(t) = def_1(t) \otimes def_2(t) \otimes ... \otimes def_n(t)$ für alle Tupel t. Über das Hadamard-Produkt werden die c-Semiring-Elemente zu den entsprechenden Wertetupeln abstrakt multipliziert. Somit werden die Elemente aus S beim Zusammenfassen wie gewünscht behandelt.

3.3 Lösen eines SCSPs mithilfe von FSMs

Dieser Abschnitt beschäftigt sich mit dem Lösen eines SCSPs, das mithilfe von FSMs repräsentiert ist. Hierbei werden Automatenoperationen angewendet, um die SCSP-Lösung zu berechnen. Des Weiteren wird die Umsetzung der besten Konsistenzstufe und der abstrakten Lösung behandelt.

Als Erstes wird auf die Realisierung der Tupel-Projektion eingegangen. Daraufhin wird beschrieben, wie die Kombination und die Projektion von Constraints mit FSMs realisiert ist. Letztlich wird auf die Lösung, die beste Konsistenzstufe und die abstrakte Lösung eines Constraint-Problems eingegangen. Um den Ansatz zu erarbeiten, wird auf die SCSP-Definitionen aus Abschnitt 2.3.3 zurückgegriffen.

3.3.1 Tupel-Projektion

Gegeben das Constraint-System $CS = (S,D,V)$ und zwei Variablenmengen I und I' mit $I' \subseteq I \subseteq V$, dann denotiert $\downarrow_{I'}^{I}$ die Projektion von jedem Wertetupel t von der Variablenmenge I auf die Variablenmenge I' (siehe Definition 21 aus Abschnitt 2.3.3).

Die Tupel-Projektion $\downarrow_{I'}^{I}$ kann über eine rationale Zeichenkettentransduktion $L(\downarrow_{I'}^{I})$ umgesetzt werden, die eine Abbildung von $(\triangleleft\, L(D)\, \triangleright)^{|I|}$ nach $(\triangleleft\, L(D)\, \triangleright)^{|I'|}$ realisiert. Die Transduktion kann über die rationalen Sprachen $L(I)$ und $L(I')$ konstruiert werden:

$$L(\downarrow_{I'}^{I}) = ((\begin{pmatrix} L(I) \\ \circ \\ \triangleleft(\mapsto)\blacktriangleleft \\ \circ \\ \triangleright(\mapsto)\blacktriangleright \\ \circ \\ L(I') \end{pmatrix} \circ \blacktriangleleft(\Sigma - \blacktriangleright)^* \blacktriangleright \mapsto \epsilon)^{-1} \circ \blacktriangleleft(\Sigma - \blacktriangleright)^* \blacktriangleright \mapsto \epsilon)^{-1} \quad (3.14)$$

Innerhalb des regulären Ausdrucks wird über die optionale Umschreibung der Markierungssymbole $\triangleleft$ und $\triangleright$ auf $\blacktriangleleft$ und $\blacktriangleright$ bezüglich $L(I)$ eine rationale Zeichenkettentransduktion erstellt, die $L(I)$ auf $L(I')$ abbildet. Ausgehend hiervon wird eine Zeichenkettentransduktion konstruiert, die Wertetupel für I auf Wertetupel für I' umschreibt. Hierzu werden die Variablen, die durch $\blacktriangleleft$ und $\blacktriangleright$ markiert sind, bezüglich des Definitions- und Wertebereichs der Transduktion auf Epsilon abgebildet. Dieses Variablenentfernen ist der Übergang von der Variablenrepräsentation in die Tupelrepräsentation.

Es soll ein Beispiel für eine konkrete Tupel-Projektion gegeben werden. Hierzu wird an das Beispiel zur syntaktischen Analyse aus den vorherigen Abschnitten angeknüpft. So existiert ein Constraint-System $CS = (S, D, V)$ mit $D = \{\texttt{S}, \texttt{SUBJ}, \texttt{OBJA}\}$ und mit $V = \{v_1, v_2, v_3, v_4\}$ und $v_1 \prec v_2 \prec v_3 \prec v_4$. Des Weiteren werden die zwei Mengen $I = \{v_1, v_4\}$ und $I' = \{v_1\}$ mit $I' \subseteq I \subseteq V$ angenommen. Der FSA, der die Variablenteilmenge I repräsentiert ist in Abbildung 3.7 aus Abschnitt 3.2.1 gegeben, der FST $A(\downarrow^I_{I'})$ hingegen, der die Tupel-Projektion $\downarrow^I_{I'}$ realisiert, in Abbildung 3.11.

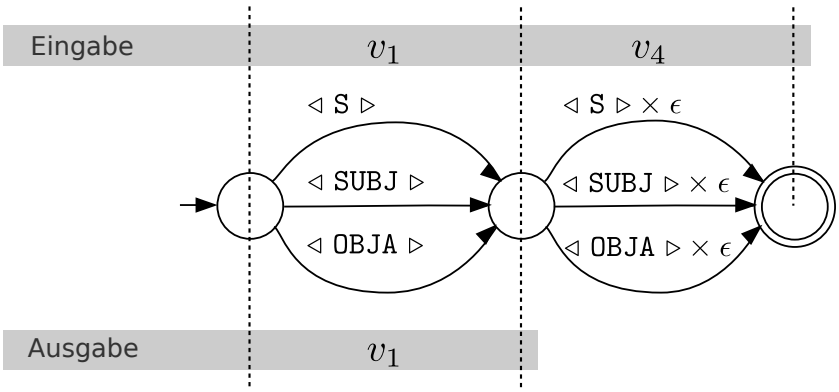


Abbildung 3.11: Beispiel-Tupel-Projektion von $\{v_1, v_4\}$ auf $\{v_1\}$ als FST.

3.3.2 Kombination

Zwei Constraints zu kombinieren bedeutet, einen neuen Constraint zu konstruieren, der ihre Typen im Skopus hat und dessen Wert mithilfe der Tupel-Projektion aus ihren Werten berechnet wird. Das c-Semiring-Element, das ein Tupel über den Wert des neuen Constraints erhält, ergibt sich aus der abstrakten Multiplikation der c-Semiring-Elemente, die den entsprechenden

Sub-Tupeln über die Werte der zugrunde liegenden Constraints zugewiesen werden (siehe Definition 22 aus Abschnitt 2.3.3).

Gegeben seien die Constraints (def_1, con_1) und (def_2, con_2), die durch $(L(def_1),L(con_1))$ und $(L(def_2),L(con_2))$ repräsentiert sind, dann ist deren Kombination $(L(def_1),L(con_1))\otimes(L(def_2),L(con_2))$ der Constraint $(L(def),$ $L(con))$ mit $(def, con) = (def_1, con_1) \otimes (def_2, con_2)$. Die rationale Sprache $L(def)$ kann über die Variablenvereinigung berechnet werden:

$$L(con) = L(con_1 \cup con_2) = L(con_1) \sqcup L(con_2) \tag{3.15}$$

Die rationale Potenzreihe $L(def)$ ist hingegen definiert durch:

$$L(def) = \begin{pmatrix} Dom(L(\downarrow_{con_1}^{con}) \circ L(def_1)) \\ \cap \\ Dom(L(\downarrow_{con_2}^{con}) \circ L(def_2)) \end{pmatrix} \tag{3.16}$$

Innerhalb des regulären Ausdrucks werden über die Tupel-Projektion $L(\downarrow_{con_1}^{con}$) die c-Semiring-Elemente zu den Wertetupeln für *con* bezüglich $L(def_1)$ und $L(def_2)$ berechnet. Die beiden Ergebnisse werden dann über das Hadamard-Produkt verrechnet. Hierbei werden die c-Semiring-Elemente der Wertetupel abstrakt multipliziert. Somit werden die Elemente aus S bei der Kombination wie gewünscht behandelt.

Es soll ein konkretes Beispiel für die Kombination zweier Constraints gegeben werden. Hierbei liegt der Fokus auf dem Constraint-Wert. Hierzu wird an das Beispiel zur syntaktischen Analyse aus den vorherigen Abschnitten angeknüpft. So existiert ein Constraint-System $CS = (S, D, V)$ mit $D = \{\texttt{S}, \texttt{SUBJ}, \texttt{OBJA}\}$ und mit $V = \{v_1, v_2, v_3, v_4\}$ und $v_1 \prec v_2 \prec v_3 \prec v_4$, welches über den Semiring $S = (\mathbb{R}_0^+ \cup \{+\infty\}, min, +, +\infty, 0)$ ein gewichtetes CSP implementiert. Des Weiteren werden zwei Constraints $c_1 = (def_1, con_1)$ und $c_2 = (def_2, con_2)$ mit $con_1 = \{v_2, v_3\}$ und $con_2 = \{v_3, v_4\}$ angenommen. Der Constraint c_1 erlaubt für die Variablen v_2 und v_3 ausschließlich eine Belegung mit SUBJ gefolgt von OBJA oder mit SUBJ geführt von OBJA. Hierbei wird eine Wertezuweisung mit SUBJ gefolgt von OBJA mit dem Gewicht 1 bestraft und eine Belegung mit SUBJ geführt von OBJA mit dem Gewicht 5. Der Constraint c_2 hingegen bestraft jede Wertezuweisung für die Variablen v_3 und v_4 mit S mit dem Gewicht 1, mit SUBJ mit dem Gewicht 2 und mit OBJA mit dem Gewicht 3. Nun sei der Constraint $(L(def), L(con))$ das Er-

gebnis der Komposition $(L(def_1), L(con_1)) \otimes (L(def_2), L(con_2))$. Dann sind die WFSAs, die die einzelnen Werte der Constraints repräsentieren, untereinander in Abbildung 3.12 aufgeführt.

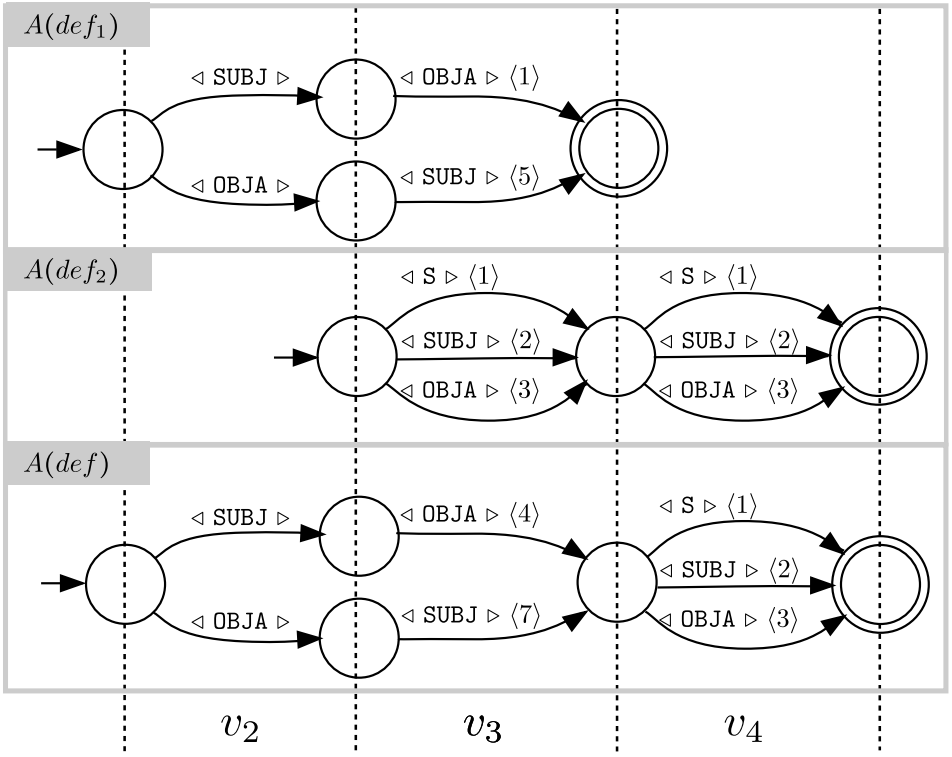


Abbildung 3.12: Beispiel-Verrechnung der Constraint-Werte $A(def_1)$ und $A(def_2)$ zu dem Constraint-Wert $A(def)$ bei der Constraint-Komposition.

3.3.3 Projektion

Mithilfe der Projektion kann ein Constraint über eine Menge von Variablen projiziert werden, um einen neuen Constraint zu berechnen, der eine Teilmenge des Typs des zugrunde liegenden Constraints im Skopus hat. Wenn ein Wertetupel des neuen Constraints über verschiedene Wertetupel aus dem Original-Constraint ermittelt werden kann, werden die entsprechenden c-Semiring-Elemente abstrakt addiert (siehe Definition 23 aus Abschnitt 2.3.3).

Gegeben ein Constraint (def, con), der durch (def, con) repräsentiert ist, und gegeben eine Variablenmenge $I \subseteq V$, dann ist die Projektion $(L(def), L(con)) \Downarrow_I$ der Constraint $(L(def'), L(con'))$ mit $(def', con') = (def, con) \Downarrow_I$.

Die rationale Sprache, die den Typ des Constraints repräsentiert, entspricht der rationalen Sprache, die die Variablenmenge I darstellt:

$$L(con') = L(I) \tag{3.17}$$

Die rationale Potenzreihe, die den Wert des Constraints umsetzt, ist definiert durch:

$$L(def') = Range(L(def) \circ L(\downarrow_I^{con})) \tag{3.18}$$

Innerhalb des regulären Ausdrucks wird die Tupel-Projektion $L(\downarrow_I^{con})$ auf die rationale Potenzreihe $L(def)$ angewendet. Wenn einem Wertetupel aus $L(def')$ verschiedene Wertetupel aus $L(def)$ zugrunde liegen, ergibt sich das c-Semiring-Element aus der abstrakten Addition der entsprechenden c-Semiring-Elemente. Somit werden die Elemente aus S bei der Projektion wie gewünscht behandelt.

Es soll ein konkretes Beispiel für die Tupel-Projektion gegeben werden. Hierbei liegt der Fokus auf dem Constraint-Wert. Hierzu wird an das Beispiel zur syntaktischen Analyse aus dem vorhergehenden Abschnitt 3.3.2 angeknüpft. Dort ist ein Constraint-System $CS = (S, D, V)$ mit $D = \{$`S`,`SUBJ`, `OBJA`$\}$ und mit $V = \{v_1,v_2,v_3,v_4\}$ und $v_1 \prec v_2 \prec v_3 \prec v_4$ gegeben, welches über den Semiring $S = (\mathbb{R}_0^+ \cup \{+\infty\},min,+,+\infty,0)$ ein gewichtetes CSP implementiert. Hier wird angenommen, dass der Constraint (def,con), der aus der Kombination zweier Constraints entstanden ist (siehe Abbildung 3.12, unten), auf die Menge $I = \{v_2\}$ projiziert wird. Der WFSA, der den Wert def und der WFSA, der den Wert des Constraints $(def',con') = (def,con) \Downarrow_I$ repräsentiert, sind in Abbildung 3.13 untereinander aufgeführt.

3.3.4 Lösung

Gegeben ein Constraint-Problem $P = (C,con)$, dann ist die Lösung von P definiert als ein Constraint, der den Typ *con* besitzt und aus dem gesamten Constraint-Problem induziert wurde. Hierzu werden zuerst alle Constraints in C kombiniert. Anschließend wird der resultierende Constraint über *con* projiziert.

Angenommen alle Constraints in C sind über rationale Potenzreihen und rationale Sprachen repräsentiert und in dieser Repräsentation in der Menge C_L zusammengefasst. Dann ist die Lösung eines Constraint-Problems defi-

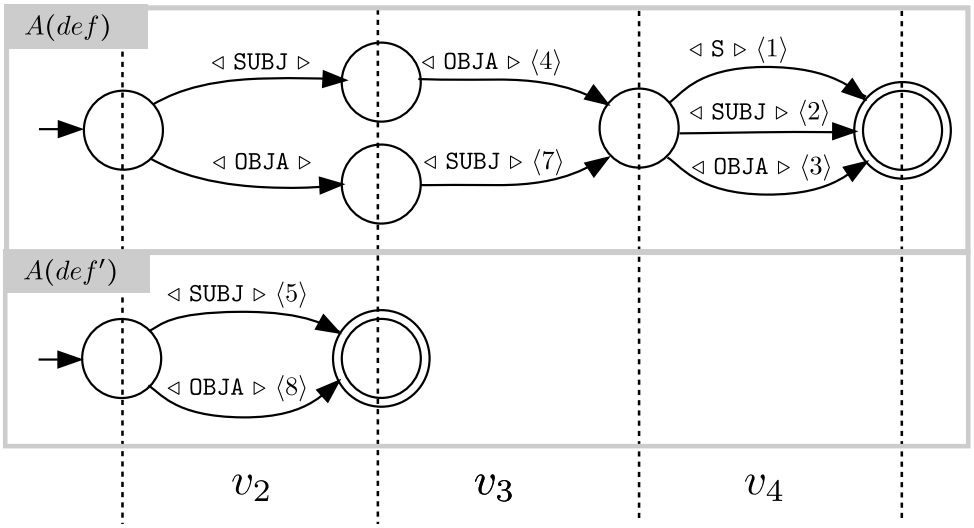


Abbildung 3.13: Beispiel-Berechnung des Constraint-Wertes $A(def')$ aus dem Constraint-Wert $A(def)$ bei der Constraint-Projektion von $\{v_2, v_3, v_4\}$ auf $\{v_2\}$.

niert als der Constraint $(L(def),L(con))$ mit $(def,con) = (\bigotimes C) \Downarrow_{con}$. Dieser Constraint kann über die Operationen Komposition und Projektion berechnet werden, deren Umsetzung in Abschnitt 3.3.2 und 3.3.3 beschrieben ist:

$$(L(def), L(con)) = (\bigotimes_{(L(def'),L(con'))\in C_L} (L(def'), L(con'))) \Downarrow_{con} \qquad (3.19)$$

Wenn eine Repräsentationsform des Lösungs-Constraint gewählt wird, bei der der Wert und der Typ als eine rationale Potenzreihe dargestellt ist, wie es in Abschnitt 3.2.2 beschrieben ist, kann ein regulärer Ausdruck angegeben werden, der das gesamte Constraint-Problem repräsentiert und anhand dessen das Constraint-Problem gelöst werden kann. Bei der Angabe eines solchen monolithischen regulären Ausdrucks wären jedoch die Lösungsvorschrift und die reine Repräsentation nicht mehr voneinander getrennt.

Es soll eine Komplexitätsanalyse des Lösens eines SCSPs, das mithilfe von FSMs repräsentiert ist, gegeben werden: Um ein Constraint-Problem zu lösen, wird die Kombinationsoperation wiederholt angewendet, die Projektionsoperation hingegen nur ein Mal am Ende. Daher spielt diese bei der Komplexitätsanalyse keine Rolle. Über die Kombinationsoperation werden Tupel-Projektionen auf Constraint-Werte angewendet, und es werden die

Ergebnisse über das Hadamard-Produkt miteinander verrechnet. Die Tupel-Projektion wird hierbei herangezogen, um mit den unterschiedlichen Constraint-Typen umgehen zu können. Wenn jedoch angenommen wird, dass sich wie in Vempaty (1992) alle Variablen im Skopus eines Constraints befinden, kann die Tupel-Projektion aus der Komplexitätsberechnung herausgelassen werden. Dann greift die Regel, dass es innerhalb eines Constraint-Problems keine zwei Constraints mit dem gleichen Typ geben darf. Daher müssen die entsprechenden Constraints vorab zu einem zusammengefasst werden. Die Umsetzung des Zusammenfassens ist in Abschnitt 3.2.3 beschrieben. Die einzelnen Werte der Constraints werden dort über das Hadamard-Produkt verrechnet. Genau dieser Schritt wird hier in der Komplexitätsanalyse betrachtet. Das Hadamard-Produkt zweier WFSAs $\mathcal{A}_1$ und $\mathcal{A}_2$ hat die Zeit- und Platzkomplexität von $O(|\mathcal{A}_1| \times |\mathcal{A}_2|)$, wobei $|\mathcal{A}_1|$ und $|\mathcal{A}_2|$ die Anzahl der Zustände des WFSA $\mathcal{A}_1$ bzw. $\mathcal{A}_2$ bezeichnet (siehe Mohri, 2009, Abschnitt 5.1). Angenommen es existiert ein Constraint-Problem $P = (C, con)$ mit $C = \{(def_1, con'), (def_2, con'), ..., (def_n, con')\}$ und $con' = V$, dann wird anhand des Ausdrucks in 3.13 aus Abschnitt 3.2.3 $(n-1)$-mal nacheinander die Schnitt-Operation auf die WFSAs, die die Werte der Constraints repräsentieren, angewendet: $A(def_1) \cap A(def_2) \cap ... \cap A(def_n)$. Die Zeit- und Platzkomplexität des Zusammenfassens der WFSAs ist demnach $O(|A(def_1)| \times |A(def_2)| \times ... \times |A(def_n)|)$. Wenn k die maximale Anzahl an Zuständen ist, die von den einzelnen WFSAs erreicht wird, können die individuellen Zustandsgrößen durch k ersetzt werden. Dann ist die Platz- und Zeitkomplexität des Lösens eines Constraint-Problems $O(k^{|C|})$ und demnach wie das CSP-Lösen mit FSAs im schlechtesten Fall exponentiell (vgl. Vempaty, 1992, Abschnitt „Overall Complexity of the New Strategy“).

Im Kontext der Berechnung des Schnitts ist das Bestimmen der optimalen Ordnung der Constraints, die die wenigsten Zustände während der Schnittoperationen verursacht, NP-vollständig (siehe hierzu Bonfante und Roux, 2008). Des Weiteren ist die Bestimmung der Variablenordnung, die zu dem kleinstmöglichen Lösungsautomaten führt, NP-hart (siehe Amilhastre et al., 2002, Abschnitt 6 und im Kontext geordneter binärer Entscheidungsdiagramme Bollig und Wegener, 1996).

Um das Lösen eines Constraint-Problems zu beschleunigen, können die FSMs nach der Anwendung einzelner Automatenoperationen optimiert werden. Die Optimierung bezieht sich hierbei auf azyklische WFSAs, die über

einen c-Semiring und somit über einen tropischen Semiring mit Top-Element gewichtet sind. Im Fall eines tropischen Semirings hat die Überführung eines azyklischen WFSA in einen Epsilon-freien WFSA eine Zeitkomplexität von $O(|Q|^2 + |Q| \times |E|)$, wobei Q die Menge von Zuständen und E die Menge von Übergängen bezeichnet (siehe Mohri, 2001, Abschnitt 3.2). Hierbei ist zu beachten, dass eine Epsilon-Entfernung nur dann notwendig ist, wenn eine Constraint-Projektion angewendet wurde. In diesem Fall werden Zeichenketten über Epsilon-Umschreibungen verkürzt. Jeder azyklische Epsilon-freie WFSA kann in einen deterministischen azyklischen WFSA überführt werden, der das gleiche Verhalten aufweist. Die Determinisierungskomplexität ist hierbei exponentiell (siehe Mohri, 2009, Abschnitt 6.2). Eine Determinisierung ist jedoch nur dann nötig, wenn eine Epsilon-Entfernung vorhergegangen ist, also wenn die Constraint-Projektion angewendet wurde. Nach der Constraint-Kombination ist hingegen keine Determinisierung notwendig. Wenn das Hadamard-Produkt zweier deterministischer WFSAs berechnet wird, ist das Ergebnis stets deterministisch. Wenn ein Identitätstransduktor involviert ist, gilt dies auch für die Komposition. Während der Constraint-Kombination wird der Definitionsbereich eines WFSTs ermittelt. Die Ermittlung des Definitionsbereichs eines deterministischen WFST, kann zu einem WFSA führen, der nichtdeterministisch ist. Innerhalb der Constraint-Kombination führt die Ermittlung des Definitionsbereichs jedoch zu einem deterministischen WFSA, da der zugrunde liegende WFST auf der Eingabeseite deterministisch ist. Dies ist der Fall, da der WFST aus der Komposition eines FSTs, der stets funktional und auf der Eingabeseite deterministisch ist, mit einem deterministischen gewichteten Identitätstransduktor entsteht. Ein deterministischer azyklischer WFSA kann im Fall des tropischen Semirings mit einer Zeitkomplexität von $O(|Q| + [E])$ in einen minimalen deterministischen azyklischen WFSA überführt werden (siehe Mohri, 2009, Abschnitt 6.4). Diese Äquivalenztransformation sollte nach jeder Constraint-Kombination angewendet werden, da nicht gewährleistet ist, dass das Ergebnis des Hadamard-Produktes und der Komposition minimal ist.

Wenn angenommen wird, dass jeder Constraint alle Variablen im Skopus hat, ist die Repräsentationsform der von Vempaty (1992) sehr ähnlich. Wenn die Constraints hingegen nicht alle Variablen im Skopus haben, muss Mehraufwand betrieben werden, um die Constraints zu kombinieren, da hier unter Umständen die Tupel-Projektion angewendet wird. Jedoch kann über diesen

zusätzlichen Aufwand das Anwachsen der Zwischenergebnisse während des Lösens erheblich abgemildert werden, da innerhalb eines Constraint-Wertes nicht für jede Variable eine Wertzuweisung repräsentiert sein muss (siehe zur Verdeutlichung auch Abbildung 3.10 aus Abschnitt 3.2.2). Des Weiteren ermöglicht die Trennung zwischen Typ und Wert die Umsetzung von Propagierungstechniken.

3.3.5 Beste Konsistenzstufe und abstrakte Lösung

Die beste Konsistenzstufe gibt an, inwieweit die Constraints eines Constraint-Systems erfüllt werden können. Diese wird berechnet, indem alle Constraints kombiniert und auf die leere Variablenmenge projiziert werden. Der Wert des resultierenden Constraints stellt dann die beste Konsistenzstufe dar.

Gegeben ein Constraint-System $CS = (S, D, V)$ und ein Constraint-Problem $P = (C, con)$, das durch $(C_L, L(con))$ mithilfe von FSMs repräsentiert ist. Dann entspricht die beste Konsistenzstufe $blevel(P)$ dem c-Semiring-Element $L(def)(\epsilon)$, wobei der Constraint $(L(def), L(\varnothing))$ entsprechend der Lösung eines Constraint-Problems berechnet wird:

$$(L(def), L(\varnothing)) = (\bigotimes_{(L(def'),L(con'))\in C_L} (L(def'), L(con'))) \Downarrow_{\varnothing} \tag{3.20}$$

Wie in Abschnitt 2.3.3 beschrieben, kann alternativ auch erst das Constraint-Problem $(C_L, L(con))$ gelöst und daraufhin das Ergebnis auf die leere Variablenmenge projiziert werden: $(\otimes C_L) \Downarrow_{\varnothing} = (\otimes C_L) \Downarrow_{con} \Downarrow_{\varnothing}$.

Die beste Konsistenzstufe kann aber auch über das Sprachgewicht bestimmt werden. Hierbei wird im ersten Schritt die Lösung berechnet und im zweiten Schritt das Sprachgewicht der rationalen Potenzreihe ermittelt, die den Wert des Lösungs-Constraints repräsentiert. Dieses Gewicht stellt dann die beste Konsistenzstufe dar. Aufseiten der FSMs kann das Gewicht eines WFSA über einen Single-Source-Shortest-Distance-Algorithmus berechnet werden (vgl. Abschnitt 2.2.1). Da der entsprechende WFSA stets azyklisch und über einen c-Semiring, der einen tropischen Semiring mit Top-Element darstellt, gewichtet ist, kann die beste Konsistenzstufe mit einer Zeitkomplexität von $O(|Q| + [E])$ ermittelt werden. Hierbei bezeichnet Q die Menge von Zuständen und E die Menge von Übergängen (siehe Mohri, 2002, Abschnitt 4.1).

Eine andere interessantere Variante der Lösung eines Constraint-Problems ist die abstrakte Lösung. Diese enthält ausschließlich solche Wertetupel, dessen c-Semiring-Elemente nicht 'schlechter' sind als andere. Die Zeichenketten, die diese Tupel repräsentieren, können über den WFSA ermittelt werden, der den Wert des Lösungs-Constraint darstellt. Wenn der c-Semiring linear geordnet ist, entsprechen die c-Semiring-Elemente zu den einzelnen Wertetupeln der abstrakten Lösung der besten Konsistenzstufe. Dann kann ein Single-Source-Shortest-Path-Algorithmus angewendet werden, um die entsprechenden Zeichenketten aus dem WFSA zu ermitteln. Die Zeitkomplexität dieser Ermittlung ist linear und entspricht der Zeitkomplexität der Berechnung der besten Konsistenzstufe. Wenn für den c-Semiring jedoch eine Pareto-Ordnung definiert ist, muss hingegen ein Single-Source-Multiobjective-Shortest-Path-Algorithmus herangezogen werden, dessen Zeitkomplexität von der Anzahl der Ordnungskriterien abhängt (siehe Tarapata, 2007).

3.3.6 Constraint-Propagierung

Um Constraint-Propagierungstechniken umzusetzen, können lokale Konsistenzregeln auch auf ein SCSP angewendet werden, das mithilfe von FSMs repräsentiert ist. Dies ist möglich, da die Regeln ausschließlich Bezug auf die Typen der Constraints nehmen und Subprobleme mithilfe der Kombination und Projektion von Constraints gelöst werden können.

Damit der Fix-Point innerhalb eines lokalen Konsistenzalgorithmus erkannt werden kann, muss es möglich sein, Constraint-Probleme auf Gleichheit zu prüfen. Dies beinhaltet hier eine Äquivalenzprüfung der Constraint-Werte, die über WFSAs repräsentiert sind. Die Berechnung, ob zwei WFSAs äquivalent sind, ist im Allgemeinen nicht immer möglich. Ein Check auf Äquivalenz ist aber dann umsetzbar, wenn die WFSAs deterministisch sind und somit die Zwillingseigenschaft besitzen. Ein Äquivalenztest kann über das Pushing von Gewichten und über einen Standard-Algorithmus zum Test der Gleichheit von FSAs realisiert werden (siehe Mohri, 2009, Abschnitt 6.3). Da azyklische WFSAs die Zwillingseigenschaft besitzen, ist es stets möglich, die Constraint-Werte auf Äquivalenz zu prüfen.

Ein Constraint-Wert, der als WFSA repräsentiert ist, hat eine ähnliche Form wie ein entfalteter FSA im Kontext des Regular-Membership-Constraints. So ist es möglich, für einen Constraint-Wert einen entfalteten FSA

zu erstellen, der in Schichten unterteilt ist (vgl. Abschnitt 3.1.3). Für ein SCSP über dem Booleschen Semiring kann dann der Propagierungsalgorithmus aus Pesant (2004) herangezogen werden, um generalisierte Kantenkonsistenz herzustellen. Wenn ein WFSA, der einen Constraint-Wert repräsentiert, deterministisch ist, ist es aus Effizienzgründen vorteilhaft, wenn er zusätzlich minimal ist. Es wäre aber auch möglich, von nichtdeterministischen WFSAs auszugehen, was unter Umständen zu einer Effizienzsteigerung führen kann (vgl. Lagerkvist, 2008). Alternativ kann auf die Repräsentationsform und den Propagierungsalgorithmus aus Cheng und Yap (2010) zurückgegriffen werden. Falls ein SCSP nicht über den Booleschen Semiring definiert ist, können hingegen die Repräsentationsformen und die Propagierungsänsatze zum Cost-Regular-Constraint (siehe Demassey et al., 2006) und dem Multi-cost-Regular-Constraint (siehe Menana und Demassey, 2009) herangezogen werden.

3.4 Erweiterung des SCSP-Frameworks mithilfe von FSMs

Im Folgenden werden Erweiterungen des SCSP-Frameworks vorgestellt, die innerhalb der Darstellung durch FSMs umgesetzt sind. Diese sind demnach nur im Kontext endlicher Techniken zu verstehen und umfassen die Einbeziehung eines abzählbar unendlichen Wertebereichs, die simultane Repräsentation mehrerer Constraint-Probleme und -Systeme und die Verwendung von quantifizierten regulären Constraints.

3.4.1 Unendlicher Wertebereich

Das SCSP-Framework setzt voraus, dass der Wertebereich endlich ist (siehe Abschnitt 2.3.3 Definition 18). In diesem Abschnitt wird beschrieben, wie das SCSP-Framework um einen unendlichen Wertebereich erweitert werden kann. Die grundlegende Idee ist hierbei, wie in Golden und Pang (2006) den unendlichen Wertebereich durch eine unendliche rationale Sprache zu repräsentieren. Wenn dieser beispielsweise aus allen möglichen Zeichenketten über dem Alphabet Δ besteht, kann er über die unendliche rationale Sprache Δ^* dargestellt werden. Er ist in diesem Fall abzählbar unendlich, da die Menge

aller Worte über einem endlichen Alphabet stets abzählbar ist. Es existieren jedoch unendliche Wertebereiche, die nicht über eine rationale Sprache repräsentiert und somit nicht über reguläre Mittel beschrieben werden können. Ein Beispiel hierfür ist ein Wertebereich aus Zahlenpalindromen. Hier wird demnach angenommen, dass ein Constraint-System einen unendlichen Wertebereich besitzen kann, solange er mit regulären Mitteln beschreibbar ist.

Gegeben ein Constraint-System $CS = (S, D, V)$ mit einem unendlichen Wertebereich $D = \{d_1, d_2, ...\}$, der über reguläre Mittel beschrieben werden kann. Des Weiteren sei eine bijektive Abbildung $\varphi : D \to \Delta^*$ angenommen, die gegeben ist durch $\varphi(d_r) = d'_r$ mit $d'_r \in \Delta^*$ für jedes r mit $1 \leq r$. Dann ist die unendliche rationale Sprache, die den unendlichen Wertebereich widerspiegelt, definiert durch:

$$L(D) = \{\varphi(d_1), \varphi(d_2), ...\} = \{d'_1, d'_2, ...\} \tag{3.21}$$

Da der Wertebereich hier durch eine unendliche rationale Sprache umgesetzt ist, werden die Variablenmengen durch unendliche rationale Sprachen und die Constraint-Werte durch rationale Potenzreihen mit unendlichem Support repräsentiert. Hierbei helfen die Markierungszeichen $\triangleleft$ und $\triangleright$ bzw. $\blacktriangleleft$ und $\blacktriangleright$ dabei, eine eindeutige Zuordnung von den Werten zu den Variablen zu gewährleisten. Es ist also nicht nötig, wie in Hansen und Andersen (2007) die Wertezuordnung über ein komplexes Alphabet zu realisieren.

Das Lösen eines Constraint-Problems mithilfe von FSMs, wie es in Abschnitt 3.3.4 beschrieben ist, ist mit einem unendlichen Wertebereich ohne weitere Änderungen möglich. Lediglich die Zeitkomplexitäten für die Äquivalenztransformationen Epsilon-Entfernung und Minimierung unterscheiden sich, da nun mit zyklischen FSMs umgegangen werden muss. So hat die Epsilon-Entfernung in diesem Fall eine Zeitkomplexität von $O(|Q||E|+|Q|^2 log(|Q|))$ (siehe Mohri, 2001, Abschnitt 3.2) und die Minimierung eine Zeitkomplexität von $O(|E|log(|Q|))$, wobei Q die Menge von Zuständen und E die Menge von Übergängen bezeichnet (siehe Mohri, 2009, Abschnitt 6.4).

Es soll ein Beispiel für die Repräsentation eines unendlichen Wertebereichs angeführt werden. Hierzu wird an den Beispielwertebereich aus Abschnitt 3.2.1 angeknüpft. Dort ist ein Wertebereich aus natürlichen Zahlen gegeben. Hier wird angenommen, dass dieser nicht begrenzt und somit

$D = \mathbb{N}_0^+$ definiert ist. Die Menge der natürlichen Zahlen kann über eine rationale Sprache über dem Alphabet $\Delta = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$, das Ziffern enthält, repräsentiert werden. Der Wertebereich kann über die folgende rationale Sprache repräsentiert werden:

$$L(D) = 0 \cup (1 \cup 2 \cup 3 \cup 4 \cup 5 \cup 6 \cup 7 \cup 8 \cup 9)(0 \cup 1 \cup 2 \cup 3 \cup 4 \cup 5 \cup 6 \cup 7 \cup 8 \cup 9)^* \quad (3.22)$$

Der entsprechende FSA $A(D)$ ist in Abbildung 3.14 gegeben. Die natürliche Zahl n zu einer beliebigen Zeichenkette $d_{k-1}...d_1d_0$ ist definiert durch: $n = \sum_{i=0}^{k_1} 10^i d_i = d_0 + 10d_1 + 10^2d_2 + ... + 10^{k-1}d_{k-1}$. Zeichenketten, die mehr als ein Zeichen enthalten, dürfen hierbei nicht mit dem Zeichen '0' beginnen, da sonst einzelne natürliche Zahlen über verschiedene Zeichenketten dargestellt wären. Der Wertebereich ließe sich auch über eine rationale Sprache repräsen-

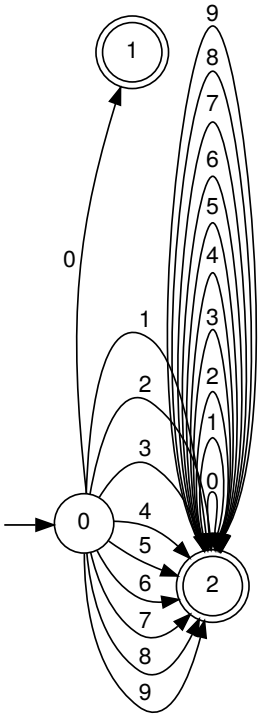


Abbildung 3.14: Unendlicher Wertebereich.

tieren, die auf einer anderen Zeichenkettencodierungen basiert. So ließe sich auch eine binäre Codierung über dem Alphabet $\Delta = \{1, 0\}$ realisieren. Die natürliche Zahl zu einer Zeichenkette $d_{k-1}...d_1d_0$ wäre dann definiert durch: $n = \sum_{i=0}^{k_1} 2^i d_i = d_0 + 2d_1 + 2^2d_2 + ... + 2^{k-1}d_{k-1}$. Eine unäre Codierung über dem

Alphabet $\Delta = \{1\}$ wäre auch möglich. Die natürliche Zahl entspräche dann der Länge der entsprechenden Zeichenkette.

Zusammenfassend kann ein unendlicher Wertebereich innerhalb eines Constraint-Systems umgesetzt werden, wenn es möglich ist, diesen als unendliche rationale Sprache zu repräsentieren. Jedoch ist die Formulierung von Constraints innerhalb dieser Erweiterung eingeschränkt. Es ist beispielsweise nicht möglich zu fordern, dass zwei Variablen den gleichen Wert besitzen. Diese Forderung müsste über eine rationale Potenzreihe formuliert werden, deren Support die Kopiersprache darstellt. Der Support wäre in diesem Fall nicht regulär. Aus dem gleichen Grund ist es nicht möglich zu fordern, dass der Wert einer Variable das Ergebnis der Konkatenation der Werte anderer Variablen ist. So lassen sich beispielsweise nicht alle Constraints formulieren, die in Golden und Pang (2006) vorgeschlagen werden. Des Weiteren unterliegen die Constraints den gleichen Einschränkungen, die für den Wertebereich selbst gelten. Angenommen, der Wertebereich sind die natürlichen Zahlen, dann kann z. B. nicht über einen Constraint gefordert werden, dass der Wert einer bestimmten Variable eine Palindromzahl ist. Solange der reguläre Rahmen nicht verlassen wird, können aber beliebige Constraints formuliert werden. Wenn der Wertebereich aus natürlichen Zahlen besteht, kann beispielsweise ein Maximum oder Minimum vorgegeben werden oder es kann postuliert werden, dass ein Wert eine gerade Zahl ist. Wenn der Wertebereich hingegen Zeichenketten umfasst, kann eine untere und obere Schranke für die Zeichenkettenlänge gefordert werden oder es kann verlangt werden, dass der Wert einer Variable eine bestimmte Zeichenkette enthalten muss.

3.4.2 Simultane Repräsentation

In diesem Abschnitt wird beschrieben, wie verschiedene SCSPs simultan repräsentiert werden können. Die simultane Repräsentation folgt der Idee des *MUltiply SEgmented CSP* (MUSE-CSP, Helzerman und Harper, 1996). Das MUSE-CSP ist eine Erweiterung des klassischen CSP, die es ermöglicht, verschiedene ähnliche Instanzen des binären CSPs, für die es schwierig wäre, sie als ein einzelnes CSP umzusetzen, kompakt zu repräsentieren. Diese werden zusammengefasst über einen gerichteten azyklischen Graphen repräsentiert, wobei eine Ecke des Graphen entweder einer Variable oder einer Menge von zusammengefassten Variablen entspricht. Eine simultane Repräsentation ist

hierbei genau dann sinnvoll, wenn sich die Instanzen sehr ähneln und viele Variablen zusammengefasst werden können. Die Lösung eines MUSE-CSPs entspricht letztlich der Lösung eines beliebigen CSPs, dass im MUSE-CSP enthaltenen ist.

In der vorliegenden Arbeit wird im Rahmen der FSMs ein Ansatz vorgestellt, bei dem mehrere SCSPs simultan repräsentiert und gelöst werden können. Das bedeutet, dass hier neben harten auch weiche Constraints einbezogen werden können. Des Weiteren wird ein Ansatz präsentiert, der nicht nur auf unäre und binäre Constraints beschränkt ist. Die Grundidee ist hierbei, Variablenmengen, die durch rationale Sprachen dargestellt sind, kompakt zu repräsentieren, indem diese vereinigt werden.

Im Folgenden wird erst auf die simultane Repräsentation von Constraint-Systemen und Constraints und darauf aufbauend auf die kompakte Darstellung mehrerer Constraint-Probleme eingegangen. Abschließend wird der Ansatz um ein selektives Zusammenlegen von Variablen erweitert, und es wird Constraint-Propagierung einbezogen.

Simultanes Constraint-System

Die simultane Repräsentation mehrerer unterschiedlicher Constraint-Systeme ist nur dann möglich, wenn sie über den gleichen c-Semiring definiert sind und wenn sie einen deckungsgleichen Wertebereich besitzen. Die einzelnen kompakt repräsentierten Constraint-Systeme werden als *Instanzen* eines simultanen Constraint-Systems bezeichnet. Im Folgenden kommt diese Begrifflichkeit auch bei den Bestandteilen eines Constraint-Systems zur Anwendung.

Die simultane Repräsentation mehrerer Constraint-Systeme beinhaltet die kompakte Darstellung der einzelnen Variablenmengen. Gegeben die Constraint-Systeme $CS_1 = (S, D, V_1)$,$CS_2 = (S, D, V_2)$,...,$CS_n = (S, D, V_n)$, wobei sich die Variablenmengen $V_1, V_2, \ldots V_n$ untereinander in ihrer Kardinalität unterscheiden, dann wird für die Simultanität der unterschiedlichen Variablenmengen $V_1|V_2|\ldots|V_n$ und für ein simultanes Constraint-System, das die einzelnen Constraint-Systeme kompakt repräsentiert, $(S, D, V_1|V_2|\ldots|V_n)$ geschrieben. Hierbei spielt die Reihenfolge der Variablenmengen innerhalb der kompakten Repräsentation keine Rolle. Die simultane Variablenmenge kann

nun über eine rationale Sprache definiert werden:

$$L(V_1|V_2|\ldots|V_n) = L(V_1) \cup L(V_2) \cup \cdots \cup L(V_n) \tag{3.23}$$

Als Beispiel ist für $|V_1| = 1, |V_2| = 2, ..., |V_n| = n$ mit $D = \{d_1, d_2, ..., d_m\}$ der FSA $A(V_1|V_2|\ldots|V_n)$ in Abbildung 3.15 skizziert.

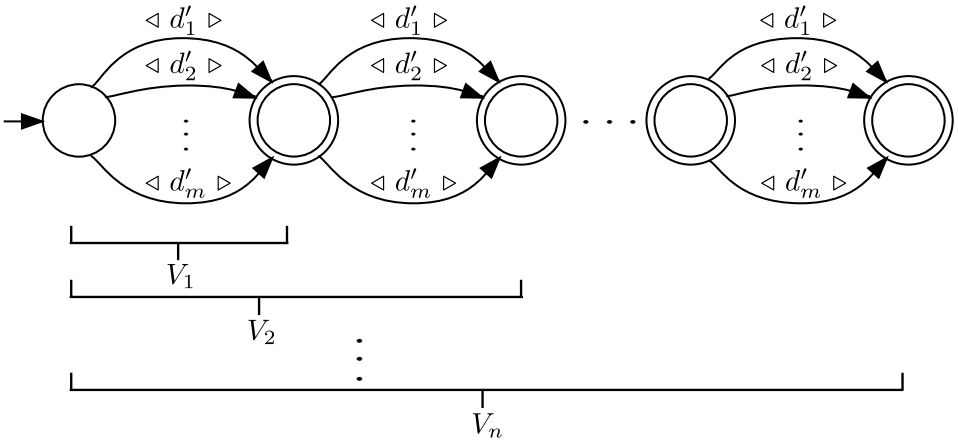


Abbildung 3.15: Simultane Repräsentation.

Es ist auch möglich, Variablenteilmengen simultan zu repräsentieren. Wenn die Variablenmengen $M_1 \subseteq V_1$,$M_2 \subseteq V_2$,...,$M_3 \subseteq V_n$ gegeben sind und wenn diese durch die rationalen Sprachen $L(M_1)$,$L(M_2)$,...,$L(M_n)$ dargestellt sind, dann ist die simultane Variablenmenge analog definiert durch:

$$L(M_1|M_2|\ldots|M_n) = L(M_1) \cup L(M_2) \cup \cdots \cup L(M_n) \tag{3.24}$$

Zwei simultane Variablenmengen eines simultanen Constraint-Systems können, wie in dem Ausdruck 3.8 in Abschnitt 3.2.1 beschrieben, vereinigt werden, wenn man dort für I und I' die entsprechenden simultanen Variablenmengen einsetzt. So ist die Vereinigung $L(M_{1_1}|M_{1_2}|\ldots|M_{1_n}) \sqcup L(M_{2_1}|M_{2_2}|\ldots|M_{2_n})$ die rationale Sprache $L(M_1|M_2|\ldots|M_n)$ mit $L(M_1|M_2|\ldots|M_n) = L(M_{1_1} \cup M_{2_1}|M_{1_2} \cup M_{2_2}|\ldots|M_{1_n} \cup M_{2_n}) = (L(M_{1_1}) \sqcup L(M_{2_1})) \cup (L(M_{1_2}) \sqcup L(M_{2_2})) \cup \cdots \cup (L(M_{1_n}) \sqcup L(M_{2_n}))$.

Auch die Tupel-Projektion kann analog zu dem Ausdruck 3.14 aus Abschnitt 3.3.1 auf die zusammengefasste Darstellungsform angewendet werden. Dort muss lediglich für I und I' die entsprechenden simultanen Variablenmengen einsetzt werden. Gegeben die simultanen Variablenmengen

$L(I_1|I_2|\ldots|I_n)$ und $L(I'_1|I'_2|\ldots|I'_n)$, dann ist die Projektion der Wertetupel von $L(I_1|I_2|\ldots|I_n)$ auf $L(I'_1|I'_2|\ldots|I'_n)$ die rationale Zeichenkettentransduktion $L(\downarrow^{I_1|I_2|\ldots|I_n}_{I'_1|I'_2|\ldots|I'_n})$ mit $L(\downarrow^{I_1|I_2|\ldots|I_n}_{I'_1|I'_2|\ldots|I'_n}) = L(\downarrow^{I_1}_{I'_1}) \cup L(\downarrow^{I_2}_{I'_2}) \cup \cdots \cup L(\downarrow^{I_n}_{I'_n})$.

Simultane Constraints

Mehrere Constraints können unter einigen Einschränkungen kompakt dargestellt werden. Die einzelnen kompakt repräsentierten Constraints werden im Folgenden als Instanzen eines simultanen Constraints bezeichnet.

Es seien die Constraints $(def_1,con_1),(def_2,con_2),\ldots,(def_n,con_n)$ mit $con_1 \subset V_1, con_2 \subset V_2, \ldots, con_n \subset V_n$ gegeben, die über die einzelnen Instanzen eines simultanen Constraint-Systems $(S, D, V_1|V_2|\ldots|V_n)$ definiert sind. Dann wird für den simultanen Constraint $\widetilde{c}$, der diese Constraints kompakt darstellt, $(def_1|def_2|\ldots|def_n, con_1|con_2|\ldots|con_n)$ geschrieben. Dessen Typ kann durch die rationale Sprache $L(con_1|con_2|\ldots|con_n)$ und dessen Wert durch die folgende rationale Potenzreihe repräsentiert werden:

$$L(def_1|def_2|\ldots|def_n) = L(def_1) \cup L(def_2) \cup \cdots \cup L(def_n) \tag{3.25}$$

wobei für alle nicht leeren Wertetupel t und für jedes beliebige $L(def_k)$ mit $1 \leq k \leq n$ und $t \in L(\downarrow^{con_k}_{con_k})$ und für jedes beliebige $L(def_j)$ mit $1 \leq j \leq n$ und $t \in L(\downarrow^{con_j}_{con_j})$ gelten muss, dass $L(def_k)(t) = L(def_j)(t)$ ist. Denselben Wertetupeln dürfen demnach über die einzelnen Constraint-Werte keine unterschiedlichen c-Semiring-Elemente zugewiesen werden. Ansonsten könnte ein Constraint innerhalb der kompakten Repräsentation indirekt Einfluss auf einen anderen Constraint nehmen, indem c-Semiring-Elemente über die abstrakte Addition über die verschiedenen Constraint-Werte hinweg verrechnet werden. Schließlich ist der simultane Constraint $\widetilde{c}$ definiert durch:

$$\widetilde{c} = (L(def_1|def_2|\ldots|def_n), L(con_1|con_2|\ldots|con_n)) \tag{3.26}$$

Für zwei gegebene simultane Constraints lässt sich die Kombination definieren. Gegeben die beiden simultanen Constraints $\widetilde{c_1} = (L(def_{1_1}|def_{1_2}|\ldots|def_{1_n}), L(con_{1_1}|con_{1_2}|\ldots|con_{1_n}))$ und $\widetilde{c_2} = (L(def_{2_1}|def_{2_2}|\ldots|def_{2_n}), L(con_{2_1}|con_{2_2}|\ldots|con_{2_n}))$, dann ist deren Kombination $\widetilde{c_1} \otimes \widetilde{c_2}$ der Constraint $\widetilde{c} = (L(def_1|def_2|\ldots|def_n), L(con_1|con_2|\ldots|con_n))$. Der Typ des Constraints ist

definiert durch die rationale Sprache:

$$\begin{gathered} L(con_1|con_2|\ldots|con_n) = \\ L(con_{1_1}|con_{1_2}|\ldots|con_{1_n}) \sqcup L(con_{2_1}|con_{2_2}|\ldots|con_{2_n}) = \\ L(con_{1_1} \cup con_{2_1}|con_{1_2} \cup con_{2_2}|\ldots|con_{1_n} \cup con_{2_n}) \end{gathered} \tag{3.27}$$

und der Wert des Constraints durch die rationale Potenzreihe:

$$\begin{gathered} L(def_1|def_2|\ldots|def_n) = \\ \left(\begin{array}{c} Dom(L(\downarrow^{con_1|con_2|\ldots|con_n}_{con_{1_1}|con_{1_2}|\ldots|con_{1_n}}) \circ L(def_{1_1}|def_{1_2}|\ldots|def_{1_n}) \\ \cap \\ Dom(L(\downarrow^{con_1|con_2|\ldots|con_n}_{con_{2_1}|con_{2_2}|\ldots|con_{2_n}}) \circ L(def_{2_1}|def_{2_2}|\ldots|def_{2_n})) \end{array} \right) \end{gathered} \tag{3.28}$$

wobei für alle nicht leeren Wertetupel t und für jedes beliebige $L(def_{1_k})$ mit $1 \leq k \leq n$ und $t \in L(\downarrow^{con_k}_{con_k})$ und für jedes beliebige $L(def_{1_j})$ mit $1 \leq j \leq n$ und $t \in L(\downarrow^{con_j}_{con_j})$ gelten muss, dass $\mathcal{L}_1 = Dom(L(\downarrow^{con_k}_{con_{1_k}}) \circ L(def_{1_k}))$ und $\mathcal{L}_2 = Dom(L(\downarrow^{con_k}_{con_{2_k}}) \circ L(def_{2_k}))$ mit $\mathcal{L}_1(t) = \mathcal{L}_2(t)$. Das bedeutet, dass einem Wertetupel für $L(con_1|con_2|\ldots|con_n)$ über die einzelnen simultan repräsentierten Constraint-Werte von $\widetilde{c_1}$ oder $\widetilde{c_2}$ keine unterschiedlichen c-Semiring-Elemente zugewiesen werden dürfen. Ansonsten könnte ein Constraint innerhalb der kompakten Repräsentation indirekt Einfluss auf einen anderen Constraint nehmen. Wenn ein simultaner Constraint ausschließlich Wertetupel der gleichen Stelligkeit besitzt, ist diese Bedingung stets erfüllt.

Des Weiteren lässt sich über einen gegebenen simultanen Constraint die Projektion definieren. Gegeben die Variablenmengen $I_1 \subseteq con_1$,$I_2 \subseteq con_2$,..., $I_n \subseteq con_n$, die durch die rationalen Sprachen $L(I_1)$,$L(I_2)$,...,$L(I_n)$ repräsentiert sind, dann ist die Projektion eines simultanen Constraints auf eine simultane Variablenmenge, geschrieben $(L(def_1|def_2|\ldots|def_n), L(con_1|con_2|\ldots|con_n)) \Downarrow_{I_1,I_2,\ldots;I_n}$, definiert als der simultane Constraint $(L(def'_1|def'_2|\ldots|def'_n), L(con'_1|con'_2|\ldots|con'_n))$. Die rationale Sprache, die dessen Typ repräsentiert ist gegeben durch:

$$L(con'_1|con'_2|\ldots|con'_n) = L(I_1|I_2|\ldots|I_n) \tag{3.29}$$

und die rationale Potenzreihe, die dessen Wert umsetzt, durch:

$$\begin{gathered} L(def'_1|def'_2|\ldots|def'_n) = \\ Range(L(def_1|def_2|\ldots|def_n) \circ L(\downarrow^{con_1|con_2|\ldots|con_n}_{I_1|I_2|\ldots|I_n})) \end{gathered} \tag{3.30}$$

wobei für alle nicht leeren Wertetupel t und für jedes beliebige $L(def_k)$ mit $1 \leq k \leq n$ und $t \in L(\downarrow_{I_k}^{I_k})$ und für jedes beliebige $L(def_j)$ mit $1 \leq j \leq n$ und $t \in L(\downarrow_{I_j}^{I_j})$ gelten muss, dass $\mathcal{L}_1 = Dom(L(\downarrow_{con_k}^{I_k}) \circ L(def_k))$ und $\mathcal{L}_2 = Dom(L(\downarrow_{con_j}^{I_j}) \circ L(def_j))$ mit $\mathcal{L}_1(t) = \mathcal{L}_2(t)$. Das bedeutet, dass ähnlich zur Kompositionsoperation einem Wertetupel für $L(I_1|I_2|\ldots|I_n)$ über die einzelnen simultan repräsentierten Constraint-Werte ($L(def_1|def_2|\ldots|def_n)$) keine unterschiedlichen c-Semiring-Elemente zugewiesen werden dürfen. Ansonsten könnte ein Constraint innerhalb der kompakten Repräsentation indirekt Einfluss auf einen anderen Constraint nehmen. Wie auch bei der Kompositionsoperation ist diese Bedingung stets erfüllt, wenn der simultane Constraint ausschließlich Wertetupel der gleichen Stelligkeit besitzt.

Es ist auch möglich, eine Instanz eines simultanen Constraints auf eine Variablenmenge zu projizieren. Gegeben den simultanen Constraint $(def_1|def_2|\ldots|def_n, con_1|con_2|\ldots|con_n)$ und die Variablenmenge $I \subseteq con_k$ mit $1 \leq k \leq n$, dann ist die Projektion einer Instanz eines simultanen Constraints auf diese Variablenmenge, geschrieben $(L(def_1|def_2|\ldots|def_n), L(con_1|con_2|\ldots|con_n)) \Downarrow_I^{con_k}$, definiert als der Constraint $(L(def'_k), L(con'_k))$. Die rationale Sprache, die dessen Typ repräsentiert ist definiert durch:

$$L(con'_k) = L(I) \tag{3.31}$$

und die rationale Potenzreihe, die dessen Wert umsetzt, durch:

$$\begin{gathered} L(def'_k) = \\ Range(L(def_1|def_2|\ldots|def_n) \circ L(\downarrow_I^{con_k})) \end{gathered} \tag{3.32}$$

Simultanes Constraint-Problem

Mehrere Constraint-Probleme können kompakt dargestellt werden, wenn sie über unterschiedliche Constraint-Systeme definiert sind. Im Folgenden werden die einzelnen kompakt repräsentierten Constraint-Probleme als Instanzen eines simultanen Constraint-Problems bezeichnet.

Gegeben die Constraint-Probleme $P_1 = (C_1, con_1)$,$P_2 = (C_2, con_2)$,…, $P_n = (C_n, con_n)$ mit $con_1 \subset V_1$,$con_2 \subset V_2$,…,$con_n \subset V_n$, die über die einzelnen Instanzen eines simultanen Constraint-Systems $(S, D, V_1|V_1|\ldots|V_n)$ definiert sind. Dann wird für ein entsprechendes simultanes Constraint-Problem $(\widetilde{C}, con_1|con_2|\ldots|con_n)$ geschrieben, wobei $\widetilde{C}$ eine Menge von simultanen

Constraints bezeichnet, die aus den Constraints der einzelnen Instanzen des simultanen Constraint-Problems gebildet wird. Ein simultaner Constraint in $\widetilde{C}$ hat die Form $(def_1|def_2|\ldots|def_n, con'_1|con'_2|\ldots|con'_n)$, wenn $(def_1, con'_1) \in C_1, (def_2, con'_2) \in C_2, \ldots, (def_n, con'_n) \in C_n$. Darüber hinaus müssen für die Bildung von $\widetilde{C}$ die einzelnen Constraint-Mengen $C_1, C_2, \ldots, C_n$ die gleiche Kardinalität besitzen, da jeder simultane Constraint aus jeweils einem Constraint der verschiedenen Instanzen des simultanen Constraint-Problems besteht. Um auch Constraint-Mengen mit unterschiedlicher Größe zu behandeln, können zusätzliche Universal-Constraints angenommen werden. Diese verändern die Lösungen der einzelnen Constraint-Probleme nicht. Die Größe der Menge $\widetilde{C}$ entspricht dann der Kardinalität der Constraint-Menge, die die meisten Constraints enthält: $|\widetilde{C}| = max(|C_1|, |C_2|, \ldots, |C_n|)$. So ist es beispielsweise möglich, für eine Instanz des simultanen Constraint-Problems einen zusätzlichen Universal-Constraint anzunehmen, der ihren Typ besitzt. Dieser Universal-Constraint kann innerhalb mehrerer simultaner Constraints Anwendung finden. Wenn beispielsweise die Constraint-Menge $\{(def_{a_1}, con_{a_1}), (def_{a_2}, con_{a_2}), (def_{a_3}, con_{a_3})\}$ und die Constraint-Menge $\{(def_b, con_b)\}$ gegeben sind, dann kann ein Universal-Constraint (def_u, con_u) angenommen werden, damit aus den beiden Mengen die simultane Constraint-Menge $\{(def_{a_1}|def_b, con_{a_1}|con_b), (def_{a_2}|def_u, con_{a_2}|con_u), (def_{a_3}|def_u, con_{a_3}|con_u)\}$ gebildet werden kann.

Die Constraint-Menge $\widetilde{C_L}$, die mithilfe von rationalen Sprachen und Potenzreihen repräsentiert ist und mit $\widetilde{C}$ korrespondiert, ist definiert durch:

$$\widetilde{C_L} = \bigcup_{(def_1|\ldots|def_n, con'_1|\ldots|con'_n) \in \widetilde{C}} (L(def_1|\ldots|def_n), L(con'_1|\ldots|con'_n)) \tag{3.33}$$

wobei die Beschränkungen zur Bildung eines simultanen Constraints stets erfüllt sein müssen. Dies ist beispielsweise immer dann möglich, wenn alle Constraints der Instanzen des simultanen Constraint-Problems den gleichen Wert besitzen. Der Typ des simultanen Constraint-Problems kann analog zu dem Typ eines simultanen Constraints über die rationale Sprache $L(con_1|con_2|\ldots|con_n)$ repräsentiert werden. Dann ist die Repräsentation eines simultanen Constraint-Problems mithilfe von rationalen Sprachen und Potenzreihen definiert durch:

$$\widetilde{P} = (\widetilde{C_L}, L(con_1|con_2|\ldots|con_n)) \tag{3.34}$$

Über die Kombinations- und Projektionsoperation lässt sich die Lösung eines simultanen Constraint-Problems definieren. Gegeben sei das simultane Constraint-Problem $\widetilde{P} = (\widetilde{C_L}, L(con_1|con_2|\ldots|con_n))$, dann ist die Lösung des simultane Constraint-Problems, geschrieben $Sol(\widetilde{P})$, der simultane Constraint:

$$\begin{gathered}(L(def_1|def_2|\ldots|def_n), L(con_1|con_2|\ldots|con_n)) = \\ (\otimes_{\widetilde{c}\in\widetilde{C_L}}\,\widetilde{c}) \Downarrow_{con_1|con_2|\ldots|con_n}\end{gathered} \tag{3.35}$$

mit $Sol(\widetilde{P}) \Downarrow_{con_1}^{con_1} = Sol(P_1), Sol(\widetilde{P}) \Downarrow_{con_2}^{con_2} = Sol(P_2), \ldots, Sol(\widetilde{P}) \Downarrow_{con_n}^{con_n} = Sol(P_n)$. Eine korrekte Berechnung der Lösung ist hierbei nur dann möglich, wenn die Bedingungen zur Kombinations- und Projektionsoperation für simultane Constraints erfüllt sind.

Zudem lässt sich auch die beste Konsistenzstufe für simultane Constraint-Probleme definieren:

$$(\otimes_{\widetilde{c}\in\widetilde{C_L}}\,\widetilde{c}) \Downarrow_{\underbrace{\varnothing|\varnothing|\ldots|\varnothing}_{n}} \tag{3.36}$$

Hierbei werden die c-Semiring-Elemente der einzelnen Instanzen über die abstrakte Addition miteinander verrechnet, sodass bei der Berechnung keine Trennung mehr zwischen den verschiedenen Instanzen gegeben ist. So wird die beste Konsistenzstufe über alle Instanzen des simultanen Constraint-Problems hinweg berechnet.

Selektives Zusammenlegen von Variablen

Die simultane Repräsentation von Constraints, Constraint-Problemen und Constraint-Systemen ist bisher sehr eingeschränkt. Dies zeigt sich in drei Punkten: 1) Angenommen, die Stelligkeit mehrerer Constraints, die kompakt repräsentiert sind, stimmen überein, dann haben sie auch den gleichen Wert und es ist nicht möglich, je nach Constraint-System eine andere Beschränkung zu realisieren; 2) Angenommen, mehrere Constraint-Probleme, die kompakt repräsentiert sind und deren Constraints dieselbe Stelligkeit aufweisen, beziehen sich auf Variablen, die in den jeweiligen Variablenordnungen in ihrer Position übereinstimmen, dann haben die einzelnen Constraint-Probleme stets einen Lösungs-Constraint mit dem gleichen Wert. 3) Die Instanzen eines simultanen Constraint-Systems müssen über Variablenmengen mit unterschiedlicher Kardinalität definiert sein.

Über die bisherige kompakte Darstellungsform der Constraint-Systeme

werden Variablen zusammengelegt, die in den jeweiligen Variablenordnungen bezüglich der Position übereinstimmen. Der Begriff des Zusammenlegens bezieht sich hierbei auf die Repräsentation als deterministische FSM, bei der Präfixe und Suffixe verschmolzen sind. Im Folgenden wird eine Erweiterung der simultanen Darstellung vorgeschlagen, die die oben aufgeführten Einschränkungen angeht, indem ein selektives Verschmelzen der Variablen ermöglicht wird. Selektiv meint hier, dass ausgedrückt werden kann, dass zwei Variablen nicht zusammengelegt werden sollen.

Um das Zusammenlegen von Variablen zu steuern, wird angenommen, dass mit einer Variable bestimmte Informationen verknüpft sein können. Hierfür soll ein Beispiel gegeben werden. Hierzu wird an die Beispiele zur syntaktischen Analyse aus den vorherigen Abschnitten angeknüpft. Angenommen jedes Wort eines Satzes wird durch eine Variable repräsentiert und angenommen der Wertebereich der Variablen besteht aus syntaktischen Labeln. Dann können den Wörtern und somit auch den Variablen bestimmte Informationen wie z. B. die Wortart, die Lemmaform oder morphologische Merkmale zugeordnet werden. Wenn ein Satz beispielsweise mehrere morphologische Lesarten besitzt, kann für jede Lesart ein Constraint-System gebildet werden, dass mit den einzelnen Variablen entsprechende Informationen verknüpft. In der simultanen Repräsentation würden dann nur die Variablen zusammengelegt werden, die die gleiche Information teilen.

Wenn bestimmte Informationen mit den Variablen eines Constraint-Systems verknüpft sind, sind diese über die Variablenpositionen direkt zugänglich. Innerhalb der simultanen Repräsentation ist dies hingegen nicht mehr der Fall. Es können verschiedene Informationen mit den Variablen unterschiedlicher Constraint-Probleme verknüpft sein, die sich an der gleichen Position der jeweiligen Variablenordnung befinden. Im Folgenden wird die Darstellung der Variablen eines Constraint-Systems so angepasst, dass auch bei der simultanen Repräsentation ein direkter Zugriff auf die mit den Variablen verknüpfte Informationen möglich ist.

Wenn Informationen mit den Variablen verknüpft sind, ist ein Constraint-System im Folgenden definiert als ein Quadrupel $CS = (S, D, V^\tau, I)$ mit $V^\tau = \{v_1^\tau, v_2^\tau, \ldots, v_n^\tau\}$ und $v_1^\tau \prec v_2^\tau \prec \cdots \prec v_n^\tau$ und $I = \{i_1, i_2, \ldots, i_m\}$. Hier bezeichnet I die Menge an möglichen Informationen, die mit den Variablen in $V = \{v_1, v_2, \ldots, v_n\}$ verknüpft sein können, und es wird V^τ und $v_1^\tau, v_2^\tau, \ldots, v_n^\tau$ geschrieben, um auszudrücken, dass die Variablen in V über die Abbildung τ

mit Informationen in I verbunden sind. Die surjektive Abbildung $\tau : V \to I$ ist hierbei definiert durch $\tau(v_r) = i_k$ mit $i_k \in I$ für jedes r mit $1 \leq r \leq n$. Nun sei eine bijektive Abbildung $\sigma : I \to \Gamma^*$ mit $\Gamma \subset \Sigma$ gegeben, die die Informationen auf ihre Zeichenkettenrepräsentation abbildet und definiert ist durch $\sigma(i_r) = i'_r$ mit $i'_r \in \Gamma^*$ für jedes r mit $1 \leq r \leq m$. Dann kann eine Information i_r durch eine einelementige rationale Sprache repräsentiert werden:

$$L(i_r) = \{\sigma(i_r)\} = \{i'_r\} \tag{3.37}$$

Alle verknüpften Informationen, die für ein Constraint-System möglich sind, können durch eine rationale Sprache dargestellt werden, die der Vereinigung der einelementigen rationalen Sprachen, die die einzelnen Informationen repräsentieren, entspricht:

$$L(I) = L(i_1) \cup L(i_2) \cup ... \cup L(i_m) \tag{3.38}$$

Dann kann eine Variable v_r^τ mit $1 \leq r \leq n$ zusammen mit ihrer verknüpften Information durch folgende rationale Sprache repräsentiert werden:

$$L(v_r^\tau) = \{\sigma(\tau(v_r^\tau))\} \diamond L(D) \tag{3.39}$$

wobei das Zeichen $\diamond \in \Sigma$ mit $\{\diamond\} \cap (\Delta \cup \Gamma) = \varnothing$ die verknüpften Informationen von den Werten des Wertebereichs abtrennt.

Analog zu dem Ausdruck 3.5 aus Abschnitt 3.2.1, kann die Variablenmenge V^τ zusammen mit ihrer verknüpften Information über eine rationale Sprache dargestellt werden. Zudem können analog zu dem Ausdruck 3.6 aus Abschnitt 3.2.1 über die Markierungszeichen ◀ und ▶ auch Teilmengen von V^τ repräsentiert werden und es können analog zu dem Ausdruck 3.8 aus Abschnitt 3.2.1 Teilmengen vereinigt werden.

Auch bei der Repräsentation der Wertetupel müssen die Informationen berücksichtigt werden, die mit den Variablen verknüpft sind. Gegeben den Wertetupel $t = (d_1, d_2, ..., d_n)$ und die Informationen $i_1, i_2, ..., i_n \in I$, die für die einzelnen Komponenten des Wertetupels möglich sind und die durch die Zeichenketten $L(i_1), L(i_2), ..., L(i_n)$ repräsentiert sind. Dann ist die Zeichenkette, die den Wertetupel zusammen mit den Informationen dargestellt, de-

finiert durch:

$$L(t) \;=\; \triangleleft\, L(i_1) \diamond L(d_1) \,\triangleright\triangleleft\, L(i_2) \diamond L(d_2) \,\triangleright \cdots \triangleleft\, L(i_n) \diamond L(d_n) \,\triangleright \qquad (3.40)$$

Ein Wertetupel ist zu einer Menge von Variablen kompatibel, wenn die Informationen im Wertetupel mit den einzelnen Informationen der Variablen übereinstimmen. Analog zu dem Ausdruck 3.14 aus Abschnitt 3.3.1 kann die Tupel-Projektion auch auf solche Wertetupel angewendet werden.

Die Darstellung der Constraints soll auf die neue Repräsentationsform der Variablen und Wertetupel angepasst werden. Es sei der Constraint (def, con^{τ}) gegeben, dann kann dessen Typ durch die rationale Sprache $L(con^{\tau})$ dargestellt werden, die die Variablenmenge zusammen mit den verknüpften Informationen repräsentiert. Der Wert des Constraints kann hingegen als eine rationale Potenzreihe umgesetzt werden, die Wertetupel aus $(\triangleleft\, L(I) \diamond L(D) \,\triangleright\,)^{|con^{\tau}|}$ auf Elemente in S abbildet und alle anderen Zeichenketten auf das Nullelement von S. So ist der Support der rationalen Potenzreihe $L(def)$ eine Teilmenge der folgenden rationalen Sprache:

$$L(def) \subseteq (\triangleleft\, L(I) \diamond L(D) \,\triangleright)^{|con^{\tau}|} \qquad (3.41)$$

Ein Wertetupel kann nur dann von solch einem Constraint erfüllt werden, wenn die Informationen der Variablen in $L(con^{\tau})$ mit den Informationen des Wertetupels kompatibel sind. Die Kombinationsoperation und die Projektionsoperation für simultane Constraints können hierbei ohne Änderung auf solche Constraints angewendet werden.

Im Folgenden soll auf die simultane Repräsentation in Kombination mit den verknüpften Informationen eingegangen werden. Es seien die Constraint-Systeme $CS_1 = (S,D,V_1^{\tau_1},I), CS_2 = (S,D,V_2^{\tau_2},I), \ldots, CS_n = (S,D,V_n^{\tau_n},I)$ gegeben, dann ist ein entsprechendes simultanes Constraint-System definiert durch $\widetilde{CS} = (S, D, V_1^{\tau_1}|V_2^{\tau_2}|\ldots|V_n^{\tau_n}, I)$. Die Repräsentation der simultanen Variablenmenge ist hier lediglich um die verknüpften Informationen ergänzt. Auch die simultanen Constraints und Constraint-Probleme können, wie in den vorhergehenden Abschnitten beschrieben, umgesetzt werden, wobei die Wertetupel und Variablenmengen lediglich um verknüpfte Informationen erweitert sind. Die in Abschnitt 3.4.2 definierten Einschränkungen für die Bildung von simultanen Constraints und für die Berechnung die Kombinations- und die Projektionsoperation beziehen sich dann auf Wertetupel, die sich bei

gleicher Werteabfolge in den verknüpften Informationen unterscheiden können. Hierdurch wird ein simultaner Constraint möglich, der je nach Constraint-System unterschiedliche Beschränkungen realisiert.

Wenn die Constraint-Systeme (S, D, V^{τ_1}, I),(S,D,V^{τ_2},I),...,(S, D, V^{τ_k}, I) mit $I = \{i_1, i_2, \ldots, i_m\}$ und $V = \{v_1, v_2, \ldots, v_n\}$ existieren, wobei über die Abbildungen $\tau_1, \tau_2, \ldots, \tau_k$ alle möglichen Verknüpfungskombinationen der Variablen zu den Informationen aus $I = \{i_1, i_2, \ldots, i_m\}$ gebildet werden. Dann steigt die Anzahl der Constraint-Systeme exponentiell mit $|V|$. Angenommen die einzelnen Constraint-Systeme werden über das simultane Constraint-System $(S, D, V^{\tau_1}|V^{\tau_2}|\ldots|V^{\tau_k}, I)$ dargestellt, dann kann die simultane Variablenmenge kompakt als FSA repräsentiert werden. Dieser ist in Abbildung 3.16 skizziert.

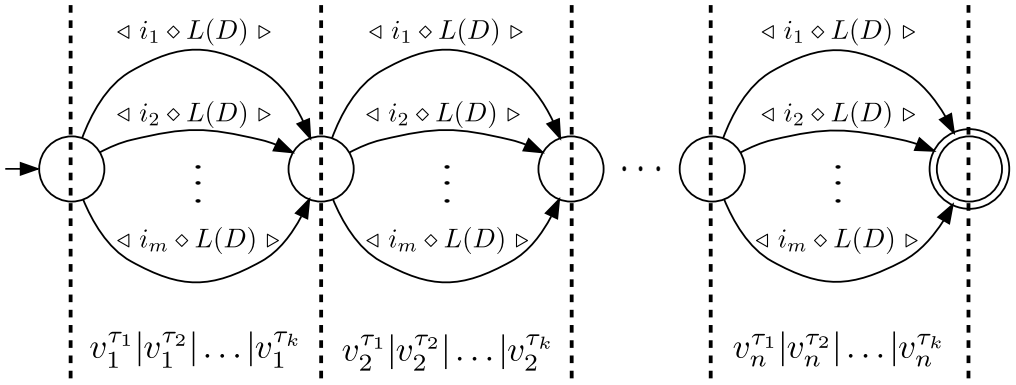


Abbildung 3.16: Repräsentation einer simultanen Variablenmenge mit verknüpften Informationen als FSA.

Innerhalb des MUSE-CSPs bezieht sich die simultane Repräsentation auf unäre und binäre harte Constraints. In der hier vorgestellten SCSP-Erweiterung lässt sich hingegen die kompakte Darstellung auf beliebige Instanzen des SCSP-Frameworks anwenden. So können neben dem klassischen CSP beispielsweise auch das gewichtete CSP oder das Fuzzy-CSP simultan repräsentiert werden. Des Weiteren ist der Ansatz nicht auf unäre und binäre Constraints beschränkt.

Es soll ein konkretes Beispiel für die simultane Repräsentation einer Variablenmenge und eines Constraint-Wertes gegeben werden. Hierzu wird an das Beispiel zur syntaktischen Analyse aus Abschnitt 3.2.2 angeknüpft. Dort bewegt sich das Beispiel im Kontext des gewichteten CSPs, wobei der Wertebereich $D = \{\texttt{S}, \texttt{SUBJ}, \texttt{OBJA}\}$ gegeben ist. Hier wird zusätzlich die Menge

an Informationen $I = \{\texttt{NN}, \texttt{VVFIN}\}$ herangezogen. Betrachtet man den natürlichsprachlichen Satz „Fliegen fliegen.“, dann kann bei Ignorierung der Groß- und Kleinschreibung für die beiden Wörter eine Lesart als finites Vollverb (VVFIN) oder als einfaches Nomen (NN) angenommen werden. Diese ambige Wortartenzuweisung kann über vier verschiedene Constraint-Systeme abgebildet werden, die auf der Variablenmenge $V = \{v_1, v_2\}$ mit $v_1 \prec v_2$ beruhen und simultan repräsentiert sind: $(S, D, V^{\tau_1}|V^{\tau_2}|V^{\tau_3}|V^{\tau_4}, I)$. Dann ist der FSA, der die simultane Variablenmenge $V^{\tau_1}|V^{\tau_2}|V^{\tau_3}|V^{\tau_4}$ mit ihren verschiedenen Wortartenverknüpfungsmöglichkeiten darstellt, in Abbildung 3.17 gegeben. In dieser kompakten Repräsentationsform teilen sich die Variablen

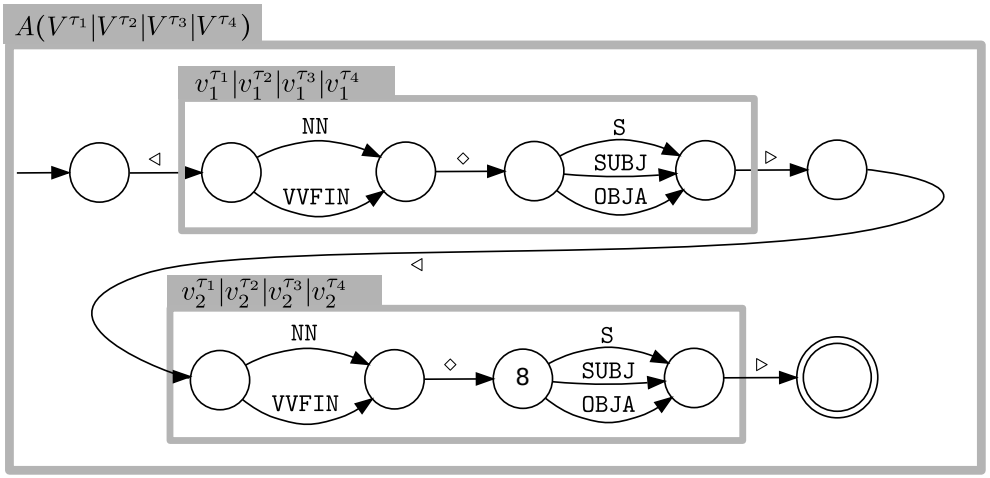


Abbildung 3.17: Beispiel für eine simultane Variablenmenge mit verknüpften Informationen als FSA.

$v_1^{\tau_1}, v_1^{\tau_2}, v_1^{\tau_3}, v_1^{\tau_4}$ die erste Variablenposition und die Variablen $v_2^{\tau_1}, v_2^{\tau_2}, v_2^{\tau_3}, v_2^{\tau_4}$ die zweite.

Ergänzend hierzu sei die weiche Beschränkung angenommen, die besagt, dass ein Nomen vor einem finiten Vollverb weder Akkusativobjekt (OBJA) noch Wurzelknoten eines Satzes (S) sein darf. Für das Verletzen dieser Einschränkung wird ein Strafgewicht von fünf vergeben. Wenn für jedes dieser Constraint-Systeme ein Constraint-Problem existiert, das alle Variablen im Skopus hat und jeweils einen Constraint enthält, der diese Beschränkung umsetzt. Dann ist der WFSA, der den Wert des simultanen Constraints repräsentiert, in Abbildung 3.18 gegeben. Da jedes enthaltene Constraint-Problem nur einen Constraint enthält, ist der Constraint-Wert gleichzeitig auch der Wert der Lösung des simultanen Constraint-Problems.

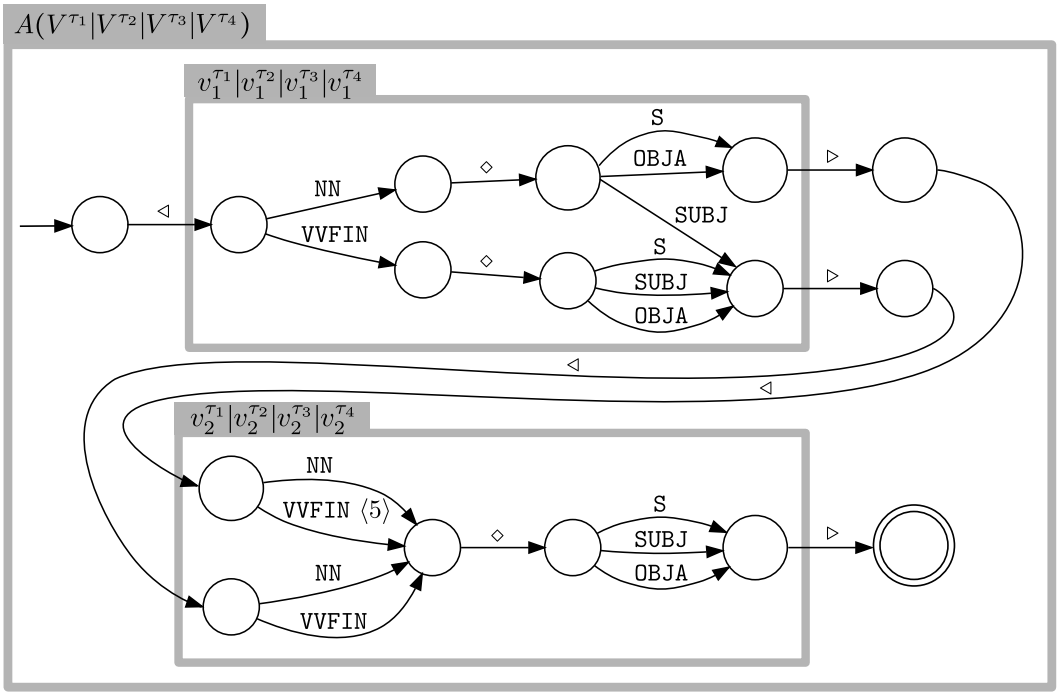


Abbildung 3.18: Beispiel für den Wert eines simultanen Constraints für eine Variablenmenge mit verknüpften Informationen als WFSA.

Constraint-Propagierung

In Helzerman und Harper (1996, Abschnitt 3) wird ein Constraint-Propagierungsalgorithmus zur Erreichung einer MUSE-CSP-bezogenen Kantenkonsistenz vorgeschlagen. Im Kontext des SCSPs lassen sich lokale Konsistenzregeln auch innerhalb der simultanen Repräsentation umsetzen. Dies ist der Fall, da die Regeln ausschließlich Bezug auf die Typen der Constraints nehmen und Teilprobleme mithilfe der Kombinations- und Projektionsoperation gelöst werden. Im Fall der kompakten Darstellung geschehen die Selektion und die Ersetzung von Constraints anhand der simultanen Typen. Die typisierten Positionen bezeichnen hier demnach simultane Variablenmengen. Bei der Anwendung einer lokalen Konsistenzregel müssen die Bedingungen für die Kombinations- und Projektionsoperation für die kompakte Darstellungsform erfüllt sein, um eine Lösung korrekt ermitteln zu können. Des Weiteren müssen unter Umständen Universal-Constraints eingeführt werden, um eine unterschiedliche Kardinalität von Constraint-Mengen auszugleichen. Um den Fix-Point innerhalb eines lokalen Konsistenzalgorithmus zu erkennen, muss

es nun möglich sein, simultane Constraint-Probleme auf Gleichheit zu prüfen. Auch das ist innerhalb der kompakten Repräsentationsform umsetzbar, da die Constraint-Werte azyklische WFSAs sind.

Interessant wäre eine Erweiterung des Ansatzes zur generalisierten Kantenkonsistenz in Pesant (2004) auf die kompakte Repräsentationsform. Hierzu müssten die mit den Variablen verknüpften Informationen einbezogen werden und es müsste mit Werten von simultanen Constraints umgegangen werden, die für unterschiedliche Skopen wohldefiniert sind und die sich auf verschiedene Constraint-Systeme beziehen. Da hier das Lösen und die Propagierung über rationale Operationen im Fokus stehen wird an dieser Stelle solch einer Erweiterung jedoch nicht nachgegangen.

3.4.3 Quantifizierte reguläre Constraints

In diesem Abschnitt wird gezeigt, wie es möglich ist, den Wert eines Constraints unabhängig von seinem Typ, der Variablenordnung und der Variablenmenge des Constraint-Systems zu definieren. Des Weiteren wird vorgeschlagen, Constraint-Werte direkt über reguläre prädikatenlogische Formeln zu erstellen. So müssen Constraints nicht mehr über den Umweg der Auflistung von gewichteten Wertetupeln konstruiert werden.

Quantifizierte Beschränkungen

Innerhalb eines Constraints geschieht die Zuordnung der Werte zu den Variablen über die Variablenordnung des Constraint-Systems. Hierzu muss die Anzahl der Komponenten der Wertetupel gleich der Anzahl der Variablen des Typs des Constraints sein. Es ist somit nicht möglich, einen Constraint zu definieren, der bei gleichbleibendem Wert für beliebige Typen wohldefiniert ist, wie es z. B. bei einem globalen Constraints der Fall ist. Um den Wert eines Constraints von dessen Typ zu entkoppeln, wird hier vorgeschlagen, die Zuordnung der Werte zu den Variablen anhand von Eigenschaften zu realisieren. Die *nativen Eigenschaften* wären hierbei:

- eine Variable zu sein;
- eine bestimmte Wertebelegung zu besitzen;
- bezüglich der Variablenordnung vor oder nach einer bestimmten Variable zu stehen;

- bezüglich der Variablenordnung im direktem Anschluss zu einer bestimmten Variable zu sein.

Es können quantifizierte Beschränkungen formuliert werden, die auf diese Eigenschaften Bezug nehmen und rationale Potenzreihen mit unendlichem Support spezifizieren.

Es soll im Kontext des Booleschen Semirings für eine existenziell quantifizierte und für eine universell quantifizierte Beschränkung ein konkretes Beispiel gegeben werden. Hierzu sei der Wertebereich $D = \{d_1, d_2, ..., d_m\}$ und eine Menge von Variablen $K = \{k_1, k_2, ..., k_n\}$ gegeben. Angenommen es liegt eine Einschränkung vor, die besagt, dass eine Variable $k_i \in K$ existiert, die die Eigenschaft hat, eine Belegung mit den Werten $d_{i_1}, d_{i_2}, ...d_{i_p} \in D$ zu besitzen. Dann kann für diese quantifizierte Beschränkung ein zyklischen WFSA angegeben werden, der in Abbildung 3.19 skizziert ist. Dieser erkennt

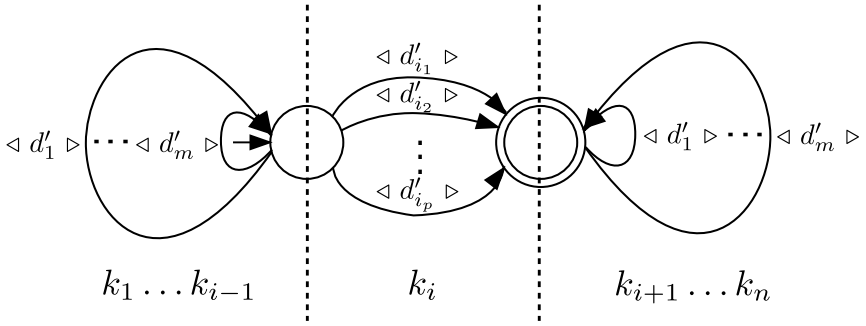


Abbildung 3.19: Existenziell quantifizierte Variable als WFSA.

die Wertetupel, die mindestens eine Komponente mit den Werten $d_{i_1}, d_{i_2}, ...$ oder d_{i_p} enthalten. Es sei nun eine Einschränkung angenommen, die besagt, dass für alle Variablen ausschließlich eine Belegung mit $d_{i_1}, d_{i_2}, ..., d_{i_p} \in D$ erlaubt ist. Dann kann für diese quantifizierte Beschränkung ebenso ein zyklischer WFSA angegeben werden, der in Abbildung 3.20 skizziert ist. Dieser erkennt die Wertetupel, deren Komponenten den Werten $d_{i_1}, d_{i_2}, ...$ oder d_{i_p} entsprechen.

Bei den quantifizierten Beschränkungen bildet der Wert eines Constraints demnach Wertetupel aus $(\triangleleft\, L(D)\, \triangleright)^+$ auf Elemente in S ab und alle anderen Zeichenketten auf das Nullelement von S. Ein konkreter Constraint mit dem Typ *con* besteht dann aus einem Paar $(\mathcal{X}_{def}, L(con))$, wobei die rationale Potenzreihe $\mathcal{X}_{def}$ dessen Wert und die rationale Sprache $L(con)$ dessen Typ

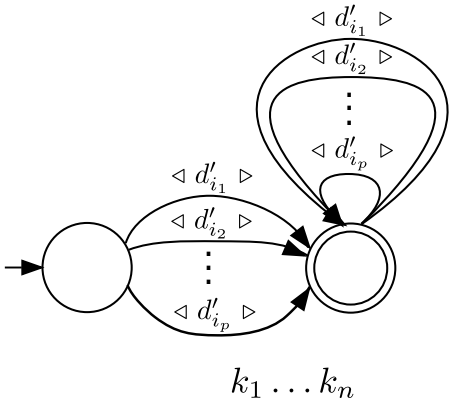


Abbildung 3.20: Universell quantifizierte Variable als WFSA.

repräsentiert. Der Support der rationalen Potenzreihe $\mathcal{X}_{def}$ ist hierbei eine Teilmenge der folgenden rationalen Sprache:

$$Support(\mathcal{X}_{def}) \subseteq (\triangleleft L(D) \triangleright)^+ \tag{3.42}$$

Der Constraint-Wert ist für jeden nicht leeren Skopus, aber auch für verschiedene Constraint-Systeme wohldefiniert. Im Folgenden wird solch ein Constraint als *quantifizierter regulärer Constraint* bezeichnet. Bei unterschiedlichen Constraint-Systemen kann ein entsprechender WFSA innerhalb verschiedener Constraints wiederverwendet werden, solange sie den gleichen c-Semiring und Wertebereich teilen. Dies ist zwar auch möglich, wenn der Wert eines Constraints durch eine rationale Potenzreihe mit endlichem Support repräsentiert wird. Aber die Wiederverwendbarkeit ist dann auf Constraints beschränkt, deren Typen die gleiche Kardinalität besitzen.

Die Kombinationsoperation und die Projektionsoperation aus Abschnitt 3.3 können ohne Änderung auf quantifizierte reguläre Constraints angewendet werden. Das Ergebnis dieser Operationen resultiert dann in einen Constraint, dessen Wert durch eine rationale Potenzreihe mit endlichem Support repräsentiert ist. Innerhalb eines Constraint-Problems sind keine zwei Constraints mit demselben Typ erlaubt. Entsprechende Constraints müssen daher initial zu einem Constraint zusammengefasst werden (siehe Abschnitt 3.2.3). Dies ist auch bei den quantifizierten regulären Constraints möglich. Aus dem Zusammenfassen resultiert ebenso ein quantifizierter regulärer Constraint, der für jeden Skopus wohldefiniert ist und der eine Konjunktion der

zugrunde liegenden quantifizierten Beschränkungen umsetzt.

Mithilfe der quantifizierten regulären Constraints lassen sich nur Bedingungen Umsetzen, die sich im regulären Rahmen befinden. So kann z. B. nicht gefordert werden, dass eine Belegung von Variablen unter Berücksichtigung einer entsprechenden Variablenordnung ein Palindrom bildet. Dies ist hingegen bei einem Constraint, dessen Wert durch eine rationale Potenzreihe mit endlichem Support repräsentiert ist, stets möglich, da dort die Anzahl an Wertetupelkomponenten immer endlich ist.

Bei quantifizierten regulären Constraints sind die Constraint-Werte durch azyklische WFSAs repräsentiert. So ist es nicht gewährleistet, dass die WFSAs die Zwillingseigenschaft besitzen. Somit sind eine Determinisierung und die Prüfung auf Äquivalenz nicht immer möglich. Über die Kombinations- und Projektionsoperation werden jedoch azyklische WFSAs berechnet. So ist für diese Constraint-Werte stets die Prüfung auf Äquivalenz möglich, was zur Erkennung des Fix-Point eines Constraint-Propagierungsalgorithmus ausreichend ist (vgl. Abschnitt 3.3.6).

Formulierung von quantifizierten Beschränkungen

Um komplexere quantifizierte Beschränkungen zu formulieren, können reguläre prädikatenlogische Formeln herangezogen werden. Auf diese Weise ist es möglich, existenziell und universell quantifizierte Beschränkungen zu vermischen und Mehrfachquantifizierungen umzusetzen.[3]

Für die Formulierung der Beschränkungen werden zwei grundlegende Prädikate angenommen: das einstellige Prädikat $Var/1$, das die Eigenschaft bezeichnet, eine Variable zu sein, wobei $Var(\alpha)$ definiert ist durch $\alpha \in \lhd L(D) \rhd$; das zweistellige Prädikat $Val/2$, dass die Eigenschaft angibt, eine bestimmte Variablenbelegung zu besitzen, wobei $Val(\alpha, \beta)$ definiert ist durch $\alpha \in \lhd \beta \rhd$. So kann im Kontext des Booleschen Semirings der existenziell quantifizierte Ausdruck für den WFSA in Abbildung 3.19 wie folgt formuliert werden:

$$\exists x \; Var(x) \wedge (Val(x, d'_{i_1}) \vee Val(x, d'_{i_2}) \vee ... \vee Val(x, d'_{i_p})) \tag{3.43}$$

[3]Hier ist zu beachten, dass bei den regulären prädikatenlogischen Formeln über Variablenmengen quantifiziert wird und nicht über die Wertebereiche von Variablen, wie es z. B. bei dem quantifizierten Constraint-Erfüllungsproblem (QCSP, Börner et al., 2003) der Fall ist. Innerhalb eines CSP sind alle Variablen implizit existenziell quantifiziert. Das QCSP hingegen ist eine Erweiterung des CSP, bei der die Wertebereiche der Variablen sowohl existenziell als auch universell quantifiziert werden können.

und die universell quantifizierte Beschränkung für den WFSA in Abbildung 3.20 folgendermaßen:

$$\forall x\ Var(x) \rightarrow (Val(x, d'_{i_1}) \vee Val(x, d'_{i_2}) \vee ... \vee Val(x, d'_{i_p})) \tag{3.44}$$

Jedoch entspricht das Verhalten der WFSAs aus Abbildung 3.19 und 3.20 nicht den rationalen Potenzreihen, die über die obigen regulären prädikatenlogischen Formeln spezifiziert werden. Der Support der rationalen Potenzreihen enthält auch Zeichenketten, die nicht der Musterstruktur eines Wertetupels folgen. Über die rationale Potenzreihe, die über eine reguläre prädikatenlogische Formel spezifiziert ist, kann aber eine rationale Potenzreihe definiert werden, die diese Musterstruktur berücksichtigt. Wenn die rationale Potenzreihe $\mathcal{A}$ gegeben ist, geschieht dies über den folgenden Ausdruck: $\mathcal{A} \cap (\triangleleft L(D) \triangleright)^+$. Aufseiten der FSMs kann sich die Zustands- und Übergangsanzahl unterscheiden, je nachdem ob die Musterstruktur beachtet wird oder nicht. Es kann sogar sein, dass ein entsprechender WFSA nur dann die leere Sprache erkennt, wenn er die Musterstruktur einbezieht. Im Folgenden wird bei den regulären prädikatenlogischen Formeln stets die Einschränkung auf $(\triangleleft L(D) \triangleright)^+$ angenommen.

Es soll ein Beispiel für einen komplexeren quantifizierten Ausdruck auf Grundlage des Wertebereichs $D = \{d_1, d_2, ..., d_n\}$ gegeben werden:

$$\forall x\ Var(x) \wedge Val(x, d'_1) \rightarrow \exists y\ Var(y) \wedge Val(y, d'_2) \wedge x \prec y \wedge x|y \tag{3.45}$$

Die Formel besagt, dass nach allen Variablen mit der Belegung d_1 direkt eine Variable mit der Zuweisung d_2 folgt.

Es ist auch möglich, reguläre Ausdrücke heranzuziehen, um quantifizierte Beschränkungen zu formulieren. Dann wird der Bezug zu dem Regular-Membership-Constraint aus Abschnitt 3.1.3 deutlich. So kann die existenziell quantifizierte Beschränkung in 3.43 durch folgenden regulären Ausdruck beschrieben werden, bei dem von den rationalen Operationen Vereinigung, Konkatenation und Sternhülle Gebrauch gemacht wird:

$$(\triangleleft \Delta^* \triangleright)^* \triangleleft (d'_{i_1} \cup d'_{i_2} \cup ... \cup d'_{i_p}) \triangleright (\triangleleft \Delta^* \triangleright)^* \tag{3.46}$$

Angenommen $\mathcal{X}_{def}$ ist ein WFSA, der den Wert eines quantifizierten Constraints mit dem Typ *con* darstellt, dann entspricht der WFSA $A(\mathcal{X}_{def} \cap$

$(\triangleleft\, \Delta^* \,\triangleright)^{|con|}$) einem entfalteten FSA eines Regular-Membership-Constraints. Solch eine Entfaltung tritt auf, wenn ein quantifizierter Constraint projektiert oder mit einem anderen Constraint kombiniert wird.

Bisher wurden ausschließlich quantifizierte Constraints über dem Booleschen Semiring betrachtet. Es sollen aber auch quantifizierten Constraints für beliebige c-Semiringe möglich sein, um Verletzbarkeit, Präferenzen und Unsicherheit modellieren zu können. Es könnte beispielsweise ein bestimmtes c-Semiring-Element vergeben werden, wenn ein quantifizierter regulärer Constraint nicht erfüllt ist. Dies ist ähnlich zu dem *bewerteten Constraint-Erfüllungsproblem* (Valued Constraint Satisfaction Problem, VCSP, Schiex et al., 1995), bei dem die Bewertungselemente den Constraints zugewiesen sind. Gegeben eine rationale Sprache $\mathcal{F}$, die durch eine reguläre prädikatenlogische Formel spezifiziert ist und gegeben ein c-Semiring-Element ω, das vergeben wird, wenn der quantifizierte Constraint verletzt wird. Dann ist der Constraint-Wert $\mathcal{X}_{def}$ definiert durch:

$$\mathcal{X}_{def} = (\mathcal{F} \cap (\triangleleft\, L(D) \,\triangleright)^+) \cup (\neg\mathcal{F} \cap (\triangleleft\, L(D) \,\triangleright)^+ \langle\omega\rangle) \tag{3.47}$$

Da bei einem c-Semiring das Einselement in der Ordnung der c-Semiring-Elemente das Maximum darstellt, kann der Wert $\mathcal{X}_{def}$ auch wie folgt definiert werden:

$$\mathcal{X}_{def} = (\mathcal{F} \cap (\triangleleft\, L(D) \,\triangleright)^+) \cup (\triangleleft\, L(D) \,\triangleright)^+ \langle\omega\rangle \tag{3.48}$$

Der Constraint-Wert besitzt hierbei stets die Zwillingseigenschaft. Dies ist der Fall, da hier ein c-Semiring-Element zugewiesen wird, dass sich im regulären Ausdruck nicht im Skopus einer Sternhülle und somit im WFST in keiner Schleife befindet.

Es ist aber auch möglich, c-Semiring-Elemente indirekt über die prädikatenlogischen Formeln zuzuweisen, wenn sie als Zeichenketten repräsentiert sind. Hierbei müssen nicht unbedingt alle c-Semiring-Elemente behandelt werden, da die Menge der Constraints eines Constraint-Systems und somit die Anzahl an c-Semiring-Elementen, die über prädikatenlogische Formeln zugewiesen werden können, begrenzt ist. Es sei die Trägermenge S des verwendete c-Semirings und eine endliche Teilmenge $\bar{S} = \{s_1, s_2, ..., s_n\}$ mit $\bar{S} \subseteq S$ angenommen, wobei $\bar{0} \in \bar{S}$ ist. Des Weiteren sei eine bijektive Abbildung $\psi : \bar{S} \to \Omega^*$ mit $\Omega \subset \Sigma$ gegeben, die c-Semiring-Elemente auf ihre Zeichenkettenrepräsentation abbildet und definiert ist durch $\psi(s_r) = s'_r$ mit $s'_r \in \Omega^*$ für

jedes r mit $1 \leq r \leq n$, wobei das Alphabet Ω keine Variablenmarkierungszeichen enthält. Dann kann ein c-Semiring-Element s_r durch eine einelementige rationale Sprache repräsentiert werden:

$$L(s_r) = \{\psi(s_r)\} = \{s'_r\} \tag{3.49}$$

Die Menge $\bar{S}$ kann über eine rationale Sprache repräsentiert werden, die aus der Vereinigung der einelementigen rationalen Sprachen resultiert, die die einzelnen c-Semiring-Elemente darstellen:

$$L(\bar{S}) = L(s_1) \cup L(s_2) \cup ... \cup L(s_n) \tag{3.50}$$

Um die Zeichenketten wieder auf c-Semiring-Elemente abbilden zu können, ist die rationale Potenzreihe $L(\psi^{-1})$ gegeben:[4]

$$L(\psi^{-1}) = L(s_1)\langle s_1 \rangle \cup L(s_2)\langle s_2 \rangle \cup ... \cup L(s_n)\langle s_n \rangle \tag{3.51}$$

Um die Vergabe der c-Semiring-Elemente über die regulären prädikatenlogischen Formel umzusetzen, wird angenommen, dass hier das Prädikat $Var(\alpha)$ durch $\alpha \in \triangleleft L(D) \square L(\bar{S}) \triangleright$ und das Prädikat $Val(\alpha, \beta)$ durch $\alpha \in \triangleleft \beta \square L(\bar{S}) \triangleright$ definiert ist. Zudem wird das zweistellige Prädikat $Score/2$ angenommen, das die Eigenschaft einer Variable bezeichnet mit einem konkreten c-Semiring-Element als Strafgewicht verknüpft zu sein, wobei $Score(\alpha, \beta)$ durch $\alpha \in \triangleleft L(D) \square \beta \triangleright$ gegeben ist. Hierbei wird das Markierungszeichen $\square \in \Sigma - \{\Delta, \Omega\}$ verwendet, um den Variablenwert von der c-Semiring-Element-Zeichenkette zu trennen.

Wenn $\mathcal{F}$ eine reguläre prädikatenlogische Formel ist, die mithilfe dieser Prädikate formuliert ist, dann kann über $\mathcal{F}$ ein Constraint-Wert spezifiziert werden:

$$Range(\mathcal{F} \circ (\triangleleft L(D)(\square L(\psi^{-1}) \times \epsilon) \triangleright)^+) \tag{3.52}$$

[4]In manchen Fällen kann eine unendliche Teilmenge der Trägermenge des verwendeten c-Semirings über eine unendliche rationale Sprache dargestellt werden (vgl. Abschnitt 3.4.1). Hierüber ist es möglich, eine rationale Potenzreihe mit unendlichem Support zu spezifizieren, die die Wörter aus der entsprechenden rationalen Sprache auf c-Semiring-Elemente abbildet. Wenn die natürlichen Zahlen betrachtet werden, ist die Spezifizierung solch einer rationalen Potenzreihe mit unendlichem Support beispielsweise dann möglich, wenn die natürlichen Zahlen über eine unäre Codierung repräsentiert sind und wenn die abstrakte Multiplikation des c-Semirings der Addition der natürlichen Zahlen entspricht. Hier wird sich jedoch auf eine endliche Teilmenge von S beschränkt.

Über den Teilausdruck '$(\triangleleft L(D)(\square L(\psi^{-1}) \times \epsilon) \triangleright)^+$' werden konkrete c-Semiring-Elemente eingeführt und es werden die c-Semiring-Element-Zeichenketten über Epsilon entfernt. Diese sind demnach nur während der Kompilation des Wertes eines quantifizierten Constraints vorhanden. Wenn die c-Semiring-Element-Zeichenketten innerhalb einer regulären prädikatenlogischen Formel keinen Einschränkungen unterliegen, wird allen Wertetupeln das Einselement zugeordnet, da dieses in der Ordnung der c-Semiring-Elemente stets das Maximum darstellt. Wenn hingegen die c-Semiring-Element-Zeichenketten innerhalb einer regulären prädikatenlogischen Formel eingeschränkt werden, können verschiedenen Wertetupeln unterschiedliche c-Semiring-Elemente zugewiesen werden. Wenn die c-Semiring-Elemente indirekt über die regulären prädikatenlogischen Formeln vergeben werden, ist jedoch die Zwillingseigenschaft der resultierenden WFSAs nicht gewährleistet.

Es soll ein Beispiel gegeben werden. Hierzu wird an das Beispiel zur syntaktischen Analyse aus Abschnitt 3.2.2 angeknüpft. Dort ist ein Wertebereich mit $D = \{\texttt{SUBJ}, \texttt{OBJA}, \texttt{OBJD}\}$ gegeben, der durch einelementige Zeichenketten repräsentiert ist. Des Weiteren wird über den c-Semiring $S = (\mathbb{R}_0^+ \cup \{+\infty\}, min, +, +\infty, 0)$ ein gewichtetes CSP implementiert. Hier wird angenommen, dass eine endliche Teilmenge der Trägermenge dieses c-Semirings durch $\bar{S} = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$ definiert ist und dass $\bar{S}$ durch einelementige Zeichenketten repräsentiert ist, deren Zeichen den Elementen in $\bar{S}$ entsprechen, sodass $\Omega = \bar{S}$ ist. Nun kann beispielsweise formuliert werden, dass jede Variable mit der Belegung `OBJA` mit einem Gewicht von 1 bestraft wird:

$$\forall x \; Var(x) \wedge Val(x, \texttt{OBJA}) \rightarrow Score(x, 1) \tag{3.53}$$

Hier richtet sich das c-Semiring-Element, dass einem Wertetupel zugeordnet wird, nach der Anzahl der `OBJA`-Werte innerhalb des Wertetupels. So würde dem Wertetupel (`SUBJ`,`OBJA`,`S`) das Strafgewicht 1 und dem Wertetupel (`S`,`OBJA`,`SUBJ`,`OBJA`,`OBJA`,`OBJA`) das Strafgewicht 4 zugewiesen werden. Solch eine Beschränkung ist beispielsweise nicht umsetzbar, wenn ein c-Semiring-Element nur dann vergeben wird, wenn ein Constraint verletzt ist.

Wenn jedoch innerhalb einer regulären prädikatenlogischen Formel verschiedene Strafgewichte zugewiesen werden, kann es zu Konflikten kommen:

$$\begin{aligned} \forall xy\ Var(x) \wedge Val(x, \texttt{SUBJ}) \wedge Var(y) \wedge Val(y, \texttt{OBJA}) \rightarrow \\ x \prec y\ \wedge Score(x, 4)\ \vee \\ y \prec x\ \wedge Score(x, 6) \end{aligned} \tag{3.54}$$

Die Formel besagt, dass einer Variablen mit der Belegung SUBJ das Strafgewicht 4 zugeordnet wird, wenn sie sich in der Variablenordnung vor einer Variable mit dem Variablenwert OBJA befindet. Wenn sich eine Variable mit dem Wert SUBJ hingegen nach einer Variablen mit der Wertzuweisung OBJA steht, wird das Strafgewicht 6 zugewiesen. So wird beispielsweise dem Wertetupel (SUBJ,S,OBJA) das Gewicht 4 und dem Tupel (OBJA, S, SUBJ) das Gewicht 6 zugeordnet. Dem Tupel (OBJA,SUBJ,OBJA) würde dahingegen das Nullelement zugewiesen werden. Dies liegt daran, dass einer Variable nicht gleichzeitig zwei verschiedene Strafgewichte zugeordnet werden können. Als Resultat dieses Konfliktes ist der Wertetupel (OBJA, SUBJ, OBJA) nicht mehr im Support der rationalen Potenzreihe vorhanden. Um dies zu verhindern, können bei den Gewichtszuweisungen alle höheren Gewichte in Option zusätzlich vergeben werden. So wird bei einem Konflikt immer das schlechteste Strafgewichte zugewiesen. Nach dieser Strategie kann die obige Beispielformel wie folgt ergänzt werden:

$$\begin{aligned} \forall xy\ Var(x) \wedge Val(x, \texttt{SUBJ}) \wedge Var(y) \wedge Val(y, \texttt{OBJA}) \rightarrow \\ x \prec y\ \wedge Score(x, \{4, 5, 6, 7, 8, 9\})\ \vee \\ y \prec x\ \wedge Score(x, \{6, 7, 8, 9\}) \end{aligned} \tag{3.55}$$

Innerhalb der Formel werden zu dem Strafgewicht 4 in Option zusätzlich die Gewichte >4 vergeben und zu dem Strafgewicht 6 die Gewichte >6. So kann dem Wertetupel (OBJA, SUBJ, OBJA) letztlich das Strafgewicht 6 zugewiesen werden.

Erweiterte Eigenschaften

Zusätzlich zu den nativen Eigenschaften einer Variable, wie sie weiter oben aufgeführt sind, können mit Variablen auch *erweiterte Eigenschaften* verknüpft werden, die sich auf zusätzliche Informationen beziehen (vgl. Abschnitt 3.4.2). Im Kontext der vorhergehenden Beispiele zur syntaktischen Analyse wäre eine erweiterte Eigenschaft einer Variable beispielsweise die, eine bestimmte Wortart oder Lemmaform zu besitzen.

Angenommen, es existiert ein Constraint-System $CS = (S, D, V^{\tau}, I)$, bei dem mit den Variablen bestimmte Informationen verknüpft sind. Dann kann der Wert eines quantifizierten regulären Constraints ($\mathcal{X}_{def}$,$L(con^{\tau})$) als eine rationale Potenzreihe repräsentiert werden, die Wertetupel aus $(\triangleleft\, L(I) \diamond L(D)\, \triangleright)^{+}$ auf Elemente in S abbildet und alle anderen Zeichenketten auf das Nullelement von S. So ist der Support des quantifizierten regulären Constraints eine Teilmenge der folgenden rationalen Sprache:

$$Support(\mathcal{X}_{def}) \subseteq (\triangleleft\, L(I) \diamond L(D)\, \triangleright)^{+} \tag{3.56}$$

Ein Wertetupel ist hier nur dann konsistent, wenn seine Informationen mit den verknüpften Informationen der Variablen in *con* übereinstimmen. Der Wert eines regulären quantifizierten Constraint kann hierbei als ein simultaner Constraint-Wert angesehen werden, der für alle Constraint-Systeme kompatibel ist, die über den Wertebereich und über die Menge an Informationen gebildet werden können (vgl. Abschnitt 3.4.2). Wenn ein simultanes Constraint-System $(S, D, V_1^{\tau_1}|V_2^{\tau_2}|\ldots|V_n^{\tau_n}, I)$ betrachtet wird, dann ist der Wert eines quantifizierten Constraints über den Semiring S, den Wertebereich D und über die Menge an Informationen I definiert und für die Variablenmengen $V_1^{\tau_1}, V_2^{\tau_2}, \ldots, V_n^{\tau_n}$ stets wohldefiniert. Daher können die Werte quantifizierter regulärer Constraints auch innerhalb simultan repräsentierter SCSPs Anwendung finden.

Es soll ein konkretes Beispiel für die Formulierung von quantifizierten Beschränkungen mit erweiterten Eigenschaften im Kontext des Booleschen Semirings gegeben werden. Hierzu wird an die Beispiele zur syntaktischen Analyse aus den vorherigen Abschnitten angeknüpft. Dort existiert der Wertebereich $D = \{\texttt{S}, \texttt{SUBJ}, \texttt{OBJA}\}$, der durch einelementige Zeichenketten repräsentiert ist. Des Weiteren wird hier die Menge an Informationen $I = \{\texttt{NN}, \texttt{NE}, \texttt{VVFIN}\}$ angenommen, die verschiedene Wortarten enthält, die mit einem Wort eines Satzes und somit mit einer Variable verknüpft sein können, wobei `NN` ein normales Nomen, `NE` einen Eigennamen und `VVFIN` ein finites Vollverb bezeichnet. Die Informationen sind durch einelementige Zeichenketten repräsentiert, deren Zeichen den Elementen in I entsprechen. Um sie zu berücksichtigen, sind die Definitionen der Prädikate $Var/1$ und $Val/2$ aus dem vorherigen Abschnitt wie folgt angepasst: $Var(\alpha)$ ist definiert durch $\alpha \in \triangleleft\, L(I_1) \diamond L(D)\, \triangleright$ und $Val(\alpha, \beta)$ durch $\alpha \in \triangleleft\, L(I_1) \diamond \beta\, \triangleright$. Zudem wird das

zweistellige Prädikat $Cat/2$ angenommen, das die Eigenschaft einer Variable bezeichnet für ein Wort zu stehen, dass eine bestimmte Wortart besitzt. $Cat(\alpha, \beta)$ ist definiert durch $\alpha \in \lhd \beta \diamond L(D) \rhd$. Ausgehend hiervon kann beispielsweise formuliert werden, dass für eine bestimmte Wortart nur ausgewählte Satzfunktionen erlaubt sind:

$$\forall x \; Var(x) \wedge Cat(x, \{\texttt{NN}, \texttt{NE}\}) \rightarrow Val(x, \{\texttt{SUBJ}, \texttt{OBJA}\}) \tag{3.57}$$

Diese Beschränkung besagt, dass ein Nomen oder ein Eigenname entweder ein Subjekt (`SUBJ`) oder ein Akkusativobjekt (`OBJA`) sein muss. Es kann aber auch formuliert werden, dass eine bestimmte Satzfunktion nur ausgewählten Wortarten zuzuordnen ist:

$$\forall x \; Var(x) \wedge Val(x, \{\texttt{S}\}) \rightarrow Val(x, \texttt{VVFIN}) \tag{3.58}$$

Die Beschränkung besagt, dass der Wurzelknoten eines Satzes (`S`) ein Vollverb sein muss. Nun könnte aber auch formuliert werden, dass ein Nomen oder Eigenname stets vor dem finiten Vollverb steht:

$$\begin{gathered}\forall xy \; Var(x) \wedge Cat(x, \{\texttt{NN}, \texttt{NE}\}) \wedge Var(y) \wedge Cat(y, \texttt{VVFIN}) \rightarrow \\ x \prec y\end{gathered} \tag{3.59}$$

Solch eine quantifizierte Beschränkung schränkt nicht direkt die Variablenwerte ein und kann zu einem inkonsistenten Constraint-Problem führen. Im Kontext der simultanen Repräsentation von Constraint-Problemen kann durch solch eine Beschränkung aber zwischen verschiedenen involvierten Constraint-Systemen disambiguiert werden.

Es ist auch möglich, über eine verknüpfte Information mehrere Eigenschaften zugänglich zu machen. So könnte beispielsweise die Wortart zusammen mit der Lemmaform betrachtet werden. Angenommen, die verknüpfte Information wird durch die rationale Sprache I repräsentiert, die der Konkatenation der rationalen Sprachen $\mathcal{I}_{cat}$ und $\mathcal{I}_{lem}$ entspricht. $\mathcal{I}_{cat}$ enthält alle möglichen Wortarten und $\mathcal{I}_{lem}$ alle Lemmaformen, die berücksichtigt werden sollen. Hierbei wird angenommen, dass beide rationalen Sprachen eindeutig voneinander abgrenzbar sind. Dann ist hier das zweistellige Prädikat $Cat(\alpha, \beta)$ definiert durch $\alpha \in \lhd \beta \; \mathcal{I}_{lem} \diamond L(D) \rhd$. Zudem kann zusätzlich das zweistellige Prädikat $Lem/2$ herangezogen werden, das die Eigenschaft einer

Variable bezeichnet, für ein Wort zu stehen, dass eine bestimmte Lemmaform besitzt. $Lem(\alpha, \beta)$ ist hierbei gegeben durch $\alpha \in \triangleleft \mathcal{I}_{cat}\ \beta \diamond L(D) \triangleright$. Mit Zugriff auf diese beiden erweiterten Eigenschaften können nun reichhaltigere Beschränkungen formuliert werden.

Kapitel 4

Robustes Dependenzparsing als SCSP mithilfe von FSMs

In diesem Kapitel wird beschrieben, wie regelbasiertes syntaktisches Dependenzparsing geschriebener Sprache innerhalb des SCSP-Frameworks mithilfe von FSMs umgesetzt werden kann. Der Ansatz beruht auf der Veröffentlichung Didakowski (2010, Abschnitt 5.3), die im Rahmen dieser Arbeit entstand. Dort wird einem Problemzerlegungsansatz nachgegangen, bei dem das Parsing-Problem bottom-up in Teilprobleme zerlegt wird und das Lösen der Teilprobleme und die Berechnung der Zerlegung simultan geschehen. Für die Umsetzung wird auf ein simultanes SCSP und auf quantifizierte reguläre Constraints zurückgegriffen. Dieser Ansatz erlaubt verletzbare Regeln und ist auf keine konkrete SCSP-Instanz festgelegt. Es ist der erste FSM-Ansatz, bei dem das Parsing darin besteht, die Eingabe sukzessive auf Epsilon abzubilden. Hier ist der Ansatz detaillierter ausgearbeitet und um mehrere Aspekte erweitert. Dies betrifft die Behandlung von nichtprojektiven Strukturen und die kompakte Repräsentation von Zwischenergebnissen.

In Abschnitt 4.1 wird auf verwandte Arbeiten eingegangen, die sich im Kontext des regelbasierten Dependenzparsings bewegen. Einerseits sind dies Arbeiten im Bereich der endlichen Techniken und andererseits Arbeiten im Bereich der Constraint-Verarbeitung. Daraufhin wird in Abschnitt 4.2 gezeigt, wie der WCDG-Ansatz mithilfe von FSMs im Rahmen des SCSP-Frameworks umgesetzt werden kann. Es wird auf Problematiken eingegangen und es wird gezeigt, dass eine naive Herangehensweise aus praktischen Gesichtspunkten unrealistisch ist. Daraufhin wird in Abschnitt 4.3 ein Pro-

blemzerlegungsansatz vorgestellt, der im Fokus dieses Kapitels steht und eine praktische Implementierung ermöglicht.

4.1 Verwandte Arbeiten

Im Folgenden werden verwandte Arbeiten aus dem Bereich des dependenzorientierten Parsings vorgestellt. Es werden einerseits Ansätze zum Parsing mithilfe von FSMs und andererseits Ansätze zum Parsing als Constraint-Erfüllungs- oder Constraint-Optimierungsproblem betrachtet.

4.1.1 Parsing mithilfe von FSMs

Es existiert ein breites Spektrum an Parsingansätzen, bei denen FSMs involviert sind. Hier werden solche Ansätze betrachtet, die dependenzorientiert sind und bei denen ausschließlich FSMs und rationale Operationen für das Parsing herangezogen werden.

Flaches konstruktives Parsing

In Grefenstette (1999) wird im Kontext der flachen, robusten Satzanalyse vorgeschlagen, mithilfe von FSTs über Klammerungen zusammenhängende nichtrekursive Gruppierungen auszuzeichnen und innerhalb dieser über spezielle Label Köpfe zu markieren. Die Klammerung solcher Strukturen wird auch als Chunking bezeichnet. Über die Label werden implizit Unterordnungsbeziehungen zwischen dem Kopf und anderen Wörtern innerhalb einer Gruppierung angezeigt. Für den Satz „Elma träumt heimlich von Saas-Fee“ ist eine entsprechende Struktur gegeben:

[$_{\texttt{NG}}$ `*HeadN` Elma $_{\texttt{NG}}$] [$_{\texttt{VG}}$ `*ActV` träumt heimlich $_{\texttt{VG}}$] [$_{\texttt{NG}}$ von `*PrepN` Saas-Fee $_{\texttt{NG}}$]

Die Klammerung [$_{\texttt{NG}}$ ... $_{\texttt{NG}}$] markiert Nominalgruppen und die Klammerung [$_{\texttt{VG}}$ ... $_{\texttt{VG}}$] Verbalgruppen. Die Markierungs-Label für die Köpfe stehen jeweils direkt vor dem Bezugswort und beginnen mit dem Zeichen ‘`*`’. Das Label `*HeadNN` markiert den nominalen syntaktischen Kopf einer Nominalgruppe, das Label `*PrepN` hingegen den nominalen semantischen Kopf einer Nominalgruppe, der durch eine Präposition modifiziert ist und das Label `*ActV` das aktivische Vollverb innerhalb einer verbalen Gruppierung.

Es wird der *gerichtete Ersetzungsoperator* (directed replacement operator, Karttunen, 1996) vorgeschlagen, um einen FST zu kompilieren, der mithilfe einer Longest-Match-Strategie bestimmte Muster klammert. Zudem wird der *konditionale Ersetzungsoperator* (conditional replacement operator, Karttunen, 1995) herangezogen, um einen FST zu erstellen, der abhängig vom Kontext Label einfügen oder umschreiben und somit Köpfe markieren kann. Das Umschreiben eines Labels ermöglicht es, initiale Default-Markierungen zu revidieren oder genereller gehaltene Labels in feinkörnigere zu überführen.

In Aït-Mokhtar und Chanod (1997) wird vorgeschlagen, eingebettete Klammerungen mit gebundener Tiefe zu erlauben, sodass komplexere Strukturen möglich werden. Solche Einbettungen werden mithilfe des gerichteten Ersetzungsoperators implementiert. Um komplexere Gruppierungen zu behandeln, werden erst Anfänge und Abschlüsse einer potenziellen Gruppierung markiert. Dann werden über den gerichteten Ersetzungsoperator anhand der Markierungen die Klammerungen eingefügt. Zudem werden sowohl Wörter als auch Wortgruppierungen mit syntaktischen Funktionen gelabelt, indem ihnen ein Funktions-Label nachgestellt wird. Ein Beispiel für solch eine Analysestruktur ist für den obigen Beispielsatz wie folgt:

[$_{VG}$ [$_{NG}$ Elma $_{NG}$]SUBJ träumt heimlich VM [$_{PG}$ von PrepN> Saas-Fee $_{PG}$]PPObj $_{VG}$]

Das syntaktische Label SUBJ bezeichnet das grammatische Subjekt, das Label PPObj ein Präpositionalobjekt, das Label MV ein Verbmodifizierer und das Label PrepN> eine Präposition mit einem Komplement zur Rechten.[1]

Auch in Megyesi und Rydin (1999) sind eingebettete Strukturen mit gebundener Tiefe erlaubt. Es werden spezielle Klammerungen verwendet, um mit einer Gruppierung eine bestimmte syntaktische Funktion zu verknüpfen. Des Weiteren werden wie in Grefenstette (1999) Köpfe innerhalb von Gruppierungen markiert, jedoch nur innerhalb von Nominal- und Verbgruppen. Ein Beispiel für solch eine Analysestruktur ist für den obigen Beispielsatz wie folgt:

[1] In Abney (1996) wird ein Ansatz vorgestellt, bei dem eine syntaktische Analyse darin besteht, sukzessive über Klammerungen eine Struktur aufzubauen. Diese werden dort als Chunks bezeichnet. Es wird eine endliche Kaskade (finite state cascade) angewendet, um einen Satz zu analysieren. Die Ausgabe einer Ebene ist hierbei die Eingabe für die nächste. Über die Ebenen hinweg werden sukzessive Bereiche der Eingabe auf einzelne Elemente reduziert. Die zu reduzierenden Bereiche sind hierbei über FSMs beschrieben. Das bedeutet, dass die FSMs lediglich in das System eingebettet sind. Das Markieren von Köpfen oder syntaktischen Funktionen wird über sogenannte Aktionen realisiert.

{***Subj [NG *HeadN Elma NG] Subj***} [VG *ActV träumt heimlich VG] {***Advl [PG von [NG *PrepN Saas-Fee NG] PG] Advl***}

Die Klammerung {***Subj ... Subj***} zeigt das Subjekt eines Satzes an und die Klammerung {***Advl ... Advl***} eine adverbiale Ergänzung.

Flaches eliminatives Parsing

In Koskenniemi (1990) wird ein eliminativer Parsing-Ansatz vorgestellt, der mithilfe von FSAs umgesetzt ist. Der Ansatz basiert auf dem methodologischen Paradigma der *Constraint Grammar* (CG, Karlsson, 1990) und realisiert eine dependenzorientierte Analyse von Sätzen, bei der syntaktische Tags verwendet werden, um flache, unterspezifizierte Dependenzstrukturen anzuzeigen. Diese Auszeichnung wird auch als syntaktisches Tagging bezeichnet. Die Unterspezifizierung besteht hierbei in der genauen Identität eines übergeordneten Wortes:

	Maria	aus	der	Boutique	im	Tessin	verweilte	
@@	@SUBJ	@<NOM	@DN>	@<P	@<NOM	@<P	@+FMAINV	@@

In diesem Beispiel bezeichnet das Tag @DN> einen Determiner, der ein rechtes Nomen modifiziert, das Tag @<NOM ein postnominales Attribut, das ein linkes Nomen modifiziert, das Tag @<P das Komplement einer links stehenden Präposition, das Tag @SUBJ das grammatische Subjekt und das Tag @+FMAINV das finite Hauptverb und das Tag @@ markiert den Anfang und das Ende des Eingabesatzes. Obwohl jedem Wort genau ein Tag zugewiesen ist, besteht eine Anbindungsambiguität bezüglich der Präposition 'im', die entweder den Eigennamen 'Maria' oder das Nomen 'Boutique' modifiziert. Solch eine Unterspezifizierung stellt eine kompakte Repräsentation unaufgelöster Ambiguitäten bereit. Nebensätze werden durch spezielle Tags markiert, wobei Rechts- und Linkseinbettung durch Iteration ersetzt und die Tiefe von Zentraleinbettung stark begrenzt ist. Allgemein gilt, dass jedes Wort nur genau einem Haupt- oder Nebensatz zuzuordnen ist. In Voutilainen und Tapanainen (1993) werden zusätzliche syntaktische Tags verwendet, um die Funktionen von Haupt- und Nebensätzen anzuzeigen. Diese werden Hauptverben ergänzend hinzugefügt und machen diese zu Repräsentanten für Haupt- und Nebensatz. Jedoch ist es nicht immer möglich, Paare, bestehend aus übergeordnetem und untergeordnetem Wort, zuverlässig anhand der syntaktischen

Tags zu extrahieren (im Kontext der CG siehe Tapanainen und Järvinen, 1997).

Die Analyse eines Satzes besteht darin, morphologische und strukturelle Ambiguitäten zu eliminieren und Satzgrenzen festzusetzen. Diese drei Schritte geschehen parallel. Hierzu wird initial ein FSA konstruiert, der der aus der Verkettung der Lesarten der einzelnen Wörter des Eingabesatzes entsteht. Dieser wird auch als Satzautomat bezeichnet und bildet die Eingabe der Analyse. Ein Beispiel hierfür ist in Abbildung 4.1 für den Satz „Fliegen fliegen“ gegeben. Aus Platzgründen sind hier keine Lemmata oder Oberflä-

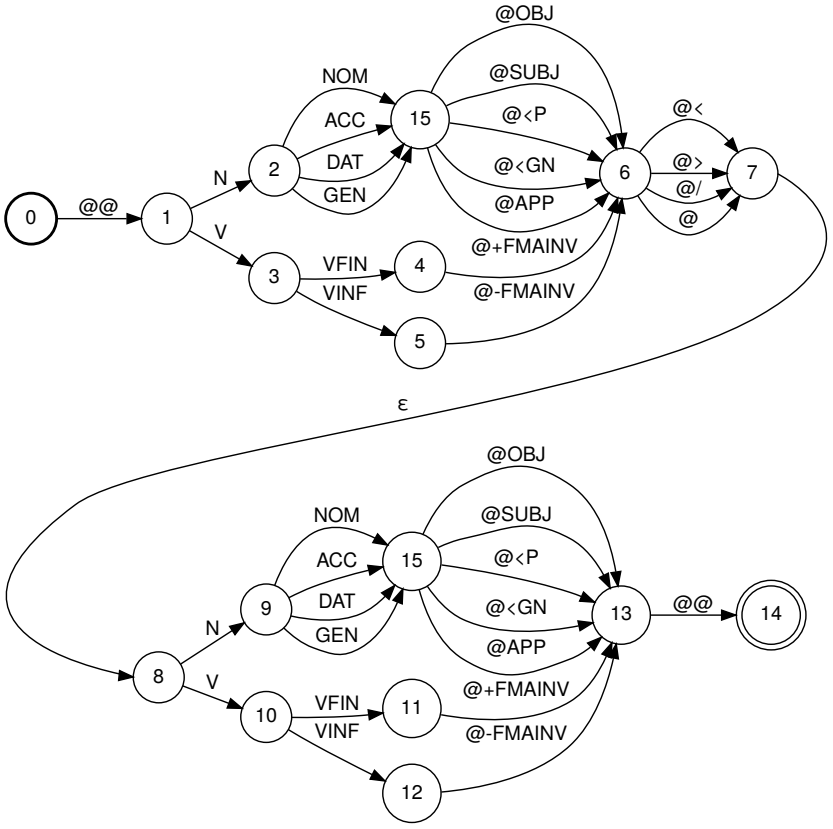


Abbildung 4.1: Eingabeautomat.

chenformen im Automaten repräsentiert und die morphologischen Merkmale sind nur angedeutet. Zudem ist bezüglich der Zuteilung der Wortarten die Groß- und Kleinschreibung ignoriert. Für das Wort ‘Fliegen’ bzw. ‘fliegen’ wird initial eine Lesart als Nomen (`NN`) und eine Lesart als Verb (`V`) angenommen. Für das Nomen sind alle Kasus möglich (`NOM`, `ACC`, `DAT`, `GEN`)

und das Verb kann finit (`VFIN`) und infinit (`VINF`) sein. Die Lesarten beider Wörter sind durch spezielle Tags voneinander getrennt: Das Tag `@` markiert eine Wortgrenze, das Tag `@/` die Grenze zwischen zwei adjazenten Sätzen (bei diesem Ansatz ist Rechts- und Linkseinbettung durch Iteration ersetzt) und die Tags `@<` und `@>` dienen der Markierung zentral eingebetteter Sätze. Über solch einen Satzautomaten können alle initialen Lesarten kompakt repräsentiert werden.

Um Lesarten zu eliminieren, können Regeln über reguläre Ausdrücke formuliert und in FSAs kompiliert werden, die grammatische Lesarten akzeptieren und ungrammatische zurückweisen. Bei der Formulierung der Regeln wird auf den *Kontextbeschränkungsoperator* (context restriction operator, Koskenniemi, 1983, Abschnitt 2.3.8) zurückgegriffen. Über die Regelautomaten kann das Auftreten von Teilzeichenketten anhand des Kontextes einschränkt werden, um linguistische Generalisierungen auszudrücken. Hierbei kann gleichzeitig auf Satzgrenzenmarkierer, morphologische Symbole und syntaktische Tags zugegriffen werden.

Um ungrammatische Lesarten zu eliminieren, werden die Regelautomaten mithilfe der Schnittoperation auf einen Satzautomaten angewendet. Dieser erkennt dann nur noch eine Untermenge der initialen Lesarten und im besten Fall genau eine grammatische. Gegeben sei ein Satzautomat $\mathcal{S}$ und eine Grammatik bestehend aus den Regelautomaten $\mathcal{R}_1, \mathcal{R}_2, ..., \mathcal{R}_n$, dann ist die Analyse eines Satzes definiert als: $\mathcal{S} \cap \mathcal{R}_1 \cap \mathcal{R}_2 \cap ... \cap \mathcal{R}_n$. Ein grundlegendes Problem bei der Analyse ist jedoch, dass sich die Zwischenergebnisse bei der Berechnung sehr stark aufblähen können, obwohl das Resultat letztendlich nur wenige Zustände und Übergänge besitzt. Um dies zu vermeiden und um die Analyse zu beschleunigen können bestimmte Strategien angewendet werden (siehe Koskenniemi et al., 1992, Abschnitt 4, Voutilainen und Tapanainen, 1993, Abschnitt 5 und Tapanainen, 1997, Abschnitt 10.3 und 10.4): Regelautomaten können bei der Analyse ausgelassen werden, wenn sie eindeutig keine Lesart entfernen würden; es können Regelautomaten zuvor in einer Offline-Berechnung zusammengefasst werden; im Fall des sequenziellen Schnitts kann die Anwendungsabfolge der Regelautomaten optimiert und es können FSAs zwischen den Berechnungsschritten minimiert werden; im Fall des parallelen Schnitts kann die Analyse als ein Suchproblem mithilfe von Backtracking realisiert werden. Es existieren auch Ansätze, die den Fokus auf spezielle Datenstrukturen haben, die es ermöglichen, Zwischenergebnis-

se während der Regelanwendungen kompakter darzustellen (siehe Yli-Jyrä, 1995 und Piitulainen, 1995).

Für den eliminativen Ansatz wurde unter anderem die Bezeichnung *Parallel Constraint Grammar* vorgeschlagen, um den Unterschied zu einer sequenziell arbeitenden Umsetzung der CG deutlich zu machen (siehe hierzu Koskenniemi, 1997). Über die Jahre hinweg hat sich letztlich der Name *Finite-State Intersection Grammar* (FSIG) etabliert. In Tapanainen (1997, Abschnitt 10.7) wird gezeigt, dass die Zeit- und Platzkomplexität der Anwendung einer fixen FSIG linear bezüglich der Größe des Satzautomaten ist, der wiederum linear mit der Länge des Eingabesatzes wächst. Jedoch muss hier berücksichtigt werden, dass der implizite FSA, der sich aus der Zusammenfassung aller Regelautomaten ergibt, in der Komplexitätsabschätzung zu einer enorm großen und für die praktische Anwendung heiklen Konstante führen kann (siehe auch Yli-Jyrä, 2005a, Abschnitt 17.2.3).

Parsing als erweiterter endlicher Ansatz

In Oflazer (2003) wird erstmalig ein Ansatz zum dependenzorientierten Bottom-Up-Parsing mit FSMs vorgestellt, bei dem der reguläre Rahmen verlassen wird und FSTs in einem Fixed-Point-Algorithmus eingebunden sind, wobei ausschließlich Automatenoperationen angewendet werden, um einen Satz zu analysieren.[2] Genauer handelt es sich um projektives Dependenzparsing, bei dem die Berechnung der Analysen aus einer ungebundenen Anzahl an Zeichenketten-Transformationen besteht. Dort bezieht sich das Parsing auf das Türkische und auf Unterordnungsbeziehungen zwischen inflektionalen Gruppen. Um den Sachverhalt zu vereinfachen, werden im Folgenden Unterordnungsbeziehungen zwischen Wörtern betrachtet.

Unterordnungen werden auf Zeichenkettenebene innerhalb verschiedener *Kanäle* (Channels) codiert, innerhalb derer keine überkreuzenden Unterordnungen erlaubt sind. Eine schematische Darstellung der Codierung eines Wortes mit Berücksichtigung der Kanäle ist in Abbildung 4.2 gegeben. Die Zeichen ‘<’ und ‘>’ werden verwendet, um die Codierung eines Wortes nach links und rechts hin abzugrenzen. Die Zeichen ‘(’ und ‘)’ dienen hingegen zur Ab-

[2]Es existiert auch ein Ansatz zum Top-Down-Parsing kontextfreier Grammatiken und Tree-Adjoining-Grammatiken, der FSTs in einem Fixed-Point-Algorithmus einbindet und der ausschließlich Automatenoperationen anwendet, um einen Satz zu analysieren (Roche, 1997).

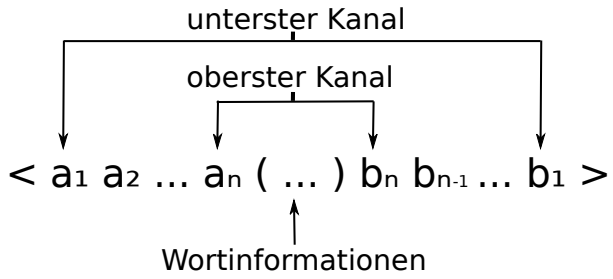


Abbildung 4.2: Wort- und Kanalcodierung.

grenzung der verschieden sprachlichen Informationen eines Wortes wie z. B. der Wortart oder der Lemmaform. Ein Kanal ist über die Position in der Teilzeichenkette $a_1a_2...a_n$ bzw. $b_nb_{n-1}...b_1$ definiert. So wäre beispielsweise a_5 und b_5 dem fünften Kanal zuzuordnen. Innerhalb der Kanäle werden spezielle Zeichen verwendet, um Unterordnungen anzuzeigen: Kleinbuchstaben markieren untergeordnete Wörter und Großbuchstaben übergeordnete. Das Zeichen '0' zeigt an, dass in einem Kanal keine Unterordnung vorliegt, das Zeichen '1' hingegen, dass sich ein Wort zwischen zwei Wörtern befindet, die in dem Kanal in einer Unterordnungsbeziehung stehen.

Die Eingabe des Parsingalgorithmus ist ein Satzautomat, der der Konkatenation der einzelnen Wortcodierungen entspricht, wobei initial kein Kanal vorhanden ist. Während des Parsing wird dann bei jedem Bottom-Up-Schritt sukzessive ein neuer Kanal an oberster Stelle hinzugefügt (also direkt an den Grenzzeichen '(' und ')'). Dort werden dann die Unterordnungen über entsprechende Buchstaben angezeigt. Es wird ausgeschlossen, dass Wörter in einem Kanal in einer Unterordnung eingebunden und gleichzeitig in einem untereren Kanal mit einer '1' markiert sind. Hierdurch werden Überkreuzungen und somit nichtprojektive Strukturen unterbunden. Der Algorithmus terminiert, wenn keine Unterordnungen mehr möglich sind. Während des Parsings werden Analysen, die im obersten Kanal keine Unterordnungen anzeigen, aus dem Satzautomaten entfernt. Dies gilt auch für Analysen, die genau ein unabhängiges Wort enthalten. Letztere werden aber in einem Ergebnisautomaten gespeichert.

Die rechts- und linksseitigen Unterordnungen können über Regeln eingeschränkt werden, die als FST repräsentiert sind und auf Wortinformationen des über- und untergeordneten Wortes (z. B. Wortart oder Lemmaform), auf Wortinformationen von Wörtern zwischen über- und untergeordnetem Wort

und auf Informationen über die Wortabfolge Bezug nehmen können. Sie sind mithilfe einer Variante des *optionalen Ersetzungsoperators* (optional replacement operator, Beesley und Karttunen, 2003) implementiert und gelten parallel, das heißt, dass die Regelordnung nicht relevant ist.

Einige sprachliche Einschränkungen lassen sich über die Regeln jedoch nicht ausdrücken. So kann beispielsweise nicht gefordert werden, dass zu einem Nomen höchstens ein Determiner existiert. Um solch eine Eindeutigkeitsbedingung umzusetzen, müsste auf die Kanalinformation eines Wortes Bezug genommen werden. Daher werden Constraints vorgeschlagen, die über FSAs implementiert sind, Zugriff auf die Kanalinformationen besitzen und nach jedem Bottom-Up-Schritt angewendet werden. Sie werden über reguläre Ausdrücke erstellt, die keinen weiteren Regelformatbeschränkungen unterliegen. Diese Kombination aus Constraints und Bottom-Up-Parsing kann hierbei als eine Mischung aus einem konstruktiven und einem eliminativen Ansatz angesehen werden. Über den konstruktiven Teil wird übergeneriert und über den eliminativen Teil werden übergenerierte Analysen wieder ausgeschlossen.

Um robustes Parsing zu ermöglichen, werden schrittweise mehrere unabhängige Wörter erlaubt, falls keine Ergebnis mit nur einem unabhängigen Wort gefunden werden kann. Hierzu wird die *nachsichtige Komposition* (lenient composition, Karttunen, 1998) herangezogen. Der Ansatz weist jedoch Probleme mit zyklischen Dependenzstrukturen auf. Zwar können einige Zyklen mithilfe von Constraints ausgeschlossen werden, dies ist aber nicht immer möglich.

Parsing durch Kontraktionen

In Yli-Jyrä (2012) wird ein Ansatz vorgestellt, der Dependenzparsing über gewichtete Kontraktionen von Kanten innerhalb eines Dependenzbaums mithilfe von WFSTs realisiert. Um die Dependenzbäume auf Zeichenkettenebene darzustellen, wird vorgeschlagen, diese über ein bestimmtes Klammerungsschema anzuzeigen. Ein projektiver Dependenzbaum für den den Satz „Eva sitzt barfuß am Zürichsee“ kann durch folgende ausgeglichene Dependenzklammerung repräsentiert werden:

Eva		sitzt		barfuß		am		Zürichsee
<SUBJ	#	SUBJ\/ADVL/PP	#	ADVL>	#	<DET	#	DET\PP>

Es existieren zwei grundlegende Klammerungstypen, die Unterordnungen anzeigen. Der eine Klammerungstyp markiert das untergeordnete Wort mit einer öffnenden Klammer der Form '<...' und das übergeordnete Wort mit einer schließenden Klammer der Form '...\'. Der andere Klammerungstyp markiert das übergeordnete Wort mit einer öffnenden Klammer der Form '/...' und ein untergeordnetes Wort mit einer schließenden Klammer der Form '...>'. Die Label innerhalb der Klammersymbole zeigen verschiedene Unterordnungsbeziehungsarten an. In dem obigen Beispiel bezeichnet `SUBJ` das grammatische Subjekt, das Label `ADVL` ein Adverbial, das Label `PP` eine untergeordnete Präpositionalgruppe und das Label `DET` den Determiner eines Nomens. Die Trennzeichen `#` repräsentieren die Knoten des Dependenzbaums.

Über gewichtete Kontraktionsregeln können Klammerpaare angegeben werden, die kontraktiert werden sollen. Hierbei kann auf Klammersymbole Bezug genommen werden, die sich innerhalb oder außerhalb der zu kontraktierenden Klammerung befinden. Wenn sich Klammersymbole innerhalb einer zu kontraktierenden Klammerung befinden, handelt es sich um eine nichtprojektive Kontraktion. Um die Kontraktionen zu gewichten, ist jede Regel mit einer nicht negativen reellen Zahl verknüpft. Die entsprechenden Automaten werden dann zu einen WFST zusammengefasst, der die Kontraktionen über das Umschreiben von Klammersymbolen und Trennzeichen auf Epsilon realisiert.

Die Eingabe der syntaktischen Analyse bildet ein FSA, der den Eingabesatz mithilfe von Klammersymbolen und Trennzeichen repräsentiert. Er enthält viele strukturelle Ambiguitäten, die initial über eine Lexikon-Komponente eingeschränkt sein können; eine genauere Beschreibung dieser Einschränkungen ist jedoch nicht beschrieben. Während des Parsing werden die Kontraktionsregeln iterativ mithilfe der Kompositionsoperation auf die Eingabe angewendet. Die Anzahl an Anwendungen ist hierbei parametrisiert und hat Einfluss auf die Multiplanarität der Analyse. Abschließend wird der resultierende WFST mit einem FSA komponiert, der nur das leere Wort akzeptiert. Wenn die Komposition nicht scheitert, ist die Analyse erfolgreich und der Eingabesatz grammatisch und der Definitionsbereich stellt das Parsingergebnis dar. Das bedeutet, dass in diesem Ansatz das Parsing darin besteht, die Eingabe auf das leere Wort zu reduzieren, wobei Gewichtungen verwendet werden, um zwischen alternativen Analysen zu disambiguieren. Solch ein Vorgehen wurde bereits in Didakowski (2010, Abschnitt 5.3) im Kontext des

Dependenzparsings als Constraint-Optimierungsproblem angewendet.

Um die WFSTs bei der Berechnung klein zu halten, wird vorgeschlagen, während dem Parsing partielle Analysen zu 'vergessen' und gleichzeitig Zwischenergebnisse über einen Memoisation-Ansatz abzulegen und wiederzuverwenden. Nach jeder einzelnen Anwendung der Kontraktionsregeln wird der resultierende WFST gespeichert. Daraufhin wird die zweite Projektion des WFSTs ermittelt und in die weitere Analyse gegeben. Dadurch werden partiell Analysen 'vergessen' und müssen nicht mehr in der fortlaufenden Analyse durchgeschleift werden. Final kann das Parsing-Ergebnis anhand der memoisierten Zwischenergebnisse berechnet werden. Ein ähnliches Vorgehen wird bereits in Didakowski (2008b) vorgeschlagen. Dort wird jedoch die Substitution herangezogen, um die Zwischenergebnisse zu 'vergessen' und wieder einzubringen.

In Yli-Jyrä et al. (2012) wird für den Parsing-Algorithmus ein konkreter Grammatikformalismus angegeben, der dem Regelformalismus von Gaifman (1965) nachempfunden ist. Die Kontraktionsregeln sind ungewichtet und prüfen lediglich auf Ausgeglichenheit von Klammerungen. Es wird ein spezieller Klammerungstyp verwendet, um fragmentäre Analysen zu einem umfassenden Dependenzbaum zu verbinden. Analysen die am wenigsten Klammerungen diesen Typs enthalten werden bevorzugt, wodurch robustes Parsing möglich wird. In Yli-Jyrä (2005b) wird ein Ansatz zur Umsetzung des Regelformalismus von Gaifman (1965) vorgestellt, der im weiteren Sinne einen FSIG-Ansatz darstellt. Dort wird im Kontext projektiver Dependenzbäume das Prüfen auf Ausgeglichenheit der Klammerungen über den Schnitt von FSAs realisiert. Hierauf fußt auch der Ansatz von Ginter et al. (2006) im Rahmen der Umsetzung der Link-Grammatik und planarer Link-Strukturen. Dort wird jedoch von der Ineffizienz solch eines Ansatzes in der Praxis berichtet.

4.1.2 Parsing als CSP und COP

Es existieren zwei grundsätzliche Ansätze zum Dependenzparsing als Constraint-Erfüllungs- oder -Optimierungsproblem. Dies ist einerseits die *Constraint Dependency Grammar* (CDG) zusammen mit deren Erweiterungen und andererseits die *Extensible Dependency Grammar* (XDG). Beide Ansätze werden im Folgenden kurz eingeführt.

Constraint Dependency Grammar

In Maruyama (1990a) wird der Formalismus *Constraint Dependency Grammar* (CDG) als ein nicht generativer Ansatz vorgestellt. Dort wird das Dependenzparsing als ein Disambiguierungsproblem über initial ambige Dependenzrelationen formuliert, bei dem es das Ziel ist, alle strukturellen Ambiguitäten zu eliminieren. Es ist der erste Ansatz, der Dependenzparsing als CSP umsetzt.

Die einzelnen Wörter eines Satzes werden als Variablen repräsentiert, deren Werte die Identität des übergeordneten Wortes und die Art der Unterordnungsbeziehung anzeigen. Wenn ein Wort unabhängig ist und keiner Unterordnung unterliegt, ersetzt ein reservierter Bezeichner die Identität des übergeordneten Wortes. Dieser steht für ein Dummy-Wurzelwort. Zu einem Wort können mehrere Variablen existieren, die verschiedene Rollen implementieren. Eine Rolle meint hier einen bestimmten Unterordnungstyp, wie z. B. die syntaktische Unterordnung. Eine vollständige Belegung der Variablen einer Rolle entspricht hierbei einer Dependenzstruktur.

Constraints stellen verundete Teilformeln einer prädikatenlogischen Formel dar, die ausschließlich universalquantifiziert ist und in Pränexform vorliegt. Über die Constraints können einzelne Kanten im Dependenzbaum 'angefasst' werden, um Bedingungen zu prüfen, die sowohl lexikalische Eigenschaften als auch Dominanzbeziehungen und lineare Präzedenzen betreffen und die verschiedene Rollen involvieren können. Hierbei sind nichtprojektive Strukturen dann möglich, wenn sie nicht explizit verboten sind.

Es soll ein Beispiel für eine vollständige Variablenbelegung gegeben werden. Hierzu wird sich auf genau eine Rolle beschränkt, und zwar auf die sprachliche Ebene der syntaktischen Unterordnung. Es sei für den Satz „Das Aargau will die Frau aber nicht besuchen“ folgende Variablenbelegung gegeben: v_1 = $(\texttt{DET}, 2)$,v_2 = $(\texttt{OBJA}, 8)$,v_3 = $(\texttt{S}, \texttt{ROOT})$,$v_4$ = $(\texttt{DET}, 5)$,v_5 = $(\texttt{SUBJ}, 3)$, v_6 = $(\texttt{ADV}, 8)$, v_7 = $(\texttt{ADV}, 8)$, v_8 = $(\texttt{AUX}, 3)$. Die Nummerierung der Variablen korrespondiert mit den Positionen der Wörter im Satz. `ROOT` ist der reservierte Bezeichner für das Dummy-Wurzelwort. Das syntaktische Label `DET` bezeichnet den Determiner eines Nomens, das Label `SUBJA` das grammatische Subjekt, das Label `OBJA` das Akkusativobjekt, das Label `ADV` adverbiale Modifikation, das Label `AUX` ein nicht finites Verb innerhalb einer Verbgruppe und das Label `S` den Wurzelknoten eines vollständigen Satzes. In Abbildung 4.3 ist der Dependenzbaum zu dieser Belegung gegeben. Er ist nichtprojektiv,

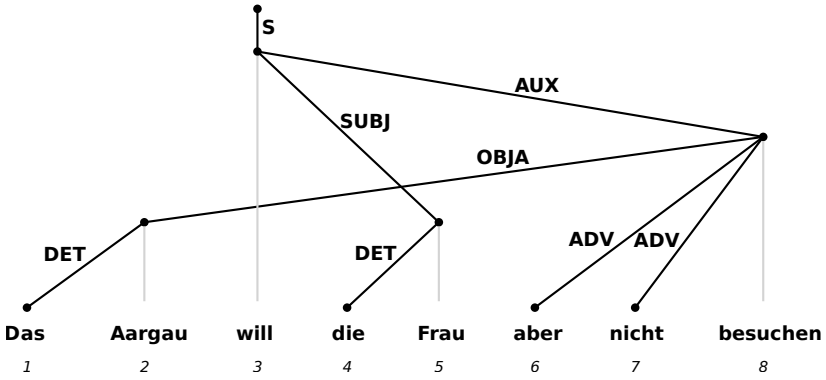


Abbildung 4.3: Dependenzbaum zu einer Variablenbelegung.

da er eine kreuzende Kante enthält.

Um das Parsing effizient umzusetzen, wird Constraint-Propagierung vorgeschlagen. Es wird gezeigt, dass im Fall eines binären CSP die Analyse eines Satzes der Länge n eine Zeitkomplexität von $O(n^4)$ besitzt.

Um Obligatorität im Kontext der Erfüllung von Subkategorisierungsrahmen zu implementieren, werden in Maruyama (1990b) spezielle Rollen vorgeschlagen, die umgedrehte Dependenzkanten beinhalten. Dies ist notwendig, weil dem Formalismus ein Existenzquantor fehlt, um zu fordern, dass mindestens ein bestimmtes Element des Subkategorisierungsrahmens realisiert sein muss.

In Maruyama (1990c) wird gezeigt, dass aus jeder kontextfreien Grammatik in Greibach-Normalform eine CDG erstellt werden kann, die die gleiche Sprache generiert. Des Weiteren wird nachgewiesen, dass die CDG sogar Sprachen aus dem schwach-kontextsensitiven Bereich wie z. B. die Kopiersprache beschreiben kann. In Harbusch (1997) wird ergänzend hierzu dargelegt, dass die CDG mächtiger ist als die *Tree-Adjoining-Grammar* (TAG) und dass jede TAG in eine äquivalente CDG überführt werden kann.

In Helzerman und Harper (1992) wird ein Parallelisierungsansatz vorgestellt, um das CDG-Parsing zu beschleunigen. In Helzerman et al. (1992) wird im Kontext der maschinellen Spracherkennung und dem syntaktischen Parsing eine Erweiterung des CDG-Ansatzes vorgeschlagen, um Wortgraphen zu verarbeiten, die mehrere Satzhypothesen enthalten. Hierzu wird eine simultane Repräsentation vorgestellt, welche in Helzerman und Harper (1996) als MUSE-CSP bezeichnet wird (vgl. Abschnitt 3.4.2). In Harper et al. (1993) werden semantische Merkmale herangezogen, um semantisch inkonsistente

Satzhypothesen auszuschließen. Diese werden in die Werte der syntaktischen Unterordnungen codiert. In Harper und Helzerman (1995, Abschnitt 3) wird CDG im Kontext des textbasierten Parsings um die Behandlung von lexikalischen Ambiguitäten und um die Behandlung von morphologischen Merkmalen wie Kasus, Numerus und Genus erweitert.

In Heinecke et al. (1998) wird CDG als Constraint Optimierungsproblem reformuliert. Hierzu werden Gewichtungen einbezogen, die mit den Regeln einer Grammatik und somit mit den Constraints verknüpft sein können. Hierdurch wird es möglich, verletzbare bzw. weiche Regeln zu formulieren. Diese Erweiterung ist innerhalb des Frameworks der *Weighted Constraint Dependency Grammar* (WCDG) implementiert (siehe Foth et al., 2004 oder Foth et al., 2005). Dieses erlaubt es, Gradation zu modellieren, die nicht auf die Begriffe Kompetenz und Performanz beschränkt ist, sondern viele andere Aspekte beinhaltet, bei denen gewichtete Informationen nachweislich hilfreich sind („... we therefore assume gradation not to be restricted to competence or performance but to include many more aspects where weighted information proves helpful“, Schröder, 2002, Abschnitt 1.1). Diese Perspektive macht es möglich graduelle Grammatikalität, strukturelle Präferenzen, unsichere Informationen und Defaults einzubeziehen. Hierdurch wird WCDG fehlertolerant und robust. Des Weiteren werden zusätzliche Rollen vorgeschlagen, um referenzielle Beziehungen und thematische Rollen zu behandeln. Jedoch ist das Parsing einer WCDG NP-vollständig und Methoden zur Konsistenzprüfung sind nicht unbedingt hilfreich, da sie den Suchraum nur geringfügig und im schlechtesten Fall gar nicht einengen würden.

In Foth et al. (2000b) wird ein transformationsbasierter Lösungsansatz vorgeschlagen, um eine exhaustive Suche beim Problemlösen zu vermeiden. Die Idee dahinter ist, die Analyse mit einer vollständigen Dependenzstruktur zu starten und diese schrittweise zu reparieren. Hierbei kann ein Ergebnis auch dann zurückgegeben werden, wenn der Lösungsalgorithmus vorzeitig gestoppt wird. Es besteht die sogenannte *Anityme-Eigenschaft* (anityme property). Es zeigt sich, dass die Laufzeit eines solchen Verfahrens in der Praxis deutlich besser ist als eine vollständige Suche. Es kann jedoch nicht gewährleistet werden, dass das Ergebnis der tatsächlichen optimalen Analyse entspricht. In der Praxis sind die Analyseergebnisse aber vergleichbar mit denen einer vollständigen Suche. In Erweiterung hierzu wird in Foth et al. (2003) vorgeschlagen, einen Satz aufzuspalten und die Einzelteile separat mithilfe

eines transformationsbasierten Verfahrens zu analysieren und die Teilergebnisse später zu einem globalen Ergebnis zu verbinden. Auf diese Weise kann einerseits Analysezeit eingespart und andererseits die gesparte Zeit für eine intensivere Analyse der Einzelteile aufgewendet werden.

In McCrae et al. (2008) wird eine WCDG-Erweiterung vorgestellt, die es ermöglicht, nicht lokale Phänomene zu erfassen. Hierzu werden die beiden Prädikate *has* und *is* vorgeschlagen, über die Bedingungen ausgedrückt werden können, die für die direkt untergeordneten Kanten eines Wortes (has) oder für die direkt übergeordnete Kante eines Wortes (is) gelten. Diese Beschränkungen werden rekursiv über unäre *Hilfsconstraints* (ancilla constraints) geprüft. Ein Constraint, der diese Prädikate einbezieht, wird als *überlokaler Constraint* (supra-local constraint) bezeichnet. Das Prüfen eines überlokalen Constraints ist jedoch nur dann möglich, wenn eine komplette Dependenzstruktur, also eine vollständige Belegung der Variablen, vorliegt. Dies ist beispielsweise bei einem transformationsbasierten Lösungsansatz stets der Fall.

Extensible Dependency Grammar

In Duchier (1999) wird ein Constraint-basierter Ansatz zum Dependenzparsing vorgeschlagen, bei dem das Parsing als ein endliches Konfigurationsproblem über Graphen mit gelabelten Kanten im Rahmen eines CSPs behandelt wird. In dieser Repräsentationsform wird ein Wort eines Satzes durch einen Knoten im Graphen dargestellt und dessen untergeordneten Wörter durch dessen Töchter. Der Ansatz erlaubt es, über syntaktische Wohlgeformtheitsregeln lexikalisierte Bedingungen zwischen dem übergeordneten Wort und seinen direkt untergeordneten Wörtern auszudrücken und er erlaubt es, Subkategorisierungsrahmen und Informationen über den Yield einzubeziehen. Diese Regeln werden in eine Constraint-Programmiersprache überführt, sodass mithilfe eines Constraint-Solvers über Propagierungstechniken und Nebenläufigkeit eine effiziente Verarbeitung möglich ist.

In Duchier und Debusmann (2001) wird eine Erweiterung des Ansatzes vorgeschlagen, die es ermöglicht, lineare Präzedenz zu modellieren. Hierzu werden zwei Ebenen eingeführt: eine für unmittelbare syntaktische Unterordnungen und eine zusätzliche für topologische Felder. Die erste beschreibt ungeordnete Dependenzbäume, die nichtprojektiv sein können, die zweite hingegen projektive Dependenzbäume über topologische Abhängigkeiten. In-

nerhalb der zweiten Ebene sind die Knoten zusätzlich gelabelt, um die Positionierung von den übergeordneten zu den untergeordneten Wörtern auszudrücken. Die Menge an Labeln ist hierbei total geordnet, wodurch eine Ordnung des Baumes und eine Umsetzung von linearen Beschränkungen möglich werden. Unter der Annahme, dass die zusätzliche Ebene die fehlende Ausdrucksmöglichkeit bezüglich der linearen Präzedenz kompensiert, bestehen gewisse Parallelen zu Maruyama (1990b) im Kontext der CDG. Dort werden zusätzliche Ebenen vorgeschlagen, um Subkategorisierungsrahmen zu implementieren. Jedoch werden über diese keine komplexen Baumstrukturen aufgebaut und sie sind ausschließlich technischer Natur und haben keinen linguistischen Hintergrund. In Duchier und Debusmann (2001) hingegen bildet die zusätzliche Ebene eine komplexe linguistisch motivierte Struktur.

In Debusmann et al. (2004) wird die *Extensible Dependency Grammar* (XDG) als ein Grammatikformalismus vorgeschlagen, der auf den Prinzipien der Dependenzgrammatik beruht und eine parallele Grammatik-Architektur ermöglicht. Der Ansatz fußt auf der Arbeit von Duchier (1999) und generalisiert den Ansatz von Duchier und Debusmann (2001) auf eine unbegrenzte Anzahl von sprachlichen Ebenen. Hierdurch wird es möglich, zusätzlich zur Syntax auch semantische Aspekte einzubeziehen. Im Speziellen wird eine Ebene für die Prädikat-Argument-Struktur und eine für den Skopus von natürlichsprachlichen Quantoren vorgeschlagen. Regeln können für die einzelnen Ebenen separat formuliert werden und es können Regeln definiert werden, die die Ebenen miteinander in Beziehung setzen. Erstaunlich ist hierbei, mit welcher Konsequenz die Arbeiten von Menzel (1995) und Menzel (1998) und Heinecke et al. (1998) ignoriert werden, welche im Kontext der Constraint-Verarbeitung längst die Behandlung verschiedener sprachlicher Ebenen wie Syntax und Semantik über eine Modularisierung in unterschiedliche Beschreibungsebenen thematisieren.

In Debusmann, 2007a wird gezeigt, dass aus jeder lexikalisierten kontextfreien Grammatik eine XDG erstellt werden kann, die die gleiche Sprache generiert. Eine kontextfreie Grammatik in Greibach-Normalform ist in diesem Zusammenhang auch eine lexikalisierte kontextfreie Grammatik, was Parallelen zur CDG deutlich macht. Des Weiteren wird gezeigt, dass XDG auch Sprachen aus dem schwach kontextsensitiven Bereich beschreiben kann und dass es möglich ist, Scrambling zu modellieren, wobei der schwach kontext-

sensitive Rahmen verlassen wird.[3] Diese Ausdrucksstärke hat jedoch ihren Preis. So wird in Debusmann, 2007b gezeigt, dass die Komplexität des Parsens eines Satzes innerhalb von XDG bei fixer Grammatik und fixem Lexikon NP-vollständig ist.

4.2 Naive WDG-Umsetzung

In diesem Abschnitt wird beschrieben, wie mit den in dieser Arbeit entwickelten Mitteln der WCDG-Ansatz mithilfe von FSMs als SCSP umgesetzt werden kann. Der Fokus liegt hier auf dem Parsen geschriebener, natürlicher Sprache. Die Umsetzungsbeschreibung ist in seinen Grundzügen auch in Didakowski (2010, Abschnitt 5.2) zu finden. Es werden Probleme aufgezeigt, die mit einer naiven Umsetzung des WCDG-Ansatzes einhergehen. Hierzu wird zuerst auf das Constraint-System, die Wahl der Gewichtsstruktur und die Repräsentation der Variablenmenge eingegangen. Daraufhin wird eine Umsetzung mit klassischen Constraints und darauffolgend mit quantifizierten regulären Constraints vorgestellt.

4.2.1 Constraint-System

Eine CDG kann über ein SCSP umgesetzt werden, das über den Booleschen Semiring ein klassisches CSP implementiert und mithilfe von FSMs repräsentiert ist. Die Werte des SCSPs sind dann Paare, bestehend aus einem Label, das den Unterordnungstyp anzeigt, und der Position des übergeordneten Wortes. Der Unterordnungstyp kann innerhalb der Zeichenkettenrepräsentation als ein Zeichen repräsentiert werden. Die Position des übergeordneten Wortes hingegen ist eine Zahl und kann über eine Zeichenkette abgebildet werden, die diese codiert (siehe Abschnitt 3.2.1).

Hier liegt jedoch der Fokus auf der Umsetzung des WCDG-Ansatzes und somit auf einer anderen Gewichtsstruktur. In dem Framework, auf dem der Formalismus der WCDG basiert, wird die beste Dependenzstruktur anhand von quantitativen Präferenzen ermittelt. Dieses Optimierungsproblem

[3]Scrambling bezeichnet Transformationen, die aus zugrunde liegenden Strukturen Oberflächenstrukturen mit unterschiedlicher Wort- und Satzgliedstellung erzeugen. Scrambling ist in zweierlei Hinsicht ungebunden: Es kann mehr als ein Element bewegt werden und die Bewegung ist unbegrenzt.

ist als Instanz des *bewerteten Constraint-Erfüllungsproblems* (Valued Constraint Satisfaction Problem, VCSP, Schiex et al., 1995) über das *multiplikative CDG-VCSP* implementiert (siehe Schröder, 2002, Abschnitt 2.2.3). Trägermenge des total geordneten kommutativen Monoiden bzw. der Bewertungsstruktur ist das Intervall von 0 bis 1, dessen Werte den Grad der Regelverletzung einer WCDG angeben. Hierbei steht $0,0$ für das schlechteste und $1,0$ für das beste Bewertungselement. Die abstrakte Operation der Bewertungsstruktur, mit der die Verletzungen verrechnet werden, ist die klassische Multiplikation. So wird sich beim Aufmultiplizieren der Regelverletzungen weicher Constraints stets dem schlechtesten Bewertungselement angenähert. Es werden demnach negative Präferenzen implementiert und es werden über die Ordnung der Bewertungsstruktur jeweils die Elemente präferiert, die eine leichtere Verletzung darstellen.

Im Kontext des SCSPs können die Verletzungen über ein gewichtetes CSP mithilfe des c-Semirings $(\mathbb{R}_0^+ \cup \{+\infty\}, min, +, +\infty, 0)$ umgesetzt werden. Diese Instanz implementiert über die abstrakte Addition die gewünschten negativen Präferenzen und über den negativen Logarithmus können Verletzungen von 0 bis 1 in die Trägermenge $\mathbb{R}_0^+$ überführt werden. Hierbei korrespondiert $-log(1)$ mit dem Einselement des c-Semirings; mithilfe dieses Wertes kann ein Universal-Constraint umgesetzt werden. $-log(0)$ korrespondiert hingegen mit dem Nullelement des c-Semirings; über diesen Wert kann ein harter Constraint realisiert werden.

Innerhalb einer WCDG ist es im Gegensatz zur CDG möglich, gültige Label für die einzelnen Rollen anzugeben und somit verschiedene Wertebereiche einzubeziehen. Dies kann innerhalb eines SCSPs nicht abgebildet werden, da lediglich ein globaler Wertebereich für alle Variablen existiert. Alternativ kann aber angenommen werden, dass die Belegungen für die einzelnen Rollen über unäre Constraints eingeschränkt sind.

Für diese Repräsentationsform einer Variable und eines Satzes soll ein konkretes Beispiel gegeben werden. Hierfür sei genau eine Rolle angenommen, die die syntaktische Unterordnung behandelt. Mögliche syntaktische Label für diese seien `S`, `SUBJA`, `OBJD`, `OBJP`, `PP`, `GMOD`, `ATTR` und `DET`. Des Weiteren sei davon ausgegangen, dass die Wortpositionen von 1 bis 99 möglich sind und dass einem unabhängigen Wort die 0 als Wurzelposition zugewiesen wird. Dann kann eine Variable durch den FSA in Abbildung 4.4 repräsentiert werden. Dieser akzeptiert beispielsweise die Werte $(\texttt{S}, 00)$, $(\texttt{SUBJ}, 12)$

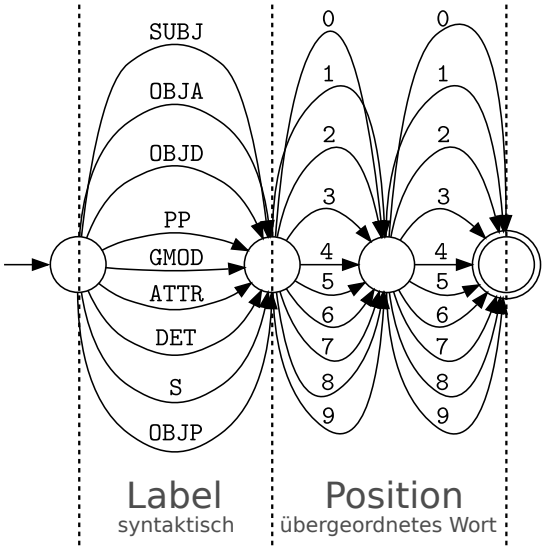


Abbildung 4.4: Repräsentation einer Variable als FSA.

und $(\texttt{PP}, 65)$. Da eine Position über zwei Zeichen codiert ist, wird die 0 durch '00' wiedergegeben. Des Weiteren wird angenommen, dass die Ordnung der Variablen der Reihenfolge der Token im Satz entspricht. Dann wird der Satz „Wir stehen vor der Berner Zytglogge" durch die Variablen $V = \{v_1, v_2, v_2, v_3, v_4, v_5\}$ mit $v_1 \prec v_2 \prec v_3 \prec v_4 \prec v_5 \prec v_6$ abgebildet, wobei v_1 für 'Wir', v_2 für 'stehen', v_3 für 'vor', v_4 für 'der', v_5 für 'Berner' und v_6 für 'Zytglogge' stehen. Die Position des übergeordneten Wortes korrespondiert hierbei mit der Position innerhalb der Variablenordnung. So hat beispielsweise v_2 die Position 2 und v_5 die Position 5.

Wenn mehrere Rollen innerhalb einer WCDG involviert sind, müssen für ein Wort mehrere Variablen angelegt werden. Wenn bezüglich der Rollen $r_1, r_2, ..., r_m$ die Ordnung $r_1 \prec r_2 \prec ... \prec r_m$ angenommen wird, werden für ein Wort k die Variablen $v_{k_{r_1}} \prec v_{k_{r_2}} \prec ... \prec v_{k_{r_m}}$ angelegt. Die Ordnung der Variablen ist dann wie folgt definiert: $(v_{1_{r_1}} \prec v_{1_{r_2}} \prec ... \prec v_{1_{r_m}}) \prec (v_{2_{r_1}} \prec v_{2_{r_2}} \prec ... \prec v_{2_{r_m}}) \prec ... \prec (v_{n_{r_1}} \prec v_{n_{r_2}} \prec ... \prec v_{n_{r_m}})$. Es folgt ein Beispiel für mehrere Rollen. Angenommen, es existiert zusätzlich zur syntaktischen Unterordnung eine Rolle für die semantische mit den Labeln `Agens`, `Patiens`, `Thema`, `Experiencer`. Des Weiteren sei davon ausgegangen, dass für die syntaktische Unterordnung (r_{syn}) und die semantische Unterordnung (r_{sem}) folgende Ordnung definiert ist: $r_{syn} \prec r_{sem}$. Dann kann die Variablenmenge zu einem Satz, der aus genau einem Wort besteht, durch den FSA in Abbildung 4.5

repräsentiert werden.

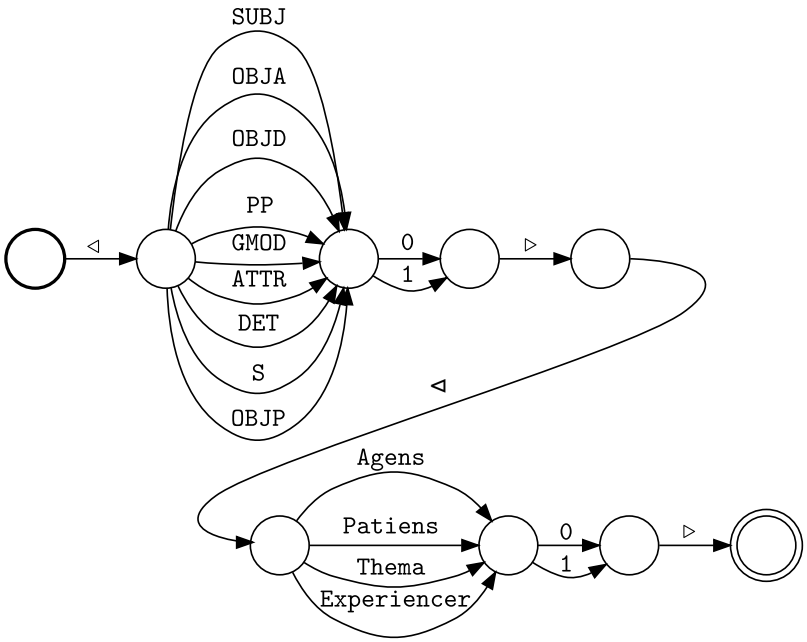


Abbildung 4.5: Variablenmenge für mehrere Rollen als FSA.

Lexikalische und morphologische Ambiguitäten können über ein simultanes SCSP behandelt werden, dessen Variablen mit Informationen über die Wortart und über lexikalische und morphologische Merkmale verknüpft sind. Als ein Beispiel für diese simultane Repräsentationsform soll das Wort 'lieben' betrachtet werden. Bei Ignorieren der Groß- und Kleinschreibung kann dieses unter anderem ein finites Verb (`VVFIN`) im Plural (`pl`) oder ein Nomen (`NN`) im Plural (`pl`) mit dem Geschlecht Neutrum (`neut`) und mit den möglichen Kasus Nominativ (`nom`), Akkusativ (`acc`) oder Dativ (`dat`) sein. Nun sei eine simultane Variablenmenge gegeben, deren Instanzen jeweils genau eine Variable enthalten. Jede dieser Variablen ist Repräsentant des Wortes 'lieben' in einer seiner morphologischen Lesarten. Des Weiteren sei die Rolle der syntaktischen Unterordnung angenommen. Dann ist der FSA, der die simultane Variablenmenge repräsentiert, in Abbildung 4.6 gegeben. Die morphologischen Merkmale sind über zusätzliche Zeichen den Wortarten nachgestellt. Diese Codierungsart ist analog zu der innerhalb des FSIG-Ansatzes (vgl. Koskenniemi, 1990). Über die simultane Variablenmenge sind auf diese Weise vier morphologische Lesarten kompakt repräsentiert.

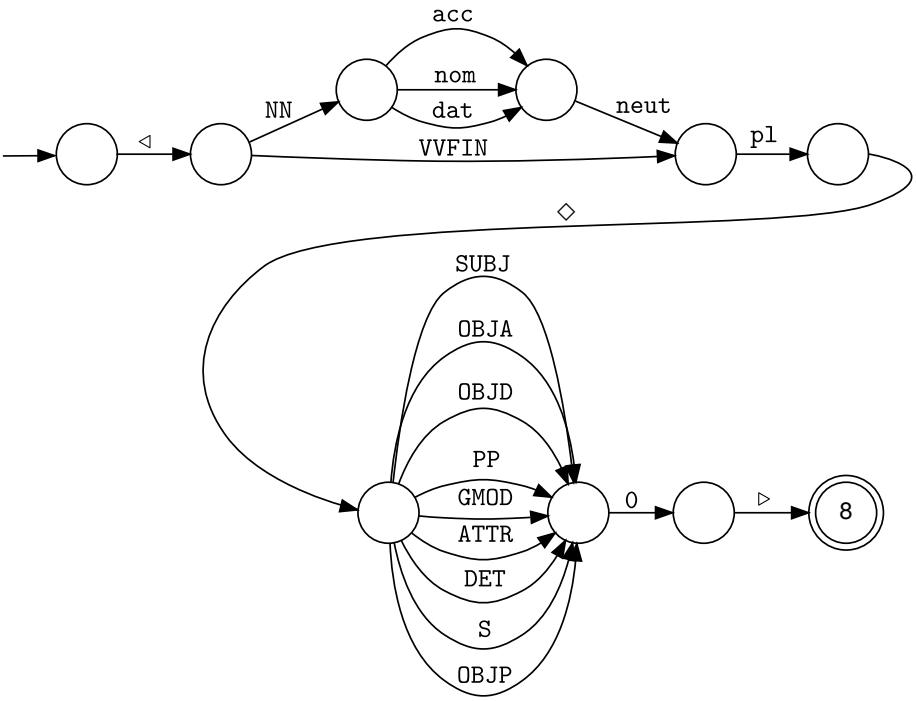


Abbildung 4.6: Simultane Variablenmenge als FSA.

Bei der Umsetzung von überlokalen Constraints muss innerhalb der WCDG eine vollständige Variablenbelegung vorliegen, um Hilfsconstraints gezielt anhand der syntaktischen Struktur anwenden zu können. Dies ist jedoch innerhalb des FSM-Ansatzes problematisch. Es existiert nicht wie beim transformationsbasierten WCDG-Ansatz stets genau eine momentane vollständige Variablenbelegung als Lösungskandidat (vgl. Foth et al., 2000b). Alternativ ist es aber möglich, mit den einzelnen Dependenz-Labeln Merkmale zu verknüpfen, die anzeigen, ob die Anwendung eines Hilfsconstraints zu einer Inkonsistenz führt oder nicht. Hierbei wird mit einem Dependenz-Label ein endlicher binärer Merkmalsvektor verknüpft, bei dem jedes Merkmal für einen Hilfsconstraint steht, der entweder eine 'is'- oder oder eine 'has'-Bedingung ausdrückt. Wenn die Bedingung eines Hilfsconstraints erfüllt ist, ist für ein entsprechendes Merkmal ausschließlich der Wert konsistent, der einen erfolgreichen Constraint-Check anzeigt. Wenn die Bedingung hingegen nicht erfüllt ist, ist nur der Wert konsistent, der für einen gescheiterten Constraint-Check steht. Wenn innerhalb der Hilfsconstraints wiederum auf Hilfsconstraint-Merkmale Bezug genommen wird, ist es möglich, syn-

taktische, semantische oder lexikalische Informationen über mehrere Unterordnungen hinweg zu transportiert. Beim Ermitteln der abstrakten Lösung werden die Hilfsconstraints wie normale Constraints behandelt, wodurch der Ansatz im Gegensatz zu McCrae et al. (2008) nicht auf unäre Hilfsconstraints beschränkt ist. Jedoch wächst der Wertebereich des Constraint-Systems exponentiell mit der Anzahl an Hilfsconstraints. In der Repräsentation mit FSMs kann dieser aber kompakt dargestellt werden. Es soll ein Beispiel für die Umsetzung von überlokalen Constraints gegeben werden. Hierzu werden zwei Hilfsconstraints angenommen, einerseits der Constraint c_{is}, der eine 'is'-Bedingung implementiert, und andererseits der Constraint c_{has}, der eine 'has'-Bedingung umsetzt. Über diese beiden sei die Ordnung $c_{is} \prec c_{has}$ definiert, die auch für die zugehörigen Merkmale gilt. Des Weiteren seien die Merkmalswerte 'true' und 'false' möglich, die auf Zeichenkettenebene als ein Zeichen repräsentiert werden und einen erfolgreichen oder gescheiterten Constraint-Check anzeigen. Dann ist der FSA, der einen Wertebereich unter Berücksichtigung dieser zusätzlichen Merkmale darstellt, in Abbildung 4.7 gegeben.

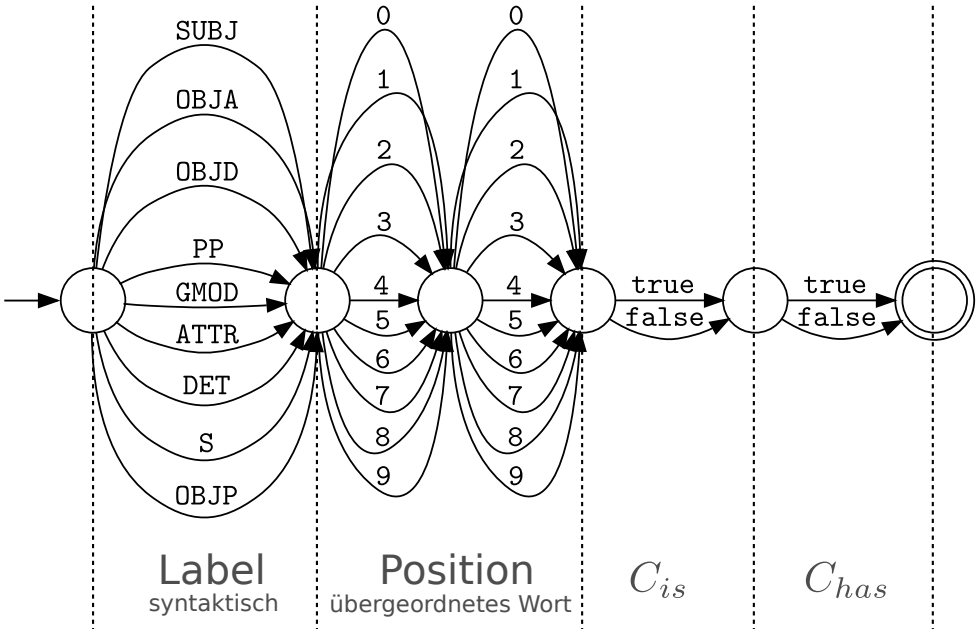


Abbildung 4.7: Variable mit Merkmalen für Hilfsconstraint-Checks als FSA.

4.2.2 Constraints

Anhand einer gegebenen WCDG-Grammatik mit all-quantifizierten Formeln und einem Eingabesatz können gewichtete Wertetupel generiert und in Constraints überführt werden, die mithilfe von WFSAs repräsentiert sind. Ausgehend von diesen Constraints kann dann innerhalb eines Constraint-Problems wie in Abschnitt 3.3.5 beschrieben die abstrakte Lösung berechnet werden, die die Analysen enthält, die den Regeln der WCDG am meisten genügen.

Beim Lösen eines SCSPs zu einer adäquaten WCDG würden sich die Zwischenergebnisse sehr stark aufblähen. Das Parsing ist vergleichbar mit der Anwendung einer FSIG, insbesondere wenn über ein simultanes SCSP lexikalische und morphologische Ambiguitäten abgebildet werden. Hierbei verschärft sich die Problematik der stark anwachsenden Zustandsanzahl der FSMs im Vergleich zur FSIG sogar noch zusätzlich, da die eindeutigen Unterordnungen einer CDG nicht so kompakt repräsentierbar sind wie die unterspezifizierten einer FSIG. Zwar wäre über die Trennung vom Wert und Typ eines Constraints ein abgemildertes Anwachsen möglich, jedoch wäre schnell der Punkt erreicht, an dem alle Variablen involviert sind. Zudem werden hier Analysestrukturen nicht nur über harte Constraints eingeschränkt, sondern auch über weiche bewertet. So müssen innerhalb der FSMs zusätzlich Gewichtungen repräsentiert werden, was das Problem unhandhabbarer Zwischenergebnisse verschärft. Zwar wäre wie in Didakowski (2008a) ein Verfahren denkbar, dass für Teilprobleme abstrakte Lösungen berechnet, um ein starkes Anwachsen der Zustandsanzahl der FSMs zu vermeiden. Jedoch wäre nicht klar, für welche Teilprobleme eine vorzeitige Disambiguierung angewendet werden sollte und inwieweit die abstrakten Lösungen die Lösung des gesamten (simultanen) SCSPs beeinflussen würden. Letztendlich müsste die Struktur des Satzes zumindest teilweise vorweg bekannt sein (siehe hierzu auch Foth et al., 2003). Eine grobe Segmentierung eines Eingabesatzes in seine Haupt- und Nebensätze würde das Problem beispielsweise nicht lösen können, da die Größe der Teilelemente immer noch unbeschränkt wäre. So müssten analog zur FSIG Strategien angewendet werden, um die FSMs handhabbar zu halten (vgl. Abschnitt 4.1.1).[4]

Eine andere Möglichkeit, das starke anwachsen der Zwischenergebnisse

[4]In Yli-Jyrä (2001) wird im Kontext der FSIG vorgeschlagen, das Nicht-Leerheitsproblem des Schnitts von FSAs auf ein CSP abzubilden, um das Problem der unhandhabbaren Zwischenergebnisse zu umgehen.

zu verhindern, wäre die Herstellung von Kantenkonsistenz, wie sie in Maruyama (1990a) im Kontext der CDG vorgeschlagen wird. So könnte beispielsweise, wie in Abschnitt 2.3.3 beschrieben, über lokale Konsistenzregeln Semiring-basierte (generalisierter) Kantenkonsistenz hergestellt werden. Dies ist jedoch problematisch, da die abstrakte Multiplikation des c-Semirings eines gewichteten CSPs nicht idempotent ist. Es ist aber möglich, über die c-Semiring-Struktur ein gewichtetes CSP auf ein klassisches CSP zu abstrahieren. Dieses enthält weniger Constraints, da dort weiche Constraints und somit weiche Regeln ignoriert werden. Über solch ein CSP kann dann die (generalisierte) Kantenkonsistenz berechnet werden. An dieser Stelle kann auch das Propagierungsverfahren des Regular-Membership-Constraints einbezogen werden. Die Anwendung des Propagierungsverfahrens für den Cost-Regular-Constraint ist hingegen nicht sinnvoll, da das Setzen der oberen und unteren Schranke für die Gewichtungen den gleichen Effekt hätte wie das Abstrahieren in ein klassisches CSP (vgl. Abschnitt 3.3.6). Jedoch hat die Umsetzung der Constraint-Propagierung über FSMs keinen echten Vorteil zu bestehenden Verfahren, die beispielsweise eine Parallelisierung der Berechnung der Kantenkonsistenz erlauben (siehe Helzerman und Harper, 1992 für CDGs mit der Stelligkeit 2). Zudem ist, wie in Abschnitt 4.1.2 erwähnt, der Nutzen der Kantenkonsistenzherstellung im Kontext der WCDG fraglich, da der Suchraum eventuell nur geringfügig eingeschränkt werden kann. Inwieweit ein Verringern des Suchraums möglich ist, hängt hierbei von der Art der WCDG-Regeln und dem Verhältnis von weichen und harten Constraints ab.

4.2.3 Quantifizierte reguläre Constraints

Eine weitere Umsetzungsvariante der WCDG ist die Kompilation eines monolithischen quantifizierten regulären Constraints. Dieser kann auf die Variablenmenge eines entsprechenden Constraint-Problems projiziert werden. So müssen Wertetupel nicht mehr für jeden Eingabesatz dynamisch generiert und kompiliert werden.

Die Menge an Positionen für die übergeordneten Wörter muss hierbei vorab gesetzt sein, ansonsten wäre es nicht möglich, die Äquivalenz zweier Positionen (hier die Positionsinformation eines Variablenwertes und die konkrete Position eines Wortes) auszudrücken. Dies impliziert aber auch eine

begrenzte Satzlänge für die Eingabe des Parsing-Ansatzes.[5] Für eine praktische Anwendung wäre solch ein Limit jedoch hinnehmbar.

Angenommen es existiert solch ein monolithischer quantifizierter regulärer Constraint. Dann wird Bei der Projektionsoperation die Komposition aus einem FST T, der die Tupel-Projektion implementiert, und dem WFSA R, der den Wert des quantifizierten regulären Constraints repräsentiert, berechnet. Die Komplexität der Komposition ist hierbei $O(|T| \times |R|)$. Angenommen, eine WCDG ist fest und die Menge an Wortpositionen ist gebunden, dann ist auch der Wert eines entsprechenden quantifizierten Constraints fest. Dessen Größe kann daher als Konstante C angesehen werden. So ist die Zeitkomplexität des Parsings in $O(|T|) = O(C \times |T|)$, wobei $|T|$ mit der Menge an involvierten Variablen und somit mit der Satzlänge wächst (vgl. Tapanainen, 1997, Abschnitt 10.7.3 im Kontext der FSIG). Jedoch kann C eine sehr große Konstante sein.

Angenommen, es existiert eine WCDG mit genau einer Rolle, dann kann der Constraint-Wert, der alle Regeln umfasst, über folgende reguläre prädikatenlogische Formel spezifiziert werden:

$$\begin{aligned} \forall x_1, x_2, \ldots, x_k \; Var(x_1) \wedge Var(x_2) \wedge \cdots \wedge Var(x_k) \rightarrow \\ (\neg F_1 \rightarrow Score(x_1, S_{F_1})) \\ \wedge \\ (\neg F_2 \rightarrow Score(x_1, S_{F_2})) \\ \wedge \\ \vdots \\ \wedge \\ (\neg F_n \rightarrow Score(x_1, S_{F_n})) \end{aligned} \tag{4.1}$$

Das Prädikat $Var/1$ bezeichnet, wie in Abschnitt 3.4.3 beschrieben, die Eigenschaft, eine Variable zu sein und das Prädikat $Score/1$, wie in Abschnitt 3.4.3 beschrieben, die Eigenschaft einer Variable mit einem konkreten c-Semiring-Element als Strafgewicht verknüpft zu sein. Die Ausdrücke S_{F_1}, $S_{F_2}, \ldots, S_{F_n}$ bezeichnen rationale Sprachen, die die Gewichte zu den CDG-Regeln $F_1, F_2, \ldots, F_n$ auf Zeichenkettenebene codieren. Diese werden innerhalb der Formel der quantifizierten logischen Variable x_1 zugewiesen.

[5] Zur gebundenen Satzlänge im Zusammenhang mit endlichen Techniken siehe auch Mohri und Sproat (2006, Abschnitt 3).

Wenn mehrere Rollen berücksichtigt werden sollen, muss die Information, welche Variable welcher Rolle zuzuordnen ist, den entsprechenden Variablen hinzugefügt werden. Dann können Prädikate umgesetzt werden, die die Eigenschaft einer Variable bezeichnen, einer syntaktischen oder semantischen Rolle zuzugehören.

Innerhalb des WCDG-Formalismus kann, auf die Wortinformationen eines übergeordneten Wortes über genau eine quantifizierte logische Variable zugegriffen werden. Dies ist hier nicht möglich. Es können allerdings zusätzliche quantifizierte logische Variablen angenommen werden, um dieses Problem zu umgehen. Bei der prädikatenlogischen Formel muss beachtet werden, dass bei den Gewichtszuweisungen optional auch alle höheren Gewichte einbezogen werden müssen, um Gewichtskonflikte zu vermeiden (siehe hierzu Abschnitt 3.4.3). Jedoch werden die Gewichtungen verschiedener Regeln immer noch nicht korrekt behandelt. Wenn einer Variable über zwei Regeln zwei unterschiedliche Gewichte zugewiesen werden, wird letztlich das schlechtere der beiden gewählt. Hier sollten die Gewichte jedoch abstrakt multipliziert werden.

Um die Gewichtungen korrekt zu behandeln, können die einzelnen WCDG-Regeln separat in Constraint-Werte kompiliert und über das Hadamard-Produkt zusammengefasst werden. Gegeben die Constraint-Werte χ_{F_1},χ_{F_2},…, χ_{F_n}, wobei χ_{F_r} mit $1 \leq r \leq n$ über die reguläre prädikatenlogische Formel $\forall x_1, x_2, \ldots, x_k \; Var(x_1) \wedge Var(x_2) \wedge \cdots \wedge Var(x_k) \rightarrow (\neg F_r \rightarrow Score(x_1, S_{F_r}))$ spezifiziert ist, dann ist der Constraint-Wert, der alle Regeln der WCDG umfasst, definiert durch $\chi_{F_1} \cap \chi_{F_2} \cap \cdots \cap \chi_{F_n}$. Nun werden die entsprechenden Gewichte abstrakt multipliziert. Dies ist analog zu der Zusammenfassung von Constraints gleichen Typs innerhalb von Constraint-Problemen. Jedoch kann es immer noch vorkommen, dass innerhalb einer Regel die Gewichtungen nicht wie gewünscht behandelt werden. Das ist der Fall, wenn dieselbe Regel mehrere Male verletzt und das Strafgewicht derselben Variable zugeschrieben wird. Dann werden die Gewichtungen der einzelnen Verletzungen weiterhin nicht abstrakt multipliziert. Dies kann jedoch umgangen werden, indem das Strafgewicht den logischen Variablen zugewiesen wird, die bei einer Mehrfachverletzung nicht zu solch einer Problematik führen würden.

Innerhalb der WCDG ist es möglich, Regeln mit einem dynamischen Gewicht zu verbinden. Das Gewicht eines sogenannten *dynamischen Constraints* (dynamic constraint) ergibt sich aus einer konkreten Parsing-Eingabe bzw.

aus einem konkreten Constraint-Problem. Solche Constraints lassen sich nur bedingt über quantifizierte reguläre Constraints implementieren. Das Einstreuen von c-Semiring-Elementen kann ausschließlich im regulären Rahmen geschehen und diese können ausschließlich über die abstrakte Addition verrechnet werden. Komplexere Formeln zur Gewichtsberechnung, wie sie z. B. in Foth et al., 2005, Abschnitt 3.2 für eine Distanzregel vorgeschlagen werden, sind nicht möglich. Aber es lässt sich beispielsweise ein quantifizierter regulärer Constraint implementieren, der kurze oder lange Anbindungen präferiert. Jedoch ist im Fall der dynamischen Constraints nicht sichergestellt, dass der entsprechende WFSA die Zwillingseigenschaft besitzt und determinisierbar ist.

Das oben beschriebene Vorgehen zur Umsetzung der WCDG mithilfe von FSMs ist für eine praktische Anwendung problematisch. In der Praxis wäre es schon bei einer WCDG mit zweistelligen Constraints, wenigen Regeln und einem moderat gesetzten Maximum für die Satzlänge unmöglich, einen monolithischen quantifizierten regulären Constraint, der alle Regeln der Grammatik umfasst, über eine reguläre prädikatenlogische Formel zu kompilieren. Ein Grund hierfür ist, dass die quantifizierten Constraints für jeden Skopus und für jede mit einer Variable verknüpften Information wohldefiniert sind. Hierdurch müssen verschiedenste Varianten extensional in einem FSM repräsentiert werden. Da die Kompilierung einer prädikatenlogischen Formel generell sehr aufwendig ist, kann dies zu unhandhabbaren FSMs führen (siehe Abschnitt 2.2.2).

Angenommen, die einzelnen Regeln einer WCDG sind separat in quantifizierte reguläre Constraints kompiliert, dann wäre es auch möglich, für einen bestehenden Eingabesatz erst ein Constraint-Problem für jeden dieser quantifizierten regulären Constraints gesondert zu lösen. Die Lösungs-Constraints könnten danach innerhalb eines nachgeschalteten Constraint-Problems verwendet werden, um die Lösung anhand aller WCDG-Regeln zu berechnen. Gegeben seien die einzelnen Constraints $c_{F_1} = (\chi_{F_1}, L(con)), c_{F_2} = (\chi_{F_2}, L(con)), \ldots, c_{F_n} = (\chi_{F_n}, L(con))$, wobei *con* die Variablenmenge ist, die den Eingabesatz repräsentiert, und χ_{F_r} mit $1 \leq r \leq n$ über die prädikatenlogische Formel $\forall x_1, x_2, \ldots, x_k \; Var(x_1) \wedge Var(x_2) \wedge \cdots \wedge Var(x_k) \rightarrow F_r$ spezifiziert ist. Dann ist das Constraint-Problem für das Parsen eines Eingabesatzes de-

finiert durch:

$$P = (\{Sol((\{c_{F_1}\}, con)), Sol((\{c_{F_2}\}, con)), \ldots, Sol((\{c_{F_n}\}, con))\}, con) \tag{4.2}$$

Hier bestehen deutliche Parallelen zur Anwendung einer FSIG, bei der die Regelautomaten erst einzeln mit dem Satzautomaten und dann miteinander geschnitten werden. Der Vorteil ist hier, dass der Wert genau eines allumfassenden quantifizierten regulären Constraints, der für alle Skopen und für alle mit den Variablen verknüpften Informationen wohldefiniert ist, nicht vorweg kompiliert werden muss. Des Weiteren können Heuristiken für die Abfolge der einzelnen Schnittbildungen einbezogen werden, um das Anwachsen von Zwischenergebnissen abzumildern (vgl. Koskenniemi et al., 1992, Abschnitt 4.2 oder Tapanainen, 1997, Abschnitt 10.3.3). Jedoch ist hier letztlich das Problem unhandhabbarer Zwischenergebnisse im Gegensatz zur FSIG noch weiter verschärft und es bleibt fraglich, ob sich die WCDG-Regeln wegen der Zustandskomplexität in der Praxis überhaupt in entsprechende quantifizierte reguläre Constraints überführen lassen.

4.2.4 Innewohnende Schwierigkeiten

Eine naive Herangehensweise bei der Umsetzung der WCDG mithilfe von FSMs innerhalb des SCSP-Frameworks führt zu einem Problem, das im Kontext der FSIG allzu gut bekannt ist: die Zustandskomplexität. Einerseits ist es problematisch, Regeln vorweg in einen monolithischen Automaten zu kompilieren und andererseits können sich FSMs bei einer Analyse, die auf der Schnittbildung beruht, sehr schnell stark aufblähen. Bei der WCDG-Umsetzung sind die Probleme sogar noch weiter verschärft, da mit expliziten und nicht mit unterspezifizierten Unterordnungen aber auch mit Gewichtungen umgegangen werden muss.

Nichtsdestotrotz herrscht in der FSIG-Community weiterhin der Konsens, dass ein effizienter FSIG-Parser möglich ist. In Yli-Jyrä (2005a) wird angemerkt, dass hierfür die Grammatik in solch einer Weise zerlegt sein muss, dass sie eine kompakte Repräsentation während der einzelnen Analyseschritte beim Parsing erlaubt und dass Lokalität und Dekomposition herangezogen werden müssen, um Zwischenergebnisse zu speichern und um effizientes Parsing zu ermöglichen. In den folgenden Abschnitten wird dieser Forderung nachgegangen und es wird ein Vorgehen erarbeitet, das Lokalität und De-

komposition einbezieht.

4.3 Problemzerlegung und perfekte Relaxierung

In diesem Abschnitt wird beschrieben, wie das Parsing natürlicher geschriebener Sprache über einen Zerlegungsansatz innerhalb des SCSPs realisiert werden kann. Es wird auf den Begriff der Lokalität eingegangen, der grundlegend für den Ansatz ist. Darauf aufbauend wird gezeigt, auf welche Weise das Problem zerlegt werden kann und wie die Einzelteile miteinander verbunden sind. Abschließend wird gezeigt, dass über die vorgeschlagene Problemzerlegung eine perfekte Relaxierung mit Top-Down-Lösungsabfolge möglich ist.

4.3.1 Lokalität in der Grammatik

In Abschnitt 4.2 wurde gezeigt, wie das Dependenzparsing anhand einer WCDG mithilfe von FSMs umgesetzt werden kann. Problematisch sind hierbei FSMs von unhandhabbarer Größe, die während des Problemlösens auftreten können. Das Kernproblem liegt in dem Grammatikformalismus selbst. Dort können Abhängigkeiten definiert werden, die eine kompakte Repräsentation von Zwischenergebnissen erschweren. Der Grund hierfür ist fehlende Lokalität in der Grammatik (vgl. im Kontext der FSIG Yli-Jyrä, 2005a, Abschnitt 17.2.4).

Es soll der Begriff der Lokalität, wie er hier verstanden wird, genauer erläutert werden. Hierzu soll der WCDG-Problemlösungsansatz von Foth et al. (2003) betrachtet werden: Es wird im Kontext eines transformationsbasierten Lösungsansatzes vorgeschlagen, das Parsing-Problem in kleinere Teilprobleme aufzuspalten, wobei eine flache Anordnung von Teilproblemen angenommen wird, die nicht weiter rekursiv zerlegt sind. Die Lösungen der einzelnen Teilprobleme werden erst ermittelt und dann für die Lösungsfindung des gesamten Problems rekombiniert. Diese Rekombinierung beinhaltet eventuell eine Revidierung bisheriger Wertezuweisungen. Bei diesem Ansatz wird ausgenutzt, dass die Unterordnungsbeziehungen implizit mit Strukturen korrespondieren, die als rekursiv eingebettete Bäume betrachtet werden können. Wenn solch ein Baum einen Bereich beschreibt, der in sich geschlossen ist, kann für ihn unabhängig eine Lösung berechnet werden. Um solche Berei-

che zu definieren, muss die syntaktische Struktur eines Satzes jedoch vorweg bekannt sein. Daher wird dort einer imperfekten Lösung nachgegangen, bei der versucht wird, solche Bereiche anhand von Satzzeichen zu erraten. Dieser Ansatz implementiert hierbei keine perfekte Relaxierung, da Belegungen für ein Teilproblem revidiert werden können. Eine perfekte Relaxierung wäre hier nicht umsetzbar, da es innerhalb einer WCDG-Regel möglich ist, zwei Kanten im Dependenzbaum anzufassen, die nicht direkt miteinander verbunden sind. Solch ein Beschränkungsbereich kann stets über zwei Subprobleme hinweg gelten. Daher können Teilprobleme einer Problemzerlegung, die sich an der syntaktischen Struktur orientieren, nicht klar voneinander abgegrenzt werden. Zudem können innerhalb der WCDG mehrere Rollen einbezogen werden, wodurch im Extremfall komplett unterschiedliche komplexe Strukturen berücksichtigt werden müssen. Lokal meint in dieser Arbeit demnach, dass die Beschränkungen, die innerhalb einer Grammatik definiert werden können, eine klare Abgrenzung von Teilproblemen erlauben.

4.3.2 Syntaxnahe Zerlegung

Angenommen, die komplette syntaktische Struktur eines Satzes sei vorweg bekannt. Dann kann eine Problemzerlegung abgeleitet werden, die baumartig organisiert ist und sich stark an der Syntaxstruktur orientiert. Die einzelnen Teilprobleme können bottom-up über eine perfekte Relaxierung gelöst werden, indem der Zerlegung und somit der syntaktischen Struktur von den Blättern bis zur Wurzel gefolgt wird. Variablen, die in einem Teilproblem aber in keinem anderen Folgeproblem enthalten sind, können frühzeitig aus dem Suchraum entfernt werden. Solch ein Vorgehen korrespondiert mit dem generellen Lösen eines CSPs mit *Baumzerlegung* (tree decomposition). Die grundlegende Idee dahinter ist, Inkonsistenz auf lokaler Ebene zu erkennen, wobei lokale Inkonsistenz zu globaler Inkonsistenz führt (siehe Dechter, 2003, Abschnitt 9.2.2).

Um eine Problemzerlegung abzuleiten, wird hier initial ein ungelabelter Dependenzbaum angenommen, der einen Dummy-Wurzelknoten besitzt. Ausgehend von diesem können einzelne Teilprobleme, die im Folgenden als *Dependenzknoten-Teilprobleme* bezeichnet werden, über die einzelnen Dependenzknoten definiert werden:

Definition 29 (Dependenzknoten-Teilproblem)**.** *Ein Dependenzknoten-Teil-*

problem zu einem gegebenen Nicht-Dummy-Dependenzknoten c enthält:

(i) das Wort, das c definiert

(ii) und alle direkt untergeordneten Wörter von c.

Ein Teilbaum, der von einem Dependenzknoten-Teilproblem abgedeckt wird, hat die maximale Tiefe 1. Ein Wort innerhalb eines Dependenzknoten-Teilproblems kann als übergeordnetes oder untergeordnetes Wort fungieren. Im Folgenden wird das übergeordnete als *Governor* und das untergeordnete als *Modifier* bezeichnet. Über die Unterordnungsbeziehungen kann über mehrere Teilprobleme hinweg eine Zerlegung abgeleitet werden. Da die Positionen des übergeordneten Wortes durch die Syntaxstruktur bereits gegeben sind, sind bei der Problemlösung ausschließlich die syntaktischen Label relevant.

In Abbildung 4.8 ist ein Beispiel für eine vollständige Zerlegung über Dependenzknoten-Teilprobleme dargestellt. Diese bezieht sich auf den Satz „In

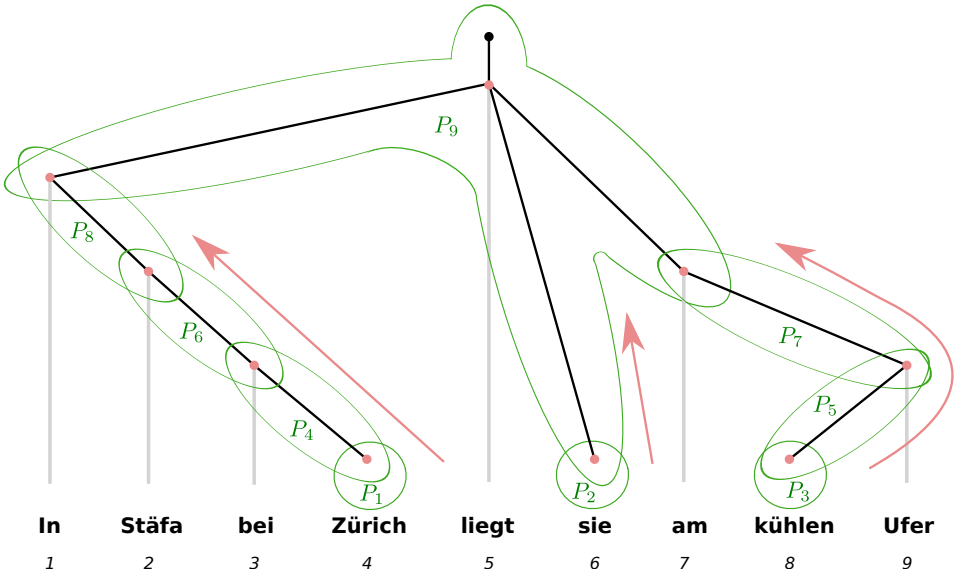


Abbildung 4.8: Baumartige Problemzerlegung anhand eines ungelabelten Dependenzbaums.

Stäfa bei Zürich liegt sie am kühlen Ufer“. Das Parsing-Problem ist in acht Teilprobleme $P_1, P_2, \ldots, P_8$ zerlegt. Die einzelnen Variablen werden über die rot eingefärbten Dependenzknoten repräsentiert. Da der Dummy-Wurzelknoten durch keine Variable repräsentiert wird, ist dieser schwarz eingefärbt. Des

Weiteren sind die Skopen der einzelnen Teilprobleme grün umrahmt und die blauen Pfeile zeigen eine Bottom-Up-Lösungsabfolge an. Die Tiefe einer Zerlegung ist durch die Satzlänge gebunden, da sie sich an den Dependenzknoten orientiert.

Wenn die Regeln der Grammatik ausschließlich über die direkte Dominanz und über die Sequenzialität der Wörter, die in der direkten Dominanz involviert sind, formuliert werden, deckt sich der Beschränkungsbereich mit dem Bereich eines Dependenzknoten-Teilproblems. Die Beschränkungen sind somit lokal und erlauben eine baumartige Aufspaltung über Dependenzknoten-Teilprobleme. In Abbildung 4.9 ist dieser Beschränkungsbereich schematisch dargestellt. Die Wörter werden mit den Kleinbuchstaben $w_1, w_2, \ldots$ und

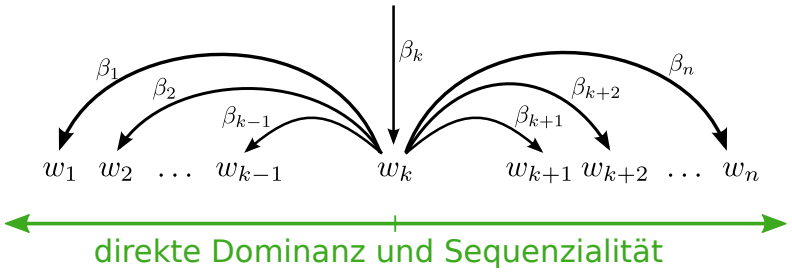


Abbildung 4.9: Beschränkungsbereich.

die Dependenz-Label mit den griechischen Buchstaben $\beta_1, \beta_2, \ldots$ bezeichnet. w_k steht für den Wurzelknoten eines Teilbaums, der über das Label β_k einem anderen Wort untergeordnet ist. Bei der Formulierung einer Regel kann auf die Wörter, auf deren Abfolge und auf den Typ der Unterordnungsbeziehungen zugegriffen werden. Hierbei können im Gegensatz zur WCDG gleichzeitig beliebig viele Kanten angefasst werden.

Der Bereich der Beschränkungen hat Ähnlichkeit mit den Wohlgeformtheitsbedingungen in Duchier (1999). Auch dort wird der Beschränkungsbereich ausgehend von einem Dependenzknoten definiert und es wird direkte Dominanz einbezogen und es wird die Einbeziehung von Sequenzialität vorgeschlagen, die jedoch nur angedeutet und letztendlich in dem XDG-Framework über eine zusätzliche Ebene umgesetzt ist (siehe Abschnitt 4.1.2).

Diskontinuierliche Strukturen

Über den Beschränkungsbereich, wie er in Abbildung 4.9 gegeben ist, ist es nicht möglich, diskontinuierliche Strukturen einzuschränken. Es kann aber

angenommen werden, dass die Wörter, die von einem Wort direkt oder indirekt dominiert werden, immer kontinuierlich sind. Dies wird beispielsweise in dem Beitrag Didakowski (2010, Abschnitt 5.3.1) vorgeschlagen. Mit dieser Einschränkung wären aber nur projektive Dependenzbäume erlaubt und es könnten diskontinuierliche Phänomene wie die freie Wortstellung im Mittelfeld oder die Topikalisierung im Deutschen nicht adäquat behandelt werden.

Innerhalb der XDG können kreuzende Kanten innerhalb der Ebene der unmittelbaren syntaktischen Dominanz über die Ebene der linearen Präzedenz eingeschränkt werden. Aufseiten der WCDG ist es hingegen möglich, zwei Kanten im Dependenzbaum anzufassen, und Bedingungen über diese zu formulieren, die deren Überkreuzung betreffen. Es wäre aber auch denkbar analog zur XDG innerhalb der WCDG eine zusätzliche Ebene für die lineare Präzedenz einzuführen. In beiden Frameworks können demnach diskontinuierliche Strukturen behandelt werden.

Um auch innerhalb des Problemzerlegungsansatzes Diskontinuität einzubeziehen, soll es möglich sein, kreuzende Kanten im Dependenzbaum anzufassen und einzuschränken. Hierzu wird das Dependenzknoten-Teilproblem aus Definition 29 um kreuzende Wörter erweitert:

Definition 30 (Dependenzknoten-Teilproblem – erweitert um kreuzende Wörter)**.** *Ein Dependenzknoten-Teilproblem zu einem gegebenen Nicht-Dummy-Dependenzknoten c enthält:*

(i) das Wort, das c definiert,

(ii) alle untergeordneten Wörter von c

(iii) und alle untergeordneten Wörter der Unterordnungen, die die direkten Unterordnungen von c kreuzen (kreuzende Wörter).

In Abbildung 4.10 ist ein Beispiel für eine vollständige Zerlegung auf Basis des erweiterten Dependenzknoten-Teilproblems dargestellt. Diese bezieht sich auf den Satz „weil die Frau mehr Käse will als Schokolade“. Das Parsing-Problem ist in acht Teilprobleme $P_1, P_2, \ldots, P_8$ zerlegt und ist aus einer nichtprojektiven syntaktischen Struktur abgeleitet.

Wenn in den Grammatikregeln zusätzlich über die kreuzenden Wörter Diskontinuität einbezogen wird, deckt sich der Beschränkungsbereich einer Regel mit dem Bereich eines Dependenzknoten-Teilproblems aus Definition

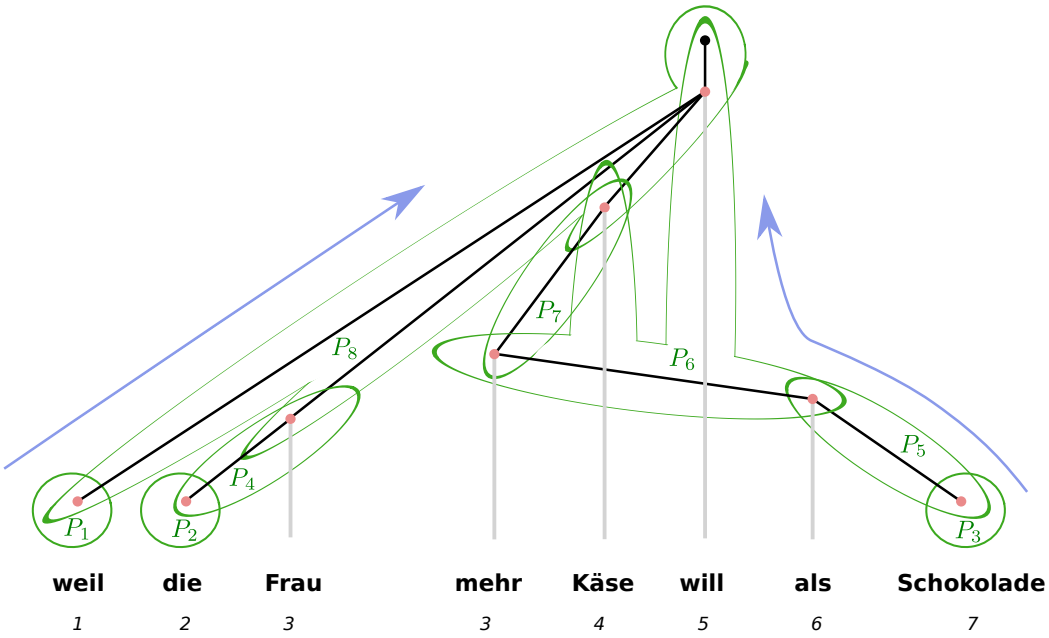


Abbildung 4.10: Problemzerlegung anhand eines nichtprojektiven ungelabelten Dependenzbaums.

30. Die Beschränkungen der Grammatik sind dann immer noch lokal. In Abbildung 4.11 ist die Erweiterung des Beschränkungsbereichs schematisch dargestellt. Bei der Formulierung der Beschränkungen kann zusätzlich auf die kreuzenden Wörter, angegeben durch die Kleinbuchstaben w_{1_1},w_{1_2},...,w_{2_1}, w_{2_2},..., und auf die Dependenz-Label zu den kreuzenden Wörtern, bezeichnet durch die griechischen Buchstaben β_{1_1},β_{1_2},...,β_{2_1},β_{2_2},..., Bezug genommen werden.

Jedoch kann eine baumartige Zerlegung anhand des Dependenzknoten-Teilproblems aus Definition 30 problematisch sein. Dies betrifft das Beispiel in Abbildung 4.10, dass sich auf den Nebensatz „weil die Frau mehr Käse will als Schokolade“ bezieht. Dort ist die Zerlegung nicht baumartig organisiert. Das Dependenzknoten-Teilproblem P_6 teilt sowohl Variablen mit P_7 als auch mit P_8. Die Dependenzknoten-Teilprobleme P_7 und P_8 liegen bei einer Bottom-Up-Betrachtung nach P_6. Aus diesem Grund ist es nicht möglich, eine perfekte Relaxierung umzusetzen. Dies ergibt sich aus der Baumartig-Bedingung aus Definition 28 in Abschnitt 2.3.3. Das grundlegende Problem liegt darin, dass sich P_7 und P_8 über die Variablen zu den Wörtern ‘mehr’ und

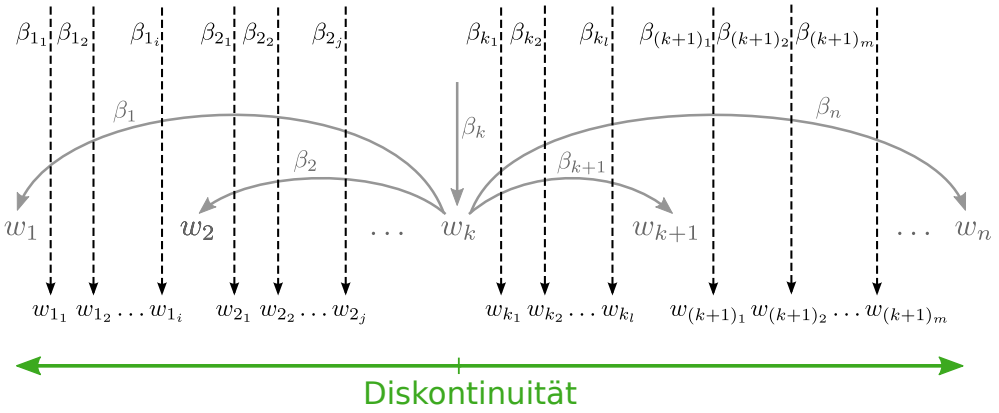


Abbildung 4.11: Erweiterung des Beschränkungsbereichs um Diskontinuität.

'will' indirekt über P_6 bedingen können. Somit verhindert das Dependenzknoten-Teilproblem P_6 eine Baumzerlegung, die sich streng an der syntaktischen Struktur orientiert.

4.3.3 Zerlegung in Ebenen

In Didakowski (2010, Abschnitt 5.3.4) wird vorgeschlagen, Teilprobleme, die wegen Diskontinuität miteinander verwoben sind, zusammenzufassen. So müssten in Abbildung 4.10 aus Abschnitt 4.3.2 die Teilprobleme P_6, P_7 und P_8 zu einem Teilproblem zusammengefasst werden. Hierdurch wäre eine Zerlegung, die sich streng an der syntaktischen Struktur orientiert, teilweise möglich. Jedoch kann das Zusammenfassen der verwobenen Teilprobleme zu einer Baumzerlegung führen, die nur wenige Teilprobleme besitzt. Prinzipiell ist eine baumartige Zerlegung unter Einbeziehung kreuzender Wörter also möglich. Eine Aufspaltung richtet sich dann aber nicht mehr streng nach der syntaktischen Struktur.

Um Diskontinuität zu behandeln, wird hier jedoch ein anderer Ansatz verfolgt. Es wird einer Problemzerlegung nachgegangen, die keine Verzweigungen besitzt. In Abbildung 4.12 ist ein Beispiel für solch eine Zerlegung gegeben, die sich auf den Beispielsatz „weil die Frau mehr Käse will als Schokolade“ aus Abschnitt 4.3.2 bezieht. Hier ist das Problem in fünf Teilprobleme aufgespalten, die im Folgenden als Schicht-Teilprobleme bezeichnet werden. Bei dieser einfachen Zerlegung ist sowohl eine Bottom-Up- als auch eine Top-Down-Lösungsabfolge möglich: Der blaue Pfeil zeigt die Bottom-Up-Lösungs-

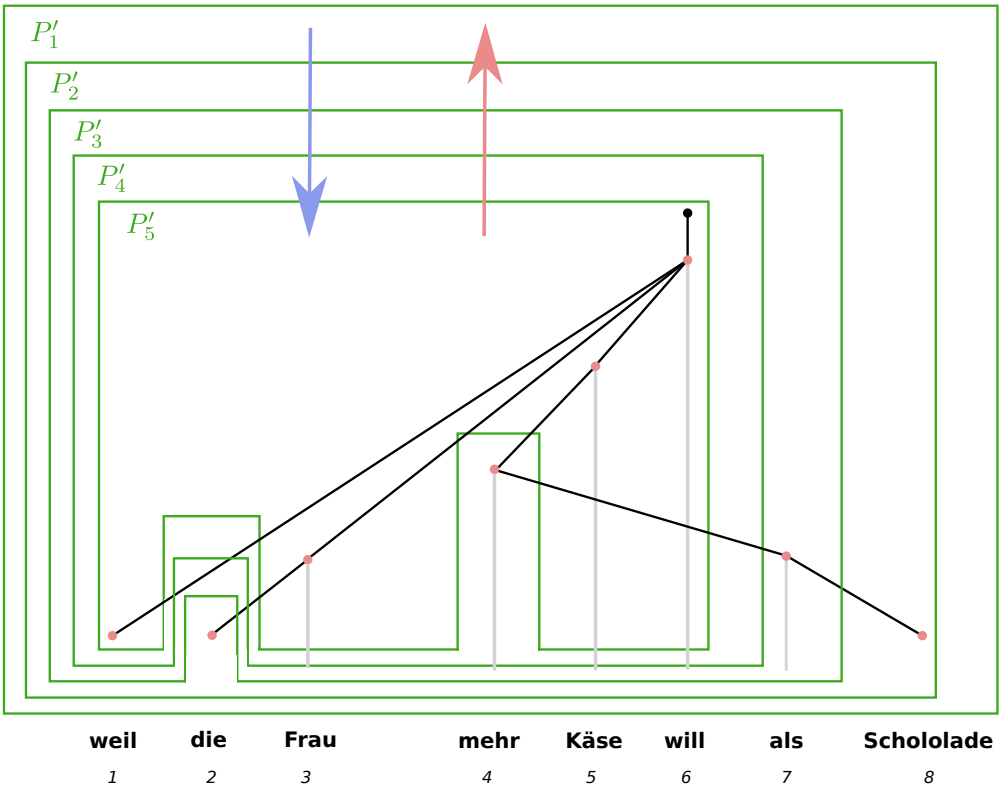


Abbildung 4.12: Nichtverzweigende Problemzerlegung anhand eines ungelabelten Dependenzbaums.

abfolge $P'_1 \prec P'_2 \prec \cdots \prec P'_5$ an und der rote die Top-Down-Lösungsabfolge $P'_5 \prec P'_4 \prec \cdots \prec P'_1$. Interessant ist hier, dass bei der Bottom-Up-Lösungsreihenfolge das initiale Schicht-Teilproblem bereits alle Variablen enthält. Der Skopus der darauf folgenden Schicht-Teilprobleme ist dann stets eine Teilmenge des Skopus des direkt vorhergehenden. Des Weiteren werden die Schicht-Teilprobleme während des Bottom-Up-Lösens ab der zweiten Schicht immer kleiner. Bei der Top-Down-Lösungsabfolge wachsen die Schicht-Teilprobleme hingegen kontinuierlich an, bis bei der vorletzten Schicht das Maximum erreicht ist. Zudem bildet der Skopus eines Schicht-Teilproblems stets eine Teilmenge des Skopus des direkt folgenden Schicht-Teilproblems, falls vorhanden.

Nun ist es möglich, die einzelnen Dependenzknoten-Teilprobleme aus Abbildung 4.10 in Abschnitt 4.3.2 den Teilproblemen aus Abbildung 4.12 zuzuordnen, ohne dass sie innerhalb eines Schicht-Teilproblems überlappen. Dies ist in Abbildung 4.13 verdeutlicht. Dort ist der Dependenzbaum über

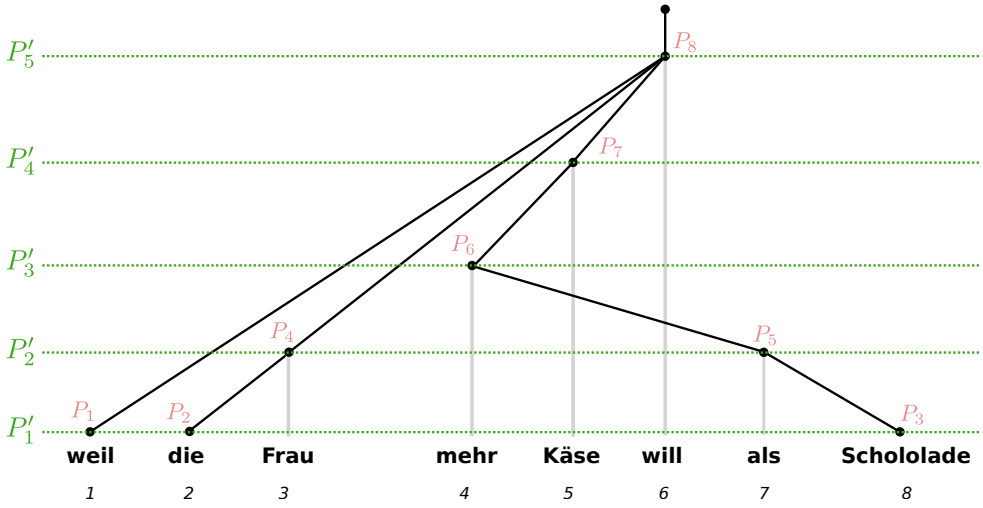


Abbildung 4.13: Zuordnung von Dependenzknoten-Teilproblemen zu Schicht-Teilproblemen anhand eines ungelabelten geschichteten Dependenzbaums.

die Dependenzknoten in mehrere Schichten aufgeteilt, wobei sich adjazente Knoten in verschiedenen Schichten befinden. Hierbei wird eine Schicht im Dependenzbaum einer Ebene in der Problemzerlegung zugeordnet. Die unterste Schicht entspricht der ersten Zerlegungsebene und die oberste Schicht der letzten. Analog hierzu wird innerhalb einer Zerlegung von der ersten, zweiten, dritten usw. Ebene oder dem ersten, zweiten, dritten usw. Schicht-Teilproblem gesprochen. Über die Schichten sind den Schicht-Teilproblemen $P'_1,P'_2,\ldots,P'_5$ die einzelnen Dependenzknoten-Teilprobleme $P_1,P_2,\ldots,P_8$ zugeordnet. So beinhaltet das Schicht-Teilproblem P'_1 die Dependenzknoten-Teilprobleme P_1,P_2,P_3, das Schicht-Teilproblem P'_2 die Dependenzknoten-Teilprobleme P_4,P_5, das Schicht-Teilproblem P'_3 das Dependenzknoten-Teilproblem P_6, das Schicht-Teilproblem P'_4 das Dependenzknoten-Teilproblem P_7 und das Schicht-Teilproblem P'_5 das Dependenzknoten-Teilproblem P_8.

Hier wäre auch eine andere Unterteilung in Schichten denkbar. Wenn beispielsweise eine Aufgliederung in acht anstatt in fünf Schichten bestünde, würden die Schichten zu den acht Schicht-Teilproblemen $P'_1,P'_2,\ldots,P'_8$ führen, die jeweils ein Dependenzknoten-Teilproblem aus $P_1,P_2,\ldots,P_8$ enthalten. Dies wird in Abbildung 4.14 verdeutlicht. Variablen, die innerhalb eines Schicht-Teilproblems in keinem Dependenzknoten-Teilproblem enthalten sind, werden im Folgenden als *Füller-Variablen* bezeichnet. Innerhalb eines Schicht-Teilproblems sind über diese Variablen keine Beschränkungen definiert.

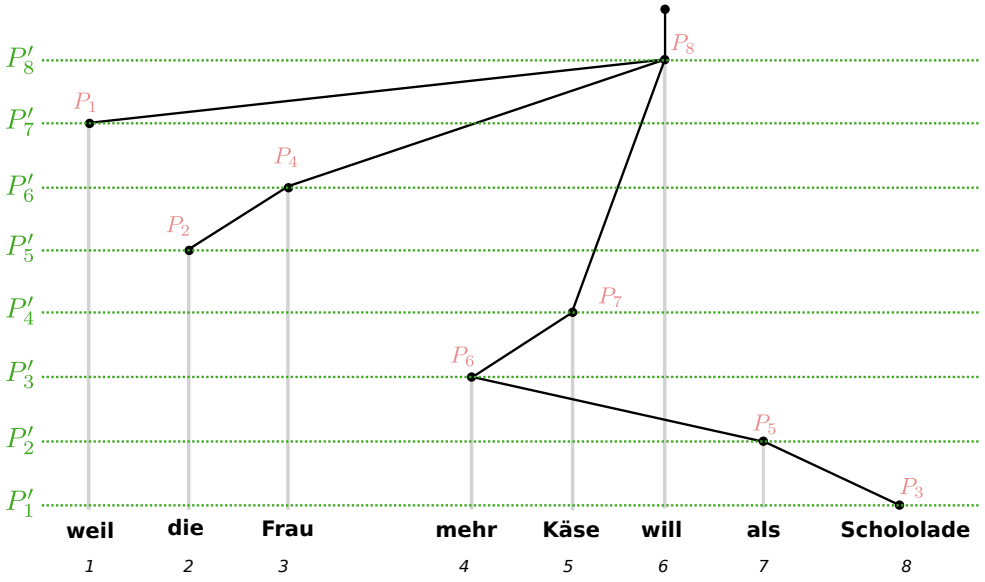


Abbildung 4.14: Zuordnung von Dependenzknoten-Teilproblemen zu Schicht-Teilproblemen anhand eines ungelabelten Dependenzbaums mit vielen Schichten.

Die Variablen, die vom ersten bis zum letzten Schicht-Teilproblem sukzessive aus dem Suchraum entfernt werden, ergeben sich über die Dependenzknoten-Teilprobleme. So sind innerhalb eines Schicht-Teilproblems stets die Variablen ausgeschlossen, die innerhalb eines Dependenzknoten-Teilproblems eines direkt vorhergehenden Schicht-Teilproblems Modifier repräsentieren. Demnach ist ein Schicht-Teilproblem wie folgt definiert:

Definition 31 (Schicht-Teilproblem)**.** *Ein Schicht-Teilproblem zu einer gegebenen Schicht l eines geschichteten Dependenzbaums enthält:*

(i) alle Dependenzknoten-Teilprobleme, die über die Dependenzknoten in l definiert sind, wobei sich die Dependenzknoten-Teilprobleme innerhalb des Schicht-Teilproblems nicht überlappen dürfen,

(ii) alle Variablen des Constraint-Problems, wenn kein vorhergehendes Schicht-Teilproblem existiert,

(iii) alle Variablen, die in einem direkt vorhergehenden Schicht-Teilproblem enthalten sind und die sich dort nicht in einem Dependenzknoten-Teilproblem befinden (Füller-Variablen),

(iv) alle Variablen, die innerhalb eines direkt vorhergehenden Schicht-Teilproblems in einem Dependenzknoten-Teilproblem enthalten sind und dort nicht als Modifier fungieren.

Somit kann über einen ungelabelten geschichteten Dependenzbaum eine Problemzerlegung in Schicht-Teilprobleme, wie sie in Abbildung 4.12 dargestellt wird, abgeleitet werden. Jedoch ist dies nicht für jeden Dependenzbaum möglich, da sich innerhalb eines Schicht-Teilproblems die Dependenzknoten-Teilprobleme nicht überlappen dürfen. Dies kann auftreten, wenn kreuzende Kanten involviert sind. In Abbildung 4.15 ist ein sehr konstruiertes Beispiel für diesen Fall gegeben. In 4.15 a und b ist eine Problemzerlegung für die un-

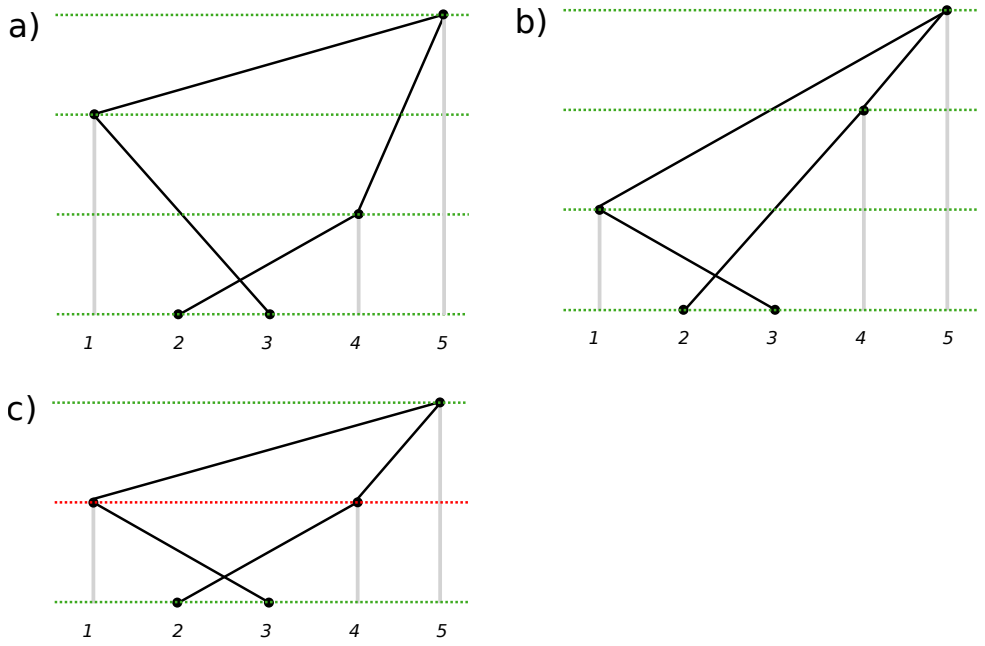


Abbildung 4.15: Problematische Dependenzbaumschichtungen für die Zuordnung von Dependenzknoten-Teilproblemen zu Schicht-Teilproblemen.

gelabelten geschichteten Dependenzbäume möglich, in 4.15 c hingegen nicht, da sich dort die Dependenzknoten-Teilprobleme innerhalb des Schicht-Teilproblems überschneiden (rot markierte Schicht).

Nun soll der Fall betrachtet werden, bei dem einem Dummy-Wurzelknoten mehrere Wörter direkt untergeordnet sind. Dies soll an dem elliptischen Satz „Zwei Frauen in St. Galen erschossen“ exemplifiziert werden, bei dem

die beiden Fragmente „Zwei Frauen“ und „in St. Galen erschossen“ dem Dummy-Wurzelknoten direkt untergeordnet sind. Für diese beiden werden zwei Baumzerlegungen angenommen, die auf Dependenzknoten-Teilproblemen beruhen. Beide Aufspaltungen sind in 4.16 abgebildet. Die Zerlegung für das

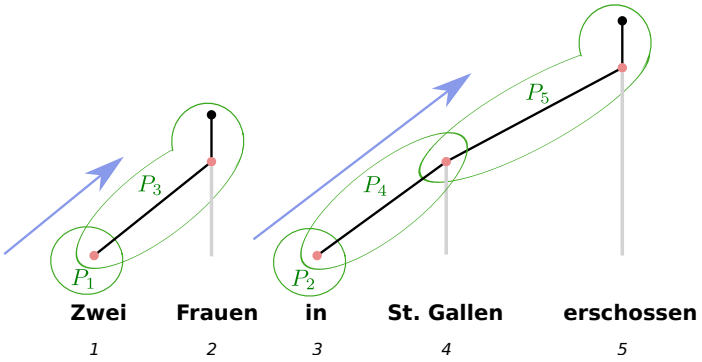


Abbildung 4.16: Zwei Baumzerlegungen für zwei Fragmente.

Fragment „in St. Galen erschossen“ mit den Dependenzknoten-Teilproblemen P_2, P_4, P_5 hat mehr Ebenen als die für das Fragment „Zwei Frauen“ mit den Dependenzknoten-Teilproblemen P_1, P_3. Trotzdem kann eine Aufspaltung auf Basis von Schicht-Teilproblemen durchgeführt werden. Dies ist in 4.17 dargestellt. Die Zuordnung von den Knoten- zu den Schicht-Teilproblemen ist

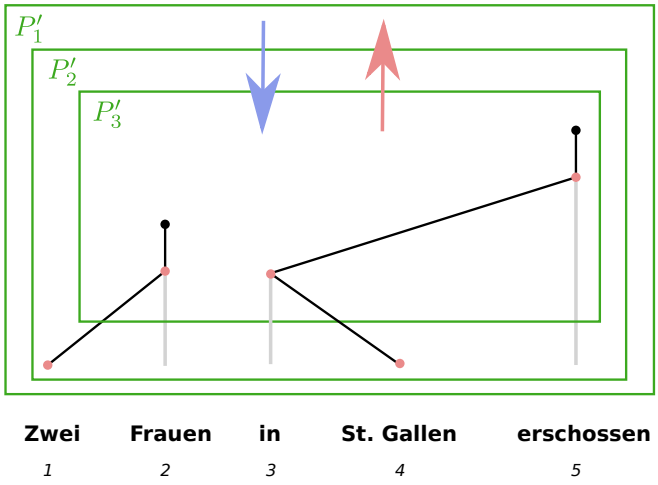


Abbildung 4.17: Nichtverzweigende Problemzerlegung bei Fragmenten.

ergänzend in Abbildung 4.18 aufgeführt. Die unterschiedliche Tiefe beider Fragmente wird ausgeglichen, indem innerhalb des Schicht-Teilproblems P'_3

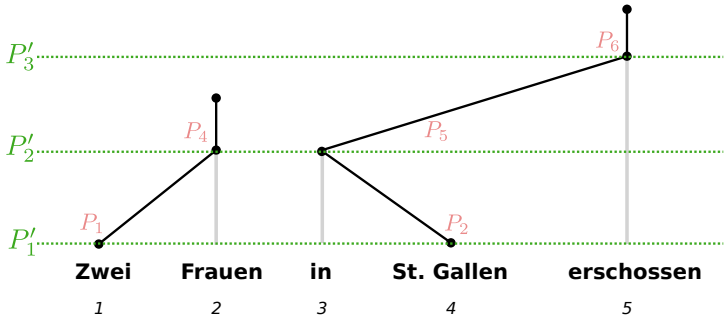


Abbildung 4.18: Zuordnung von Dependenzknoten="Teilproblemen zu Schicht-Teilproblemen bei Fragmenten.

die Variable, die mit dem Wort ‘Frauen’ korrespondiert, als Füller-Variable behandelt wird.

Gegeben ein Constraint-Problem $P = (C, con)$, dann entspricht die maximale Anzahl von Ebenen einer Zerlegung von P der Anzahl an Variablen in con. Wenn $|con| = 1$ ist, existiert genau ein Schicht-Teilproblem, das ein unäres Dependenzknoten-Teilproblem enthält und alle Variablen umfasst. Wenn $|con| > 1$ ist, beinhaltet das erste und zweite Schicht-Teilproblem alle Variablen des Constraint-Problems. Aus den folgenden Schicht-Teilproblemen wird sukzessive mindestens eine Variable entfernt. So können über $|con|$ Schicht-Teilprobleme im schlechtesten Fall $|con| - 2$ Variablen entfernt werden. Wenn $|con|$ Schicht-Teilprobleme existieren umfasst das letzte Schicht-Teilproblem genau zwei Variablen und es kann somit kein Schicht-Teilproblem mehr folgen, das ein nichtunäres Dependenzknoten-Teilproblem enthält. So besitzt eine Zerlegung im schlechtesten Fall eine Tiefe von $|con|$. Dies entspricht ohne Berücksichtigung des Dummy-Wurzelknotens der maximalen Höhe eines Dependenzbaums für einen Satz.

Zerlegungsverlängerung

Um die Tiefe einer Zerlegung zu erhöhen, ist es möglich, diese um Ebenen zu erweitern. Auf diese Weise können zwei unabhängige Zerlegungen auf die gleiche Tiefe gebracht werden. Dies wird in Abschnitt 4.5 im Kontext *simultaner perfekter Relaxierung* relevant. Um eine Verlängerung umzusetzen, werden *Dummy-Schichtteilprobleme* angenommen, die ausschließlich Füller-Variablen und somit keine Dependenzknoten-Teilprobleme enthalten. Ein Dummy-Schichtteilproblem ist wie folgt definiert:

Definition 32 (Dummy-Schichtteilproblem)**.** *Ein Dummy-Schichtteilproblem kann vor oder nach einem Schicht-Teilproblem angenommen werden, enthält kein Dependenzknoten-Teilproblem und umfasst:*

(ii) alle Variablen, die im direkt folgenden Schicht-Teilproblem involviert sind, falls ein solches existiert,

(i) alle Variablen, die dem Dummy-Wurzelknoten direkt untergeordnet sind, wenn kein Schicht-Teilproblem folgt.

Es soll ein Beispiel für eine Zerlegungsverlängerung über Dummy-Schichtteilprobleme gegeben werden. Hierfür wird an das Beispiel mit dem elliptischen Satz „Zwei Frauen in St. Galen erschossen“ angeknüpft, bei dem die Schicht-Teilprobleme P'_1, P'_2, P'_3 angenommen werden. Zusätzlich werden hier die vier Dummy-Schichtteilprobleme D_1, D_2, D_3, D_4 hinzugezogen, die die Zerlegungstiefe von 3 auf 7 erweitern. Eine entsprechende Zerlegung ist in Abbildung 4.19 gegeben, bei der die lilafarbenen Umrandungen die Skopen der Dummy-Schichtteilprobleme anzeigen. Vor und nach P'_1, nach P'_2 und nach P'_3 befinden sich Dummy-Schichtteilprobleme, wobei das finale Dummy-Schichtteilproblem D_4 die Variablen zu den Wörtern ‘Frauen’ und ‘erschossen’ und somit die Variablen, die dem Dummy-Wurzelknoten direkt untergeordnet sind, umfasst. Da mehrere Dummy-Schichtteilprobleme vor oder nach einem Schicht-Teilproblem möglich sind, könnte hier auch eine andere Aufteilung angenommen werden, um die Tiefe der Zerlegung von drei auf sieben zu erweitern.

Optimalzerlegung

Es kann mehrere Möglichkeiten geben, einen Dependenzbaum zu schichten und es können beliebig viele Dummy-Schichtteilprobleme an verschiedenen Stellen einer Zerlegung angenommen werden. Es soll daher eine Optimalzerlegung definiert werden, die eine zu präferierende Aufspaltung darstellt und wie folgt definiert ist:

Definition 33 (Optimalzerlegung)**.** *Bei einer Optimalzerlegung*

(i) umfasst ein nichtunäres Dependenzknoten-Teilproblem stets eine Variable, die sich in einem Dependenzknoten-Teilproblem des direkt vorhergehenden Schicht-Teilproblems befindet.

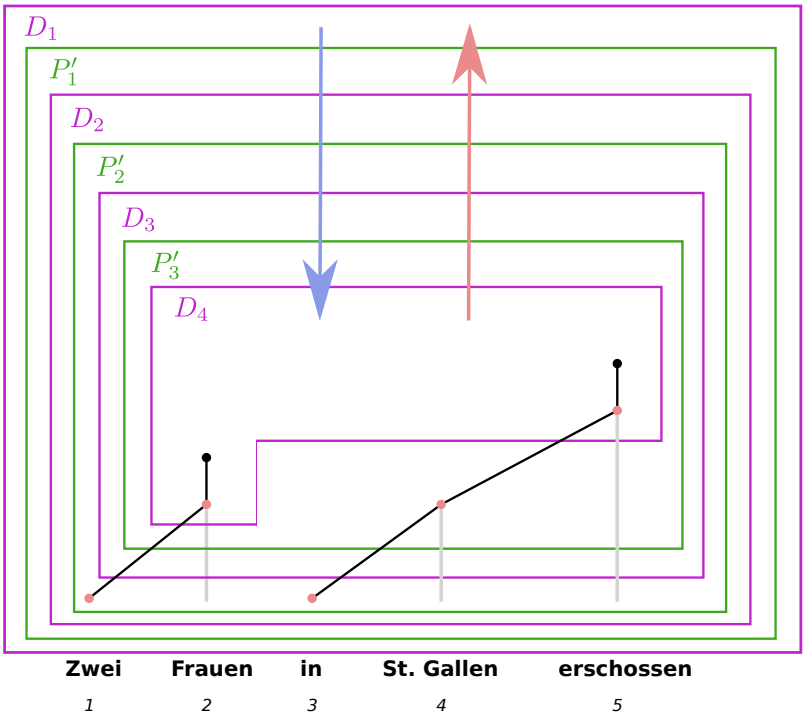


Abbildung 4.19: Problemzerlegungsverlängerung über Dummy-Schichtteilprobleme.

(ii) befinden sich alle unären Dependenzknoten-Teilprobleme in dem ersten Schicht-Teilproblem.

Eine Optimalzerlegung besitzt die Eigenschaft, stets weniger Ebenen zu besitzen als eine Nicht-Optimalzerlegung. Wenn beispielsweise ein nichtunäres Dependenzknoten-Teilproblem keine Variablen enthielte, die sich in einem Dependenzknoten-Teilproblem des direkt vorhergehenden Schicht-Teilproblems befände, würde stets eine kürzere oder gleich lange Problemzerlegung existieren, bei der dieses nichtunäre Dependenzknoten-Teilproblem in einer niedrigeren Zerlegungsebene involviert wäre. Wenn sich zudem ein unäres Dependenzknoten-Teilproblem nicht in dem initialen Schicht-Teilproblem befinden würde, würde stets eine kürzere oder gleich lange Problemzerlegung existieren, bei der dieses Dependenzknoten-Teilproblem in dem initialen Schicht-Teilproblem enthalten wäre. Um eine Optimalzerlegung zu verlängern, können Dummy-Schichtteilprobleme ausschließlich nach der letzten Ebene eingeführt werden.

Ein Beispiel für eine Optimalzerlegung ist die Problemzerlegung in Abbildung 4.13. In der Abbildung 4.14 ist hingegen eine Zerlegung zu sehen, die Unterpunkt (ii) in Definition 33 verletzt. Bei nichtprojektiven Dependenzbäumen können mehrere Optimalzerlegungen möglich sein. Ein Beispiel hierfür ist in Abbildung 4.15 a und b gegeben. Hier zeigt sich, dass bei einer Optimalzerlegung ohne Dummy-Schichtteilprobleme die Zuordnung eines Dependenzknotens zu einer i-ten Zerlegungsebene nicht unbedingt seiner Höhe plus eins entspricht.[6]

4.3.4 Perfekte Relaxierung

Für eine gegebene Problemzerlegung, die auf Schicht-Teilproblemen beruht, soll über eine perfekte Relaxierung eine Lösung berechnet werden. Es wird angenommen, dass die Dependenzknoten-Teilprobleme vorab gelöst sind. So ist mit einem Dependenzknoten-Teilproblem stets auch ein Constraint gemeint. Es kann entweder einer Bottom-Up- oder eine Top-Down-Lösungsabfolge nachgegangen werden. Aus jeder Zerlegungsebene wird eine lokale Konsistenzregel abgeleitet, dessen Beschaffenheit sich nach der Lösungsrichtung orientiert. Eine Problemzerlegung liefert hierfür drei wesentliche Informationen: die Skopen der Knoten- und Schicht-Teilprobleme, die Zuordnung von den Knoten- zu den Schicht-Teilproblemen und die Abfolge der Schicht-Teilprobleme.

Im Folgenden wird gezeigt, dass die Umsetzung einer Bottom-Up-Lösungsabfolge zu grundlegenden Problemen führt und dass eine Top-Down-Lösungsabfolge eine perfekte Relaxierung und somit die Berechnung der Lösung anhand einer Problemzerlegung ermöglicht.

Bottom-Up-Lösungsabfolge

Bei einer Bottom-Up-Lösungsreihenfolge wird aus jeder Zerlegungsebene eine lokale Konsistenzregel abgeleitet. Hierbei wird die initiale Regel der ersten Zerlegungsebene und die finale der letzten zugeordnet. Eine Regel umfasst: die Positionen der Dependenzknoten-Teilprobleme, die Position, die von der direkt vorhergehenden Regel generiert wird, falls diese existiert, und die Position, die die Regel selbst erzeugt, falls diese keiner der schon genannten

[6]Die Höhe eines Knotens entspricht der Länge des längsten Pfades ausgehend von diesem zu einem Terminalknoten.

Positionen entspricht. Demnach beinhaltet die finale Regel nicht alle Variablen des Constraint-Problems, wenn dieses mehr als drei Variablen enthält. Somit ist diese Lösungsreihenfolge nicht dafür geeignet, eine Belegung für alle Variablen des Constraint-Problems zu finden.

Es existiert aber noch ein weiteres grundlegendes Problem, das bei dieser Lösungsabfolge auftritt. Dies wird im Folgenden erläutert. Es wird eine zugrunde liegende Problemzerlegung angenommen, die keine Dummy-Schichtteilprobleme enthält. Die initiale Regel umfasst bei einer Bottom-Up-Lösungsabfolge bereits alle Variablen des Constraint-Problems. Dies hat direkte Konsequenzen für die Beschaffenheit der Folgeregeln. So muss die generierte Position einer Regel in der direkt folgenden enthalten sein, falls sie existiert. Des Weiteren müssen innerhalb einer vorhergehenden Regel gleich viel oder mehr Variablen involviert sein. Dies ergibt sich aus den drei Beschränkungen baumartig, abdeckend und bottom-up aus Definition 28 in Abschnitt 2.3.3. Zudem beinhaltet die generierte Position einer nicht finalen Regel genau die Variablen, die in der direkt folgenden involviert sind. Für die finale Regel gilt, dass alle Variablen, die in der generierten Position enthalten sind, auch in derselben Regel einbezogen sind. Jedoch ergibt sich hier ein grundsätzliches Problem: Wenn angenommen wird, dass dem Dummy-Wurzelknoten genau eine Variable untergeordnet ist, würde die finale lokale Konsistenzregel eine Position umfassen, die der Position des Knoten-Teilproblems und der des Schicht-Teilproblems entspräche. Diese Position würde aber auch mit der generierten Position der direkt vorhergehenden Regel übereinstimmen, falls diese existiert. Jedoch darf ein Constraint-Problem keine zwei Constraints enthalten, die den gleichen Typ besitzen. Dieses Problem kann auch dann auftreten, wenn kreuzende Wörter in den Dependenzknoten-Teilproblemen involviert sind. Hier liegt die grundlegende Problematik darin, dass über eine Regel eine Position generiert werden kann, für die bereits ein Dependenzknoten-Teilproblem existiert. Somit ist eine Bottom-Up-Lösungsabfolge, die sich streng an einer gegebenen Problemzerlegung orientiert, über lokale Konsistenzregeln nicht umsetzbar.

Top-Down-Lösungsabfolge

Auch bei der Top-Down-Lösungsabfolge wird aus jeder Zerlegungsebene eine lokale Konsistenzregel abgeleitet. Hier wird jedoch die initiale Konsistenzregel der letzten Zerlegungsebene und die finale der ersten zugeordnet. Eine Regel

umfasst die Positionen der Dependenzknoten-Teilprobleme, die Position, die von der direkt vorhergehenden Regel generiert wird, falls diese existiert, und die Position, die die Regel selbst erzeugt, falls diese keiner der schon genannten Positionen entspricht. Demnach beinhaltet die finale Regel alle Variablen des Constraint-Problems. Es wird somit über eine perfekte Relaxierung die Lösung für alle Variablen eines Constraint-Problems berechnet. Im Folgenden wird zunächst eine zugrunde liegende Problemzerlegung angenommen, die keine Dummy-Schichtteilprobleme enthält. Danach werden die Dummy-Schichtteilprobleme hinzugezogen.

Da bei der Top-Down-Lösungsabfolge das finale Schicht-Teilproblem alle Variablen umfasst, muss die erzeugte Position in der direkt folgenden Regel einbezogen sein. Hierbei gilt, dass alle Variablen, die in einer Regel involviert sind, auch in der generierten Position enthalten sind. Hieraus resultiert, dass die Variablen, die in einer Regel miteinbezogen sind, stets eine Teilmenge der Variablen sind, die in einer direkt folgenden Regel einbezogen sind. Es werden zudem ausschließlich Positionen generiert, die mehr als eine Variable umfassen, es sei denn, das SCSP enthält nur eine Variable. Dann wird von der initialen und einzigen Regel eine unäre Position erzeugt. Das grundlegende Problem der Bottom-Up-Lösungsabfolge kann hier nicht auftreten, da ein nichtunäres Dependenzknoten-Teilproblem beim Top-Down-Lösen stets eine 'neue' Variable einbezieht und eine Variable, die das übergeordnete Wort innerhalb eines nichtunären Dependenzknoten-Teilproblems repräsentiert, niemals in einem unären Dependenzknoten-Teilproblem involviert sein kann. Daher ist es beispielsweise möglich, innerhalb eines Schicht-Teilproblems alle Dependenzknoten-Teilprobleme zu einem einzigen zusammenzufassen. Auch dann würde die Problematik wie bei der Bottom-Up-Lösungsabfolge nicht auftreten.

Es soll nun die Behandlung der Zerlegungsebenen mit Dummy-Schichtteilproblemen betrachtet werden. Eine lokale Konsistenzregel für solch eine Ebene enthält genau eine Position. Diese entspricht der des Dummy-Schichtteilproblems. Wenn diese mit p bezeichnet wird, hat eine entsprechende Regel die Form: $p \leftarrow \varnothing$. Aus Zerlegungsebenen mit Dummy-Schichtteilproblemen können demnach lokale Konsistenzregeln abgeleitet werden.

Es ist folglich möglich, mithilfe von lokalen Konsistenzregeln anhand einer gegebenen Problemzerlegung eine Top-Down-Lösungsabfolge umzusetzen und eine Belegung für alle Variablen eines Constraint-Problems zu ermitteln.

4.4 Perfekte Relaxierung mithilfe von FSMs

Wie in Abschnitt 4.3.4 beschrieben, lässt sich innerhalb einer perfekten Relaxierung ein Top-Down-Lösen über lokalen Konsistenzregeln realisieren. Hier wird erläutert, wie diese Regeln mithilfe von FSMs umgesetzt werden können und wie mit deren Hilfe eine Lösung berechnet werden kann. Des Weiteren wird gezeigt, wie eine Bottom-Up-Lösungsabfolge über das sukzessive Zusammenfassen von lokalen Konsistenzregeln möglich wird. Für die Lösungsberechnung wird ein entsprechender Algorithmus vorgestellt.

4.4.1 Repräsentation lokaler Konsistenzregeln

In diesem Abschnitt wird beschrieben, wie lokale Konsistenzregeln im Rahmen einer Top-Down-Lösungsabfolge mithilfe von FSMs umgesetzt werden können. Hierzu wird ein Constraint-Problem $P = (C, con)$ über dem Constraint-System $CS = (S, D, V)$ angenommen, über das eine Problemzerlegung auf Basis von Schicht-Teilproblemen definiert ist. Darüber hinaus wird angenommen, dass die Dependenzknoten-Teilprobleme vorab gelöst sind und dass mit einem Dependenzknoten-Teilproblem stets auch ein Constraint gemeint ist. Eine lokale Konsistenzregel enthält unter anderem die Position, die von der direkt vorhergehenden Regel generiert wird. Wenn aber keine vorhergehende Regel existiert, wird hier die Position eines Universal-Constraint angenommen, der die Variablen im Skopus hat, die dem Dummy-Wurzelknoten direkt untergeordnet sind. Dieser Universal-Constraint wird im Folgenden als *Top-Constraint* bezeichnet.

Es sei eine lokale Konsistenzregel $r_l^p = l \leftarrow L$ mit $R = \{l\} \cup L$ und $p \in R$ angenommen. Hierbei bezeichnet p die Position, die von der direkt vorhergehenden Regel generiert wird, oder die Position des Top-Constraints. Zudem sei die Menge T mit $T \subseteq R$ gegeben, die alle Positionen der enthaltenen Dependenzknoten-Teilprobleme umfasst. Des Weiteren wird angenommen, dass über die lokale Konsistenzregel kein Dummy-Schichtteilproblem behandelt wird. Diese werden erst in Abschnitt 4.5 im Kontext simultaner lokaler Konsistenzregeln relevant. Somit wird angenommen, dass $T \neq \varnothing$. Dann können die Constraints (def_1, t_1),(def_2, t_2),...,(def_n, t_n) zu den einzelnen Positionen t_1,t_2,...,$t_n \in T$ mithilfe von FSMs repräsentiert werden: $(L(def_1), L(t_1))$, $(L(def_2), L(t_2))$,...,$(L(def_n), L(t_n))$. Wenn die Position $t = t_1 \cup t_2 \cup \cdots \cup t_n$ angenommen wird, dann ist der Constraint, der aus der Kombination der ein-

zelnen Constraints resultiert, definiert durch:

$$(\mathcal{D}, L(t)) = (L(def_1), L(t_1)) \otimes (L(def_2), L(t_2)) \otimes \cdots \otimes (L(def_n), L(t_n)) \quad (4.3)$$

Die Positionen t_1,t_2,...,t_n überlappen nicht und lassen sich daher anhand der Variablenordnung über V sortieren. Gegeben $t_1, t_2 \in T$ mit $t_1 \neq t_2$. dann gilt $t_1 \prec t_2$ genau dann wenn ein $x \in t_1$ und ein $y \in t_2$ existiert, sodass $x \prec y$. Dann kann der Constraint-Wert $\mathcal{D}$ auch direkt über die Produktoperation über rationale Potenzreihen definiert werden:

$$\mathcal{D} = L(def_1)\ L(def_2)\ \ldots\ L(def_n) \quad (4.4)$$

In dieser Definition sind die einzelnen Constraint-Typen nicht einbezogen. Solch eine Definition der Constraint-Kombination wird später auch im Kontext der simultanen lokalen Konsistenzregeln in Abschnitt 4.5 relevant.

Innerhalb einer lokalen Konsistenzregel ergibt sich die Position l aus der Vereinigung der Positionen p und t. Angenommen, es existiert der Constraint $(\mathcal{P}, p)$ zu der Position p, dann ist nach der Kombination von $(\mathcal{P}, L(p))$ und $(\mathcal{D}, L(t))$ eine Projektion auf die Position l nicht mehr notwendig. Der Wert $\mathcal{P}$ wird aber erst von der direkt vorhergehenden Regel generiert, es sei denn, es handelt sich um den Top-Constraint. So kann nicht der Wert $\mathcal{P}$, aber die Position p für die Berechnung der Kombination herangezogen werden. Daher muss die Kombination auf eine andere Weise definiert werden als in Abschnitt 3.3.2. Wenn $(\mathcal{P}, L(p)) \otimes (\mathcal{D}, L(t)) = (\mathcal{L}, L(l))$, dann ist $L(l)$ analog zu dem Ausdruck 3.15 aus Abschnitt 3.3.2 definiert durch:

$$L(l) = L(p \cup t) = L(p) \sqcup L(t) \quad (4.5)$$

und der Constraint-Wert $\mathcal{L}$ durch:

$$\mathcal{L} = Range \begin{pmatrix} \mathcal{P} \\ \circ \\ L(\downarrow_p^l)^{-1} \\ \circ \\ Dom(L(\downarrow_d^l) \circ \mathcal{D}) \end{pmatrix} \quad (4.6)$$

Hier ist die Tupel-Projektion im Vergleich zu dem Ausdruck 3.16 aus Abschnitt 3.3.2 für den Constraint-Wert $\mathcal{P}$ invertiert $(L(\downarrow_p^l)^{-1})$. Dadurch wird

die Ermittlung des Definitionsbereichs zu einer Ermittlung des Wertebereichs. Zudem ist die Wertebereichermittlung nach außen gezogen. Hierdurch kann eine lokale Konsistenzregel r_l^p, die die Position p enthält und die Position l generiert, durch die folgende rationale Potenzreihentransduktion $L(r_l^p)$ umgesetzt werden, die einen Constraint-Wert zur Position p auf einen Constraint-Wert zur Position l abbildet:

$$L(r_l^p) = \begin{pmatrix} L(\downarrow_p^l)^{-1} \\ \circ \\ Dom(L(\downarrow_d^l) \circ \mathcal{D}) \end{pmatrix} \tag{4.7}$$

Nun kann über die lokale Konsistenzregel $L(r_l^p)$ die Lösung einer Zerlegungsebene berechnet werden:

$$(\mathcal{L}, L(l)) = (Range(\mathcal{P} \circ L(r_l^p)), L(l)) \tag{4.8}$$

Dies ist so zu interpretieren, dass einer Regel zum Lösen des Teilproblems noch ein Constraint fehlt. Ist dieser vorhanden, kann die Teilproblemlösung berechnet werden.

4.4.2 Anwendung lokaler Konsistenzregeln

Bei der sequenziellen Anwendung mehrerer lokaler Konsistenzregeln, die mithilfe von FSMs repräsentiert sind, ist die Ausgabe einer Regel stets die Eingabe der direkt folgenden. Für die Berechnung der Lösung werden ausschließlich Constraint-Werte herangezogen, da die Positionen in den Regeln bereits berücksichtigt sind. Die Anwendung einer lokalen Konsistenzregel $L(r_l^p)$ auf einen Constraint-Wert ist gleichbedeutend mit $[r_l^p](P)$, also dem Anwenden von r_l^p auf ein Constraint-Problem P: Über eine Regelanwendung wird dem Problem entweder implizit ein Constraint hinzugefügt, oder es wird implizit ein bestehender Constraint ersetzt.

Angenommen, es existiert ein Constraint-Problem $P = (C,con)$ und eine endliche Sequenz von lokalen Konsistenzregeln $S = r_{l_2}^{l_1};r_{l_3}^{l_2};\ldots;r_{l_k}^{l_{k-1}}$ mit $l_k = con$, die eine perfekte Relaxierung angibt. Darüber hinaus sei angenommen, dass diese Regeln über die rationalen Potenzreihentransduktionen $L(r_{l_2}^{l_1}),L(r_{l_3}^{l_2}),\ldots,L(r_{l_k}^{l_{k-1}})$ repräsentiert sind und dass analog für deren Sequenz $L(r_{l_2}^{l_1});L(r_{l_3}^{l_2});\ldots;L(r_{l_k}^{l_{k-1}})$ geschrieben wird. Zudem sei angenommen,

dass der Top-Constraints durch $(L(\bar{1}_{l_1}),L(l_1))$ gegeben ist. Dann wird über die Anwendung der Regelsequenz ein Constraint $(\mathcal{L},L(l_k))$ generiert, dessen Typ dem des Constraint-Problems entspricht und dessen Wert wie folgt definiert ist:

$$\mathcal{L} = Range(Range(\underbrace{\ldots}_{n} Range(L(\bar{1}_{l_1}) \circ L(r_{l_2}^{l_1})) \circ L(r_{l_3}^{l_2})) \circ \underbrace{\ldots}_{n} \circ L(r_{l_k}^{l_{k-1}})) \tag{4.9}$$

Der Constraint $(\mathcal{L},L(l_k))$ stellt die Lösung des Constraint-Problems dar. Die Anwendung der Regelsequenz ist gleichbedeutend mit $[r_{l_2}^{l_1};r_{l_3}^{l_2};\ldots;r_{l_k}^{l_{k-1}}](P)$: Es werden innerhalb des Constraint-Problems bis zum Lösungs-Constraint sukzessive Constraints hinzugefügt oder ersetzt. Der Top-Constraint beeinflusst hierbei die Lösung des Constraint-Problems nicht und könnte bei der Berechnung auch weggelassen werden.

4.4.3 Zusammenfassen lokaler Konsistenzregeln

Anstatt die einzelnen Konsistenzregeln iterativ anzuwenden, können diese auch zusammen mit dem Top-Constraint zu einer lokalen Konsistenzregel zusammengefasst werden. Dann wird über das Zusammenfassen ein Constraint $(\mathcal{L},L(l_k))$ generiert, dessen Typ dem des Constraint-Problems entspricht und dessen Wert wie folgt definiert ist:

$$\mathcal{L} = Range(L(\bar{1}_{l_1}) \circ L(r_{l_2}^{l_1}) \circ L(r_{l_3}^{l_2}) \circ \cdots \circ L(r_{l_k}^{l_{k-1}})) \tag{4.10}$$

Hier wird deutlich, dass über das Zusammenfassen von lokalen Konsistenzregeln sowohl ein Top-Down- als auch ein Bottom-Up-Lösen möglich ist. Dies ergibt sich aus der Assoziativität der Kompositionsoperation.

4.4.4 Bottom-Up-Lösen

Für eine explizites Bottom-Up-Lösen ist der Algorithmus 1 gegeben, der für ein gegebenes Constraint-Problem den Lösungs-Constraint über das sukzessive Zusammenfassen von lokalen Konsistenzregeln berechnet. Innerhalb der For-Schleife in den Zeilen 2 bis 4 werden die lokalen Konsistenzregeln nach und nach Bottom-Up zusammengefasst. Die Lösungsabfolge unterscheidet sich demnach von der tatsächlichen Abfolge der Regelsequenz. Anhand des

Algorithmus 1 Perfekte Relaxierung mit Bottom-Up-Lösungsabfolge.

Eingabe: Eine endliche Sequenz von lokalen Konsistenzregeln $S = A(r_{l_2}^{l_1}); A(r_{l_3}^{l_2}); \ldots; A(r_{l_k}^{l_{k-1}})$, die eine perfekte Relaxierung für ein Constraint-Problem $P = (C, con)$ mit $l_k = con$ angibt. Ein Constraint $(A(\bar{1}_{l_1}), A(l_1))$, der den Top-Constraint repräsentiert.

Ausgabe: Der Lösungs-Constraint von P.

1: $\mathcal{L} \leftarrow A(r_{l_k}^{l_{k-1}})$
2: **for** $i \leftarrow k-1$ **downto** 2 **do**
3: $\quad \mathcal{L} \leftarrow A(r_{l_i}^{l_{i-1}}) \circ \mathcal{L}$
4: **end for**
$\mathcal{L} \leftarrow A(\bar{1}_{l_1}) \circ \mathcal{L}$
5: **return** $(Range(\mathcal{L}), A(con))$

Lösungs-Constraints kann anschließend, wie in Abschnitt 3.3.5 beschrieben, die beste Konsistenzstufe oder die abstrakte Lösung berechnet werden.

4.5 Simultane perfekte Relaxierung mithilfe von FSMs

In diesem Abschnitt wird davon ausgegangen, dass keine konkrete Problemzerlegung im Voraus gegeben ist. Dahingegen wird versucht, über simultane lokale Konsistenzregeln gleichzeitig allen möglichen Problemzerlegungen zu folgen. Wenn eine Problemzerlegung Inkonsistenzen aufweist, wird dieser nicht mehr nachgegangen. Es bleiben letztlich die Zerlegungen übrig, die konsistent sind. So werden alle möglichen Problemzerlegungen für ein Constraint-Problem ermittelt und es werden gleichzeitig die enthaltenen Teilprobleme gelöst. Es wird im Rahmen der perfekten Relaxierung einer Top-Down-Lösungsabfolge nachgegangen. Ein Bottom-Up-Lösen ergibt sich dann analog zu Abschnitt 4.4.4 aus dem Zusammenfassen von simultanen lokalen Konsistenzregeln und der Assoziativität der Kompositionsoperation. Es werden wesentliche Aspekte einbezogen: die Behandlung nichtprojektiver Strukturen, die verzögerte Anwendung von Constraints und die kompakte Repräsentation von Zwischenergebnissen während des Problemlösens.

4.5.1 Repräsentation simultaner lokaler Konsistenzregeln

Bei der Definition einer simultanen lokalen Konsistenzregel ist es nicht möglich, auf konkrete Constraint-Positionen zuzugreifen, da initial weder eine Problemzerlegung noch explizite Typen für die quantifizierten regulären Constraints gegeben sind. So muss auf relative Positionsinformationen zurückgegriffen werden, die beispielsweise darüber Auskunft geben, ob eine Variable innerhalb eines Dependenzknoten-Teilproblems den Governor, den Modifier oder ein kreuzendes Wort repräsentiert. Das Alphabet für die verknüpfte Information ist durch Γ gegeben und umfasst die Zeichen Ⓖ, Ⓜ, 🅖, 🅜, 🅘, ⓖ, ⓜ und ■. Die Bedeutungen der einzelnen Zeichen sind in Tabelle 4.1 unter der Perspektive einer Top-Down-Lösungsabfolge zusammengefasst. Hiervon wird das Zeichen 🅘 erst in Abschnitt 4.5.10 im Kontext der affixorientierten Zerlegung relevant. Allgemein können lexikalische oder morphologische Informationen wie die Wortart oder Lemmaform mit einzelnen Variablen verknüpft sein. Diese werden im Folgenden jedoch zur Vereinfachung ausgelassen, da sie für die Darlegung des Ansatzes nicht relevant sind. Über die *relativen Positionsinformationen* lassen sich Variablen definieren, die bestimmte Positionen innerhalb der Problemzerlegung innehaben. Die entsprechenden rationalen Sprachen sind in Tabelle 4.2 aufgeführt.

Die relative Position eines Constraints ergibt sich stets aus der eines anderen Constraints. Es kann beispielsweise ausgedrückt werden, dass ein Constraint alle Variablen eines anderen Constraints im Skopus hat außer denjenigen, die dort eine bestimmte syntaktische Position besitzen. So wird im Folgenden auch von der relativen Position eines Constraint-Wertes und von der Kombination von Constraint-Werten gesprochen, die mit der Constraint-Kombination korrespondiert. Im weiteren Verlauf wird bei der Umsetzung einer simultanen lokalen Konsistenzregel analog zu der Beschreibung der relativen Positionsinformationen eine Top-Down-Lösungsabfolge betrachtet.

Es wird angenommen, dass die Beschränkungen der Grammatik über reguläre quantifizierte Constraints umgesetzt sind und dass jedes Dependenzknoten-Teilproblem genau einen regulären quantifizierten Constraint umfasst. Mit einem Dependenzknoten-Teilproblem ist somit auch stets ein regulärer quantifizierter Constraint gemeint. Zudem wird angenommen, dass der Constraint-Wert $\mathcal{Q}$ existiert, über den alle regulären quantifizierten Con-

Zeichen	relative Position
Ⓖ	Governor eines Dependenzknoten-Teilproblems
Ⓜ	Modifier eines Dependenzknoten-Teilproblems
ⓖ	kreuzendes Wort, das innerhalb einer vorhergehenden Zerlegungsebene Governor war
ⓜ	kreuzendes Wort, das innerhalb einer vorhergehenden Zerlegungsebene Modifier war
🅖	Füller-Variable, die innerhalb einer vorhergehenden Zerlegungsebene Governor war
🅜	Füller-Variable, die innerhalb einer vorhergehenden Zerlegungsebene Modifier war
🅘	Füller-Variable, die innerhalb einer folgenden Zerlegungsebene in keinem Knoten-Teilproblem enthalten ist
■	Eine Variable, die sich in der aktuellen Zerlegungsebene in keinem Teilproblem befindet

Tabelle 4.1: Relative Positionsinformationen aus Top-Down-Sicht.

straints definiert sind und der alle Beschränkungen der Grammatik umfasst. Der Support der rationalen Potenzreihe, die diesen Constraint-Wert repräsentiert, ist eine Teilmenge der folgenden rationalen Sprache, die den Beschränkungsbereich aus Abbildung 4.11 aus Abschnitt 4.3 widerspiegelt:

$$Supp(\mathcal{Q}) \subseteq \ (\mathcal{M}^+ \ \mathcal{X}^*)^* \ \mathcal{G} \ (\mathcal{X}^* \ \mathcal{M}^+)^* \tag{4.11}$$

Wenn die Positionen der übergeordneten Wörter in den Variablenwerten einbezogen werden, wäre indirekt über die regulären quantifizierten Constraints ein Maximum für die Satzlänge gesetzt (vgl. Abschnitt 4.2). Daher wird hier angenommen, dass die Variablenwerte ausschließlich Dependenz-Label umfassen und dass eine explizite Dependenzstruktur später aus der Problemzerlegung abgeleitet werden kann (siehe Abschnitt 4.5.13).

rationale Sprache	Position	definiert durch
$\mathcal{G}$	Govenor	$\triangleleft$ Ⓖ $\diamond L(D) \triangleright$
$\mathcal{M}$	Modifier	$\triangleleft$ Ⓜ $\diamond L(D) \triangleright$
$\mathcal{X}$	kreuzendes Wort	$\triangleleft$ (Ⓜ $\cup$ Ⓖ) $\diamond L(D) \triangleright$
$\mathcal{I}$	Füller-Variable	$\triangleleft (\epsilon \cup$ ❶ $\cup$ ❻ $\cup$ Ⓜ) $\diamond L(D) \triangleright$
$\mathcal{U}$	in keinem Teilproblem	$\triangleleft$ ■ $\diamond L(D) \triangleright$

Tabelle 4.2: Variablen mit bestimmter Position innerhalb der Problemzerlegung.

Wie bei den lokalen Konsistenzregeln in Abschnitt 4.4.2 wird für die erste simultane lokale Konsistenzregel ein Top-Constraint herangezogen. Bei diesem muss die relative Positionsinformation berücksichtigt werden. So ist der Wert des Top-Constraints definiert durch:

$$\mathcal{T} = \epsilon \leftarrow \text{Ⓜ} \circ \mathcal{M}^+ \tag{4.12}$$

Dem Dummy-Wurzelknoten sind ausschließlich die Variablen untergeordnet, die Modifier repräsentieren ($\mathcal{M}^+$). Von diesen Variablen muss es mindestens eine geben. Darüber hinaus wird die Modifier-Informationen über Epsilon eingeführt ($\epsilon \leftarrow$ Ⓜ). Wenn ausschließlich Zerlegungen erwünscht sind, bei denen dem Dummy-Wurzelknoten genau eine Variable untergeordnet ist, kann der Wert des Top-Constraints auch durch $\epsilon \leftarrow$ Ⓜ $\circ \mathcal{M}$ definiert werden. Im Folgenden werden solche einschneidenden Einschränkungen jedoch nicht verfolgt.

Um die Lösung einer Zerlegungsebene berechnen zu können, müssen die Constraint-Werte der enthaltenen Dependenzknoten-Teilprobleme und der Constraint-Wert, der von der direkt vorhergehenden Regel generiert wird, miteinander kombiniert werden. Dies muss hierbei simultan für alle Zerlegungsmöglichkeiten einer Zerlegungsebene geschehen. So lässt sich im ersten Schritt über den Constraint-Wert $\mathcal{Q}$ eine rationale Potenzreihe definieren, die simultan alle möglichen Kombinationen von Dependenzknoten-Teilproblemen innerhalb einer Zerlegungsebene repräsentiert. Diese kann analog zu dem Ausdruck 4.4 aus Abschnitt 4.4.1 über eine Constraint-Kombination spezifiziert werden, die keine Constraint-Typen einbezieht und somit nur die Constraint-Werte betrifft. Hier wird zusätzlich angenommen, dass die Anzahl

an möglichen Dependenzknoten-Teilproblemen ungebunden ist. Daher ist die Kombination über die Plushülle und nicht wie in Abschnitt 4.4.1 über die Produktoperation realisiert. Über die Plushülle wird einerseits ausgedrückt, dass mindestens ein Dependenzknoten-Teilproblem enthalten ist und andererseits, dass die Dependenzknoten-Teilprobleme nicht überlappen. So ist die rationale Potenzreihe, die alle möglichen Kombinationen von Dependenzknoten-Teilproblemen innerhalb einer Zerlegungsebene repräsentiert, definiert durch:

$$\mathcal{Q}^+ \tag{4.13}$$

Die rationale Potenzreihe $\mathcal{Q}^+$ ist hierbei als Wert eines simultanen Constraints zu verstehen, der eine ungebundene Anzahl von Constraints simultan repräsentiert und zwar solche, die für eine beliebige Zerlegungsebene möglich sind.

Im zweiten Schritt muss die Kombination des Constraint-Werts $\mathcal{Q}^+$ mit dem Constraint-Wert berechnet werden, der von der direkt vorhergehenden Regel generiert wird. Hierzu muss der Constraint-Wert $\mathcal{Q}^+$ auf die Position, die von der Regel generiert wird, projiziert werden. Es müssen alle Positionen berücksichtigt werden, die für die Schicht-Teilprobleme einer Zerlegungsebene möglich sind. So ist das Ergebnis der Tupel-Projektion von $\mathcal{Q}^+$ auf die Position, die von der Regel generiert wird, definiert durch:

$$(\mathcal{I}^* \ \mathcal{Q})^+ \ \mathcal{I}^* \tag{4.14}$$

Hier wird über die Füller-Variablen die Position von $\mathcal{Q}^+$ auf alle möglichen Positionen der Schicht-Teilprobleme erweitert.

Nun müssen die relativen Positionsinformationen für die Kombination von $\mathcal{Q}^+$ mit dem Constraint-Wert, der von der direkt vorhergehenden Regel generiert wird, angepasst werden. Um dies zu realisieren, wird das zusätzliche Markierungszeichen $\spadesuit \in \Sigma$ verwendet. Es wird angenommen, dass über dieses Zeichen die einzelnen Knoten-Teilprobleme innerhalb von $(\mathcal{I}^* \ \mathcal{Q})^+ \ \mathcal{I}^*$ markiert sind:

$$(\mathcal{I}^* \ \spadesuit \ \mathcal{Q} \ \spadesuit \ )^+ \ \mathcal{I}^* \tag{4.15}$$

Im Folgenden werden die beiden Filter $\mathcal{F}_u$ (upper) und $\mathcal{F}_l$ (lower) definiert, über die die Problemzerlegung grundlegend gesteuert wird. Mit ihrer Hilfe wird beeinflusst, welche Positionen bzw. Constraints einbezogen und generiert werden. Sie sind aus Top-Down-Sicht definiert und beziehen sich auf die relativen Positionsinformationen und können über das Trennzeichen

♠ auf die Dependenzknoten-Teilprobleme zugreifen. Die Filter werden auf die rationale Potenzreihe $(\mathcal{I}^* \spadesuit \mathcal{Q} \spadesuit)^+ \mathcal{I}^*$ angewendet, um relative Positionsinformationen umzuschreiben oder zu entfernen:

$$\mathcal{F}_u \circ (\mathcal{I}^* \spadesuit \mathcal{Q} \spadesuit)^+ \mathcal{I}^* \circ \mathcal{F}_l \tag{4.16}$$

Hier ist anzumerken, dass die Markierungszeichen ♠ im späteren Verlauf nicht mehr vorhanden sind. Dies betrifft auch die relativen Positionsinformationen ⓖ, ⓜ, 🅖 und 🅜. Das heißt, die Zeichen werden ausschließlich innerhalb der Regelkompilierung verwendet.

Der erste Filter $\mathcal{F}_u$ behandelt die relativen Positionsinformationen in Bezug zur vorhergehenden lokalen Konsistenzregel:

$$\mathcal{F}_u = \left(\begin{array}{c} (\Sigma - \spadesuit)^* \\ \circ \\ \text{Ⓖ} \leftarrow \text{🅖} \\ \circ \\ \text{Ⓜ} \leftarrow \text{🅜} \end{array} \right) \epsilon : \spadesuit \left(\begin{array}{c} (\Sigma - \spadesuit)^* \\ \circ \\ \text{Ⓖ} \leftarrow \text{ⓖ} \\ \circ \\ \text{Ⓜ} \leftarrow \text{ⓜ} \\ \circ \\ \text{Ⓜ} \leftarrow \text{Ⓖ} \\ \circ \\ \blacksquare \leftarrow \text{Ⓜ} \end{array} \right) \epsilon : \spadesuit \;)^* \left(\begin{array}{c} (\Sigma - \spadesuit)^* \\ \circ \\ \text{Ⓖ} \leftarrow \text{🅖} \\ \circ \\ \text{Ⓜ} \leftarrow \text{🅜} \end{array} \right) \tag{4.17}$$

Modifier-Variablen sind in der direkt vorhergehenden Regel in keinem Teilproblem involviert ($\blacksquare \leftarrow$ Ⓜ). Governor-Variablen fungieren hingegen in einer vorhergehenden Regel als Modifier oder sie ist dem Dummy-Wurzelknoten untergeordnet (Ⓜ $\leftarrow$ Ⓖ). Variablen, die sich auf kreuzende Wörter beziehen, haben in einer vorhergehenden Regel die Funktion als Governor (Ⓖ $\leftarrow$ ⓖ) oder als Modifier (Ⓜ $\leftarrow$ ⓜ). Des Weiteren sind Füller-Variablen mit relativer Positionsinformation innerhalb einer vorhergehenden Regel Governor (Ⓖ $\leftarrow$ 🅖) oder Modifier (Ⓜ $\leftarrow$ 🅜). Zuletzt werden die Markierungszeichen ♠ über Epsilon eingeführt ($\epsilon : \spadesuit$), da diese nur innerhalb der Regel relevant sind.

Der zweite Filter $\mathcal{F}_l$ behandelt die relativen Positionsinformationen in

Bezug zu der darauffolgenden lokalen Konsistenzregel:

$$\mathcal{F}_l = \Big(\begin{pmatrix} (\Sigma - \spadesuit)^* \\ \circ \\ \textcircled{G} \mapsto \textcircled{G} \\ \circ \\ \textcircled{M} \mapsto \textcircled{M} \end{pmatrix} \spadesuit : \epsilon \begin{pmatrix} (\Sigma - \spadesuit)^* \\ \circ \\ \textcircled{\textcircled{G}} \mapsto \textcircled{G} \\ \circ \\ \textcircled{\textcircled{M}} \mapsto \textcircled{M} \\ \circ \\ \textcircled{G}\ (\mapsto)\ \epsilon \end{pmatrix} \spadesuit : \epsilon \Big)^* \begin{pmatrix} (\Sigma - \spadesuit)^* \\ \circ \\ \textcircled{G} \mapsto \textcircled{G} \\ \circ \\ \textcircled{M} \mapsto \textcircled{M} \end{pmatrix} \tag{4.18}$$

Eine Governor-Variable ist in der direkt folgenden Regel eventuell eine Füller-Variable ohne relative Positionsinformation oder es folgt keine Regel mehr, daher wird die entsprechende relative Positionsinformation optional entfernt ($\textcircled{G}\ (\mapsto)\ \epsilon$). Variablen ohne relative Positionsinformation werden als Füller-Variable von Regel zu Regel weitergereicht. Variablen, die sich auf kreuzende Wörter beziehen, fungieren in einer folgenden Regel als Governor ($\textcircled{\textcircled{G}} \mapsto \textcircled{G}$) oder als Modifier ($\textcircled{\textcircled{M}} \mapsto \textcircled{M}$). Des Weiteren haben Füller-Variablen mit relativer Positionsinformation innerhalb einer folgenden Regel die Funktion als Governor ($\textcircled{G} \mapsto \textcircled{G}$) oder als Modifier ($\textcircled{M} \mapsto \textcircled{M}$). Zuletzt werden die Markierungszeichen $\spadesuit$ über Epsilon entfernt ($\spadesuit : \epsilon$).

Im Folgenden wird angenommen, dass der Constraint-Wert, der von einer vorhergehenden Regel generiert wird, durch $\mathcal{P}$ bezeichnet wird. Hierbei handelt es sich gegebenenfalls um den Wert eines simultanen Constraints, da eine Regel verschiedene Lösungen simultan berechnen kann. Dieser Wert wird nun auf alle möglichen Positionen projiziert, die sich aus der Vereinigung der Position von $\mathcal{Q}^+$ und $\mathcal{P}$ ergeben können. Hierbei muss berücksichtigt werden, dass die Position von P alle Variablen aus $\mathcal{Q}^+$ enthält außer denen, die dort Modifier repräsentieren. Die Tupel-Projektion muss also relativ zu den Dependenzknoten-Teilproblemen geschehen. Das Erweitern von $\mathcal{P}$ auf die Variablen, die in $\mathcal{Q}^+$ Modifier repräsentieren, kann über die relative Positionsinformation $\blacksquare$ und somit über die rationale Sprache $\mathcal{U}$ realisiert werden, da diese innerhalb des Filters $\mathcal{F}_u$ auf $\textcircled{M}$ umgeschrieben wird:

$$Range(\mathcal{P} \circ \epsilon \leftarrow\!\shortmid \mathcal{U}) \tag{4.19}$$

Das bedeutet, dass $\mathcal{U}$ in keinem Teilproblem und somit auch nicht in $\mathcal{P}$ enthalten ist und daher über Epsilon entfernt wird ($\epsilon \leftarrow\!\shortmid \mathcal{U}$).

Nun lässt sich der Constraint-Wert $\mathcal{L}$ definieren, der aus der Kombination der Constraint-Werte $\mathcal{Q}^+$ und $\mathcal{P}$ resultiert, wobei hier das Umschreiben oder Entfernen von relativen Positionsinformationen berücksichtigt ist:

$$\mathcal{L} = Range\begin{pmatrix} \mathcal{P} \\ \circ \\ \epsilon \nleftarrow \mathcal{U} \\ \circ \\ \mathcal{F}_u \circ (\mathcal{I}^* \spadesuit \mathcal{Q} \spadesuit)^+ \mathcal{I}^* \circ \mathcal{F}_l \end{pmatrix} \tag{4.20}$$

Es muss jedoch noch einbezogen werden, dass Dummy-Schichtteilprobleme überall angenommen werden können. So kann über die simultanen lokalen Konsistenzregeln mehreren Zerlegungen nachgegangen werden, die eine unterschiedliche Tiefe besitzen. Daher ist der Constraint-Wert $\mathcal{L}$ mit Berücksichtigung der Dummy-Schichtteilprobleme definiert durch:

$$\mathcal{L} = Range\begin{pmatrix} \mathcal{P} \\ \circ \\ \epsilon \nleftarrow \mathcal{U} \\ \circ \\ \mathcal{F}_u \circ (\mathcal{I}^* \spadesuit \mathcal{Q} \spadesuit)^* \mathcal{I}^* \circ \mathcal{F}_l \end{pmatrix} \tag{4.21}$$

Über den Teilausdruck $(\mathcal{I}^* \spadesuit \mathcal{Q} \spadesuit)^* \mathcal{I}^*$ ist realisiert, dass die Zerlegungstiefe künstlich erhöht werden kann. Dieser ist gleichbedeutend zu dem Ausdruck $(\mathcal{I}^* \spadesuit \mathcal{Q} \spadesuit)^+ \mathcal{I}^* \cup \mathcal{I}^*$, bei dem über die Vereinigung mit $\mathcal{I}^*$ sichergestellt ist, dass $\mathcal{P}$ nicht eingeschränkt wird.

Nun kann anhand des Ausdrucks 4.21 eine simultane lokale Konsistenzregel $\mathcal{R}$ definiert werden, die einen Constraint-Wert auf einen anderen Constraint-Wert abbildet:

$$\mathcal{R} = \begin{pmatrix} \epsilon \nleftarrow \mathcal{U} \\ \circ \\ \mathcal{F}_u \circ (\mathcal{I}^* \spadesuit \mathcal{Q} \spadesuit)^* \mathcal{I}^* \circ \mathcal{F}_l \end{pmatrix} \tag{4.22}$$

Zudem kann über die lokale Konsistenzregel $\mathcal{R}$ der Lösungs-Constraint-Wert $\mathcal{L}$ einer Zerlegungsebene berechnet werden:

$$\mathcal{L} = Range(\mathcal{P} \circ \mathcal{R}) \tag{4.23}$$

Eine simultane lokale Konsistenzregel bildet einen Constraint-Wert auf einen anderen ab und somit implizit einen Constraint auf einen anderen. Hierbei kann das Ergebnis der Anwendung einer Regel ein simultaner Constraint sein, der über unterschiedliche Constraint-Systeme definiert ist. Die einzelnen Constraint-Systeme beruhen auf den relativen Positionsinformationen, die mit den Variablen verknüpft sind und die über die Regeln hinzugefügt, verändert oder entfernt werden. Das bedeutet, dass eine simultane lokale Konsistenzregel nicht nur implizit eine Abbildung von Constraint-Problemen auf Constraint-Probleme ist, sondern auch implizit eine Abbildung von Constraint-Systemen auf Constraint-Systeme.

4.5.2 Zusammenfassen simultaner lokaler Konsistenzregeln

Eine simultane lokale Konsistenzregel repräsentiert alle möglichen lokalen Konsistenzregeln, die für eine beliebige Zerlegungsebene angenommen werden können. Daher kann ein und dieselbe simultane lokale Konsistenzregel für alle Zerlegungsebenen herangezogen werden. Die Constraint-Menge ergibt sich dann implizit über die einzelnen Zerlegungsebenen. Das bedeutet, dass über diese simultane lokale Konsistenzregel für die verschiedenen Ebenen implizit reguläre quantifizierte Constraints definiert sind. Wenn eine simultane lokale Konsistenzregel $\mathcal{R}$ und eine fixe Zerlegungstiefe k vorliegt, wird im Folgenden für die implizite Constraint-Menge $C_{\mathcal{R}^k}$ geschrieben.

Es sei eine simultane lokale Konsistenzregel $\mathcal{R}$, eine feste Zerlegungstiefe k und ein Constraint-Problem $P = (C_{\mathcal{R}^k}, con)$ gegeben. Des Weiteren sei ein Top-Constraint-Wert $\mathcal{T}$ und ein Universal-Constraint $(L(\bar{1}_{con}), L(con))$ angenommen, der den gleichen Typ wie das Constraint-Problem besitzt und im Folgenden als *Bottom-Constraint* bezeichnet wird. Dann ist der Wert des Lösungs-Constraints $(\mathcal{S}, L(con))$ definiert durch:

$$\mathcal{S} = Range(\mathcal{T} \circ \underbrace{\mathcal{R} \circ \mathcal{R} \circ \cdots \circ \mathcal{R}}_{k} \circ L(\bar{1}_{con})) \tag{4.24}$$

Es werden die simultanen lokalen Konsistenzregeln der einzelnen Problemzerlegungsebenen unter Einbeziehung des Top- und Bottom-Constraints zu einer simultanen lokalen Konsistenzregel zusammengefasst. Über den Bottom-Constraint wird hierbei die Position, die von der letzten Regel generiert

wird, explizit gesetzt, und über den Top-Constraint wird gewährleistet, dass alle Variablen der letzten Zerlegungsebene als Modifier dem Dummy-Wurzelknoten direkt untergeordnet sind. Über das Zusammenfassen der Regeln werden sukzessiv neue simultane lokale Konsistenzregeln generiert. So ergibt sich der Lösungs-Constraint implizit aus der simultanen Anwendung aller möglichen gleich langen Sequenzen von lokalen Konsistenzregeln, die bei einer Problemzerlegung der Tiefe k angenommen werden können. Dies beinhaltet auch die Problemzerlegungen, die die Tiefe k über die Einführung von Dummy-Schichtteilproblemen erreichen. Hierbei geschehen das Ermitteln der Zerlegungen und die Anwendung der Constraints simultan. Der Wert des Lösungs-Constraint enthält letztlich keine relativen Positionsinformationen mehr. Dies macht deutlich, dass diese Informationen ausschließlich für die Spezifizierung der simultanen lokalen Konsistenzregeln und für die Kommunikation zwischen ihnen herangezogen werden.

4.5.3 Parsingkomplexität

Wenn die Grammatik fest ist und somit eine feste simultane lokale Konsistenzregel gegeben ist und wenn zusätzlich eine feste Tiefe für die Problemzerlegungen angenommen wird, ist Parsing in linearer Zeit möglich. Dies verhält sich ähnlich zur FSIG (vgl. Abschnitt 4.1.1): Es sei die feste Zerlegungstiefe k, die simultane lokale Konsistenzregel R und das Constraint-Problem $P = (C_{\mathcal{R}^k}, con)$ angenommen, das das Parsingproblem zu einem gegebenen Eingabesatz repräsentiert. Des Weiteren sei der Top-Constraint-Wert $\mathcal{T}$ und der Bottom-Constraint $A(\bar{1}_{con})$ gegeben. Dann liegt die Zeit- und Platzkomplexität des Lösens in $O(|\mathcal{T}| \times |\mathcal{R}|^k \times |A(\bar{1}_{con})|)$. Wenn n die Länge des Eingabesatzes bezeichnet, wird dieser durch n Variablen repräsentiert mit $|con| = n$. Da $|A(\bar{1}_{con})|$ linear mit $|con|$ und somit mit n wächst, kann $|A(\bar{1}_{con})|$ durch n ersetzt werden. Wenn die Grammatik fest ist, sind auch die lokale Konsistenzregel und der Top-Constraint-Wert fest. So kann $|\mathcal{T}|$ durch die Konstante T und $|\mathcal{R}|^k$ durch die Konstante R ersetzt werden. Dann liegt die Zeit- und Platzkomplexität des Parsings in $O(n) = O(T \times R \times n)$. Demnach steigt der Zeit- und Platzbedarf beim Parsing linear mit der Eingabelänge.

Ohne die Einbeziehung von Dummy-Schichtteilproblemen entspricht die maximale Tiefe einer Problemzerlegung der Kardinalität der Variablenmenge des Constraint-Problems (siehe Abschnitt 4.3). Wenn das Constraint-Pro-

blem $(C_{\mathcal{R}^k}, con)$ gegeben ist, bedeutet dies, dass nach $|con|$ Regelanwendungen bereits alle möglichen Zerlegungen für die Variablenmenge *con* berechnet wurden. Alle Anwendungen, die darüber hinaus vollzogen werden, ändern die Lösung nicht und würden ausschließlich Dummy-Teilprobleme einbeziehen. Wenn die Zerlegungstiefe der Kardinalität des Typs des Constraint-Problems entspricht, werden somit keine relevanten Zerlegungen ausgeschlossen. Es sei das Constraint-Problem $(C_{\mathcal{R}^{|con|}}, con)$ angenommen, dann liegt die Zeit- und Platzkomplexität des Lösens in $O(|\mathcal{T}| \times |\mathcal{R}|^{|con|} \times |A(\bar{1}_{con})|)$. Wenn n die Länge des Eingabesatzes ist mit $|con| = n$, kann n für $|A(\bar{1}_{con})|$ eingesetzt werden. Des Weiteren ist $\mathcal{T}$ fest, und dessen Größe kann durch die Konstante T ersetzt werden. Dann liegt die Zeit- und Platzkomplexität des Parsings in $O(|\mathcal{R}|^n \times n) = O(T \times |\mathcal{R}|^n \times n)$. Demnach steigt hier der Zeit- und Platzbedarf beim Parsing im schlechtesten Fall exponentiell mit der Eingabelänge.

4.5.4 Ambige Eingabe

Die Wörter eines Satzes können morphologische oder lexikalische Ambiguitäten beinhalten. In Abschnitt 3.4.2 wurde anhand eines Beispiels gezeigt, wie solche Ambiguitäten über ein simultanes Constraint-System berücksichtigt werden können. Dort wird für jede Lesart eines Satzes ein separates Constraint-System angenommen. Diese Constraint-Systeme werden dann über ein simultanes Constraint-System kompakt repräsentiert. Das Resultat mehrerer Lesarten ist somit ein simultaner Bottom-Constraint, der hier als ambige Eingabe bezeichnet wird.

Angenommen, es existieren zu einem Eingabesatz mehrere Lesarten, die durch die Constraint-Systeme $CS_1 = (S,D,V^{\tau_1},I)$,$CS_2 = (S,D,V^{\tau_2},I)$,..., $CS_n = (S,D,V^{\tau_n},I)$ abgebildet werden, dann ist ein entsprechendes simultanes Constraint-System gegeben durch $\widetilde{CS} = (S, D, V^{\tau_1}|V^{\tau_2}|\ldots|V^{\tau_n}, I)$. Des Weiteren sei eine simultane lokale Konsistenzregel $\mathcal{R}$, eine feste Zerlegungstiefe k und ein simultanes Constraint-Problem $P = (C_{\mathcal{R}^k}, con^{\tau_1}|con^{\tau_2}|\ldots|con^{\tau_n})$ mit $con \subseteq V$ gegeben. Die Constraints aus der Menge $C_{\mathcal{R}^k}$ sind wie in Abschnitt 3.4.3 beschrieben simultane Constraints, die für alle Constraint-Systeme wohldefiniert sind. So ist der Bottom-Constraint definiert durch $(\bar{1}_{con^{\tau_1}}|\bar{1}_{con^{\tau_2}}|\ldots|\bar{1}_{con^{\tau_n}}, con^{\tau_1}|con^{\tau_2}|\ldots|con^{\tau_n})$ mit $con \subseteq V$. Ausgehend von diesem Bottom-Constraint kann die Lösung berechnet werden. Gegeben den Top-Constraint-Wert $\mathcal{T}$, dann ist der Wert des simultanen Lösungs-Constraints

$(\mathcal{S}, L(con^{\tau_1}|con^{\tau_2}|\ldots|con^{\tau_n}))$ definiert durch:

$$\mathcal{S} = Range(\mathcal{T} \circ \underbrace{\mathcal{R} \circ \mathcal{R} \circ \cdots \circ \mathcal{R}}_{k} \circ L(\bar{1}_{con^{\tau_1}}|\bar{1}_{con^{\tau_2}}|\ldots|\bar{1}_{con^{\tau_n}})) \tag{4.25}$$

Auch wenn die einzelnen simultan repräsentierten Constraint-Systeme Variablenmengen unterschiedlicher Kardinalität besitzen, kann auf diese Weise die Zerlegung und Lösung berechnet werden.

In den folgenden Abschnitten, in denen der Zerlegungsansatz weiterentwickelt wird, wird zur Vereinfachung von einer unambigen Eingabe ausgegangen. Dort ist aber auch eine ambige möglich. Hierzu muss für den Bottom-Constraint lediglich ein simultaner Constraint angenommen werden.

4.5.5 Bottom-Up-Lösen

Ein Top-Down-Lösen ist problematisch, da allen möglichen Zerlegungen nachgegangen werden muss, die für eine bestimmte Zerlegungstiefe möglich sind. Wenn die Tiefe der Zerlegungen etwas höher gesetzt ist, wäre hierbei die Berechnung der Lösung bei einer adäquaten Grammatik wegen unhandhabbarer Zwischenergebnisse praktisch nicht umsetzbar. Dies verhält sich analog zu den Problematiken der Umsetzung der WCDG über quantifizierte reguläre Constraints in Abschnitt 4.2.

Trotz einer Top-Down-Lösungsabfolge kann das Parsingproblem alternativ bottom-up gelöst werden. Dies ergibt sich aus dem Zusammenfassen simultaner lokaler Konsistenzregeln und der Assoziativität der Kompositionsoperation (vgl. Abschnitt 4.4.4). In diesem Fall kann eine konkrete Position angenommen werden, die von der letzten Regel generiert wird. So muss im Lösungsverlauf bottom-up nur den Problemzerlegungen nachgegangen werden, die in dieser Position münden. Dies wirkt sich günstig auf die Größe der Zwischenergebnisse aus. Daher steht hier ein Bottom-Up-Lösen im Fokus.

Für ein Bottom-Up-Lösen ist der Algorithmus 2 gegeben, der den Lösungs-Constraint für ein Constraint-Problem anhand von lokalen Konsistenzregeln und einer gesetzten festen Tiefe berechnet. Innerhalb der For-Schleife in den Zeilen 2 bis 4 werden die simultanen lokalen Konsistenzregeln zu den einzelnen Zerlegungsebenen nacheinander bottom-up zu einer zusammengefasst.

Es lässt sich auch ein Fix-Point-Algorithmus definieren, der eine Lösung

Algorithmus 2 Simultane perfekte Relaxierung mit Bottom-Up-Lösen und fester Tiefe.

Eingabe: Eine simultane lokale Konsistenzregel $\mathcal{R}$ und eine feste Tiefe k für das Constraint-Problem $P = (C_{\mathcal{R}^k}, con)$. Der Wert $\mathcal{T}$ des Top-Constraints und der Bottom-Constraint $(A(\bar{1}_{con}), A(con))$.

Ausgabe: Der Lösungs-Constraint von P.

1: $\mathcal{L} \leftarrow A(\bar{1}_{con})$
2: **for** $i \leftarrow k$ **downto** 1 **do**
3: $\quad \mathcal{L} \leftarrow \mathcal{R} \circ \mathcal{L}$
4: **end for**
$\mathcal{L} \leftarrow \mathcal{T} \circ \mathcal{L}$
5: **return** $(Range(\mathcal{L}), A(con))$

für eine ungebundene Zerlegungstiefe berechnet. Die Constraint-Menge des Constraint-Problems wird dann durch $C_{\mathcal{R}^\infty}$ bezeichnet. Für solch eine ungebundene Tiefe ist der Algorithmus 3 gegeben. Innerhalb der While-Schlei-

Algorithmus 3 Simultane perfekte Relaxierung mit Bottom-Up-Lösen und ungebundener Tiefe.

Eingabe: Eine simultane lokale Konsistenzregel $\mathcal{R}$ für das Constraint-Problem $P = (C_{\mathcal{R}^\infty}, con)$. Der Wert $\mathcal{T}$ des Top-Constraints und der Bottom-Constraint $(A(\bar{1}_{con}), A(con))$.

Ausgabe: Der Lösungs-Constraint von P.

1: $\mathcal{L} \leftarrow A(\bar{1}_{con})$
2: **while** $\mathcal{L}' \leftarrow \mathcal{R} \circ \mathcal{L} \neq \mathcal{L}$ **do**
3: $\quad \mathcal{L} \leftarrow \mathcal{L}'$
4: **end while**
$\mathcal{L} \leftarrow \mathcal{T} \circ \mathcal{L}$
5: **return** $(Range(\mathcal{L}), A(con))$

fe in den Zeilen 2 bis 4 werden die simultanen lokalen Konsistenzregeln zu den möglichen Zerlegungsebenen nacheinander bottom-up zusammengefasst. Wenn über das Zusammenfassen keine neue simultane lokale Konsistenzregel mehr generiert werden kann, terminiert die Schleife. Dies geschieht spätestens nach $|con|+1$ Regelzusammenfassungen. Dann sind bereits alle möglichen Zerlegungen für *con* berechnet. Die letzte Regelzusammenfassung führt hierbei

zu keiner neuen Regel.

Der Algorithmus ähnelt dem Fix-Point-Parsing-Algorithmus von Roche (1997, Abschnitt 8.3). Dort wird eine Grammatik durch einen Regel-FST implementiert, die iterativ über die Kompositionsoperation angewendet wird. Es werden so top-down nichtkreuzende Klammerstrukturen erstellt. Während des Parsings können Analysefragmente oder ganze Analysen optional ohne Veränderung weitergereicht werden. Dadurch werden Analysen mit unterschiedlicher Einbettungstiefe möglich. Hierdurch können über den klassischen Kompositionsalgorithmus jedoch Analysen generiert werden, die bereits von einer vorhergehenden Regelanwendung erstellt worden sind (bezüglich des Kompositionsalgorithmus siehe z. B. Allauzen et al., 2009). Auch in dem Bottom-Up-Parsing-Ansatz von Oflazer (2003) kann dieser Fall auftreten, da dort Analysefragmente optional weitergereicht werden können. Dies ist innerhalb dieser Ansätze jedoch unproblematisch, da doppelte Analysen über die Äquivalenztransformationen über FSMs wieder zusammengefasst werden. Im Grunde sind diese Doppelgenerierungen nicht sichtbar, bedeuten aber computationellen Mehraufwand.

Das erneute Generieren von bereits durchgeführten Analysen kann auch bei der Zusammenfassung von simultanen lokalen Konsistenzregeln auftreten. Grund hierfür sind die Füller-Variablen, über die Analysen durchgereicht werden können. So können beim Zusammenfassen simultaner lokaler Konsistenzregeln einzelne lokale Konsistenzregeln generiert werden, die bereits zuvor innerhalb einer anderen Ebene erstellt wurden. Durch die Äquivalenztransformationen über FSMs stellt dies aber hier kein Problem dar. Jedoch muss über die Äquivalenzoperationen Mehraufwand betrieben werden, um die doppelten Analysen zusammenzulegen. Im Folgenden wird diese Art der Zerlegung als *unkontrollierte Zerlegung* bezeichnet.

4.5.6 Bottom-Up-Lösen mit Optimalzerlegung

In diesem Abschnitt wird der Zerlegungsansatz aus Abschnitt 4.5.5 um eine Optimalzerlegung erweitert. Hierzu wird zusätzlich verlangt, dass ein nichtunäres Dependenzknoten-Teilproblem mindestens eine Variable umfasst, die in einem Dependenzknoten-Teilproblem einer direkt folgenden Regel enthalten ist. Aus Top-Down-Sicht entspricht dies dem Unterpunkt (i) der Optimalzerlegungsdefinition 33 aus Abschnitt 4.3.3. Des Weiteren wird gefordert,

dass das initiale Schicht-Teilproblem und somit die letzte Regel alle unären Teilprobleme beinhaltet. Dies entspricht ergänzend dem Unterpunkt (ii) der Optimalzerlegungsdefinition. Um dies umzusetzen sind die Filter $\mathcal{F}_u$ und $\mathcal{F}_l$ aus Abschnitt 4.5.1 entsprechend der Optimalzerlegung angepasst. Wie schon zuvor sind die Filter aus Top-Down-Sicht definiert.

So ist der Filter $\mathcal{F}_u$ in 4.17 aus Abschnitt 4.5.1 wie folgt angepasst:

$$\mathcal{F}_u = \Big(\ (\lhd\,\Gamma\diamond L(D)\,\rhd)^*\quad \epsilon:\spadesuit \quad \begin{pmatrix} (\lhd\,\Gamma\diamond L(D)\,\rhd)^* \\ \circ \\ \textcircled{G} \leftarrowtail \textcircled{\textcircled{G}} \\ \circ \\ \textcircled{M} \leftarrowtail \textcircled{\textcircled{M}} \\ \circ \\ \textcircled{M} \leftarrowtail \textcircled{G} \\ \circ \\ \blacksquare \leftarrowtail \textcircled{M} \end{pmatrix} \quad \epsilon:\spadesuit\ \Big)^*\ (\lhd\,\Gamma\diamond L(D)\,\rhd)^* \tag{4.26}$$

Modifier-Variablen sind in der direkt vorhergehenden Regel in keinem Teilproblem involviert ($\blacksquare \leftarrowtail \textcircled{M}$). Governor-Variablen fungieren hingegen in einer vorhergehenden Regel als Modifier, soweit sie nicht dem Dummy-Wurzelknoten untergeordnet sind ($\textcircled{M} \leftarrowtail \textcircled{G}$). Variablen, die sich auf kreuzende Wörter beziehen, fungieren in einer vorhergehenden Regel als Governor ($\textcircled{G} \leftarrowtail \textcircled{\textcircled{G}}$) oder als Modifier ($\textcircled{M} \leftarrowtail \textcircled{\textcircled{M}}$). Dies entspricht soweit der Originaldefinition aus Abschnitt 4.5.1. Nun wird zusätzlich gefordert, dass eine Variable in der direkt vorhergehenden Regel eine relative Positionsinformation enthalten muss ($(\lhd\,\Gamma\diamond L(D)\,\rhd)^*$). Das heißt, dass die relativen Positionsinformationen erst über die letzte Regel entfernt werden können. Hierdurch wird Unterpunkt (i) der Optimalzerlegungsdefinition umgesetzt. Die Füller-Variablen, werden in dieser Definition nicht umgeschrieben. Dadurch erhält die direkt vorhergehende Regel die Information darüber, ob es sich hier um eine Füller-Variable handelt.

Der Filter $\mathcal{F}_l$ in 4.18 aus Abschnitt 4.5.1 ist ergänzend wie folgt angepasst:

$$\mathcal{F}_l =$$

$$\left(\begin{array}{c}(\Sigma - \spadesuit)^* \\ \circ \\ \blacksquare\,(\mapsto)\ \textcircled{G} \\ \circ \\ \blacksquare\,(\mapsto)\ \textcircled{M} \\ \circ \\ \blacksquare\,(\mapsto)\ \epsilon\end{array}\right) \spadesuit : \epsilon \left(\begin{array}{c}\left(\begin{array}{c}(\Sigma - \spadesuit)^* \\ \circ \\ (\Sigma - \textcircled{M})^* \\ \circ \\ \textcircled{G} \mapsto \epsilon\end{array}\right) \\ \cup \\ \left(\begin{array}{c}(\Sigma - \spadesuit)^* \\ \circ \\ \Sigma^* \textcircled{M} \Sigma^* \\ \circ \\ \textcircled{G}\,(\mapsto)\ \blacksquare \\ \circ \\ \textcircled{M}\,(\mapsto)\ \blacksquare \\ \circ \\ \textcircled{\textcircled{G}}\,(\mapsto)\ \{\textcircled{G}, \blacksquare\} \\ \circ \\ \textcircled{\textcircled{M}}\,(\mapsto)\ \{\textcircled{M}, \blacksquare\} \\ \circ \\ \Sigma^*\{\textcircled{G}, \textcircled{M}\}\Sigma^*\end{array}\right)\end{array}\right) \spadesuit : \epsilon \,)^* \left(\begin{array}{c}(\Sigma - \spadesuit)^* \\ \circ \\ \blacksquare\,(\mapsto)\ \textcircled{G} \\ \circ \\ \blacksquare\,(\mapsto)\ \textcircled{M} \\ \circ \\ \blacksquare\,(\mapsto)\ \epsilon\end{array}\right) \tag{4.27}$$

Bei den Dependenzknoten-Teilproblemen wird hier zwischen dem unären ($(\Sigma - \textcircled{M})^*$) und dem nichtunäre Fall ($\Sigma^* \textcircled{M} \Sigma^*$) unterschieden. Im Fall eines unären Dependenzknoten-Teilproblems wird die Positionsinformation der Governor-Variable über Epsilon entfernt ($\textcircled{G} \mapsto \epsilon$), da keine Regel folgt. Ebenso wird die Positionsinformation der Governor-bezogenen Füller-Variablen optional auf Epsilon umgeschrieben ($\blacksquare\,(\mapsto)\ \epsilon$). Beide Fälle betreffen die letzte Regel, in der alle relativen Positionsinformationen entfernt werden. Die Governor-Variable innerhalb eines nichtunären Dependenzknoten-Teilproblems ist in einer folgenden Regel entweder Governor oder eine Füller-Variable ($\textcircled{G}\,(\mapsto)\ \blacksquare$). Gleiches gilt für die Modifier ($\textcircled{M}\,(\mapsto)\ \blacksquare$). Variablen, die sich auf kreuzende Wörter beziehen, fungieren in einer folgen-

den Regel als Governor, Modifier oder Füller-Variable (Ⓖ (↦) { Ⓖ, 🅖} und Ⓜ (↦) { Ⓜ, 🅜}). Des Weiteren fungieren Füller-Variablen innerhalb einer direkt folgenden Regel als Füller-Variable oder als Governor (🅖 (↦) Ⓖ) oder als Modifier (🅜 (↦) Ⓜ). Hier besteht also die Information darüber, ob eine Variable in der direkt folgenden Regel eine Füller-Variable ist oder nicht. Hierauf aufbauend wird letztlich gefordert, dass innerhalb eines Dependenzknoten-Teilproblems mindestens eine der Variablen in der direkt folgenden Regel Governor oder Modifier sein muss ($\Sigma^*\{$ Ⓖ, Ⓜ$\}\Sigma^*$). Hierdurch wird Unterpunkt (ii) der Optimalzerlegungsdefinition realisiert.

Ausgehend von den beiden hier definierten Filtern $\mathcal{F}_u$ und $\mathcal{F}_l$ kann, analog zu dem Ausdruck 4.22 in Abschnitt 4.5.1, eine simultane lokale Konsistenzregel konstruiert werden, die eine Optimalzerlegung umsetzt. Diese kann jedoch die relative Positionsinformation 🅜 enthalten, die innerhalb des Wertes des Top-Constraints berücksichtigt werden muss. Daher ist die Definition des Top-Constraints-Wertes $\mathcal{T}$ folgend angepasst:

$$\mathcal{T} = \begin{pmatrix} \epsilon \nleftarrow \text{Ⓜ} \\ \circ \\ \mathcal{M}^+ \\ \circ \\ \text{Ⓜ } (\mapsto) \text{ 🅜} \end{pmatrix} \tag{4.28}$$

Eine Variable, die dem Dummy-Wurzelknoten direkt untergeordnet ist, kann in der letzten Zerlegungsebene eine Füller-Variable sein (Ⓜ (↦) 🅜).

Ausgehend von einer simultanen lokalen Konsistenzregel, die eine Optimalzerlegung implementiert, kann für die Lösungsberechnung der Bottom-Up-Lösungsalgorithmus 3 aus Abschnitt 4.5.5 herangezogen werden. Dieser benötigt hier eine zusätzliche Regelanwendung, da sich zwei generierte Regeln darin unterscheiden können, ob sie die relativen Positionsinformationen Ⓖ und Ⓜ oder 🅖 und 🅜 enthalten. Daher wird eine zusätzliche Regelanwendung benötigt, um einheitlich auf 🅖 und 🅜 umzuschreiben.

Um diese zusätzliche Regelanwendung einzusparen und um die Prüfung auf Problemstabilität effizienter zu gestalten, kann der Bottom-Up-Lösungsalgorithmus 4 herangezogen werden. Innerhalb der Repeat-Until-Schleife in den Zeilen 2 bis 4 ist die Abbruchbedingung Σ^* { Ⓖ, Ⓜ} $\Sigma^* \circ \mathcal{L} = A(\varnothing)$ umgesetzt. Wenn keine Variablen mehr vorhanden sind, die in der nächsten zusammengefassten simultanen lokalen Konsistenzregel als Governor oder Modifier

Algorithmus 4 Simultane perfekte Relaxierung mit Bottom-Up-Lösen, Optimalzerlegung und ungebundener Tiefe.

Eingabe: Eine simultane lokale Konsistenzregel $\mathcal{R}$ für das Constraint-Problem $P = (C_{\mathcal{R}^\infty}, con)$, die eine Optimalzerlegung implementiert. Der Wert $\mathcal{T}$ des Top-Constraints und der Bottom-Constraint $(A(\bar{1}_{con}), A(con))$.

Ausgabe: Der Lösungs-Constraint von P.

1: $\mathcal{L} \leftarrow A(\bar{1}_{con})$
2: **repeat**
3: $\quad \mathcal{L} \leftarrow \mathcal{R} \circ \mathcal{L}$
4: **until** $\Sigma^* \{ \textcircled{G}, \textcircled{M} \} \Sigma^* \circ \mathcal{L} = A(\varnothing)$
$\mathcal{L} \leftarrow \mathcal{T} \circ \mathcal{L}$
5: **return** $(Range(\mathcal{L}), A(con))$

fungieren können, wird die Analyse gestoppt. Das Testen der Bedingung muss aber nicht zwingend über die Kompositionsoperation umgesetzt werden. Es kann alternativ geprüft werden, ob die Zeichen Ⓖ oder Ⓜ in der Eingabezeichenkette eines erfolgreichen Pfades des WFSTs $\mathcal{L}$ vorhanden sind. Auf diese Weise kann auf diese Abbruchbedingung sehr effizient getestet werden. Die Repeat-Until-Schleife terminiert nach höchstens $|con| + 1$ Regelzusammenfassungen, da dann bereits alle möglichen Zerlegungen für *con* berechnet wurden und da die letzte Regelzusammenfassung einen WFST zur Folge hat, dessen Eingabe ausschließlich Füller-Variablen akzeptiert. Die letzte Zusammenfassung führt somit zu keiner neuen Regel. Die Abbruchbedingung ist ähnlich zu der in Oflazer (2003). Dort wird iterativ eine Regel angewendet, bis das Ergebnis die leere Sprache akzeptiert, es sei denn, es ist eine maximal gesetzte Anzahl an Regelanwendungen erreicht (siehe Abschnitt 4.1.1).

4.5.7 Memoisation von Zwischenergebnissen

In Didakowski (2008b) wird im Kontext eines erweiterten endlichen Parsingansatzes vorgeschlagen, während der Analyse Material (Zustände und Übergänge) innerhalb von Klammerungen durch Indizes zu ersetzen. Dies ist möglich, da FSMs bezüglich der Substitution abgeschlossen sind. Dadurch zeigt sich beim Parsing ein verringertes Anwachsen von Zwischenergebnissen. Hier wird hingegen auf die Memoisationstechnik in Yli-Jyrä (2012) zurückgegriffen (siehe Abschnitt 4.1.1). Diese basiert auf einer Methode, die in Yli-Jyrä

(2012, Abschnitt 5.1) als *Vergessen-Komposition* (forgetting composition) bezeichnet wird und der Projektionsoperation über Constraints ähnelt. Die Idee ist hierbei, im Laufe der Anwendung einer Kompositionskaskade stets die zweite Projektion zu 'vergessen' und somit bestimmte Analyseteile zu entfernen. Die vergessenen Einzelteile können später über die Kompositionsoperation wieder eingefügt werden, wobei die korrekte Behandlung der Gewichtungen berücksichtigt werden muss. Die Teile, die für die Ergebnisberechnung final zusammengefügt werden, werden in Yli-Jyrä (2012, Abschnitt 5.3) *memoisierte Kaskade* (memoizing composition) genannt.

Im Folgenden sei eine simultane lokale Konsistenzregel $\mathcal{R}$ zu dem Constraint-Problem $P = (C_{\mathcal{R}^k}, con)$ über dem Constraint-System (S, D, V) angenommen, wobei eine feste Tiefe $k > 1$ für die Problemzerlegung vorliegt. Des Weiteren sei der Top-Constraint-Wert $\mathcal{T}$ und der Bottom-Constraint-Wert $L(\bar{1}_{con})$ gegeben.

Die Vergessen-Komposition kann in die Berechnung des Zusammenfassens von simultanen lokalen Konsistenzregeln einbezogen werden. Dann wird nicht mehr der Lösungs-Constraint von P berechnet, sondern der Constraint-Wert mit der relativen Position des Top-Constraints:

$$\mathcal{S} = \mathcal{T} \cap Dom(\underbrace{\mathcal{R} \circ Dom(\mathcal{R} \circ Dom(\underbrace{\ldots}_{n} \circ Dom(\mathcal{R} \circ L(\bar{1}_{con}))\underbrace{\ldots}_{n}))}_{k}) \tag{4.29}$$

Das Vergessen ist über die Ermittlungen des Definitionsbereiches umgesetzt. Hierbei entspricht das Sprachgewicht von $\mathcal{S}$ der besten Konsistenzstufe von P ($w[\mathcal{S}] = blevel(P)$).

Um trotz der Vergessen-Komposition den Lösungs-Constraint von P ermitteln zu können, müssen Zwischenergebnisse gespeichert und anhand dieser die Lösung 'zusammengesetzt' werden. Es muss demnach eine memoisierte Kaskade einbezogen werden. Im Folgenden wird zunächst für multiplikativ idempotente c-Semiringe gezeigt, wie die gespeicherten Zwischenergebnisse beschaffen sind und wie über diese der Lösungs-Constraint berechnet werden kann. Danach wird der Ansatz auf multiplikativ nichtidempotente c-Semiringe erweitert.

Es wird angenommen, dass für jede der k Zerlegungsebenen eine zusammengefasste simultane lokale Konsistenzregel $\widehat{\mathcal{R}}_i$ mit $1 \leq i \geq k$ definiert ist:

$$\widehat{\mathcal{R}}_i = \begin{cases} \mathcal{R}_i \circ L(\bar{1}_{con}) & \text{falls } i = 1 \\ \mathcal{R}_i \circ \widehat{\mathcal{R}}_{i-1} & \text{sonst} \end{cases} \tag{4.30}$$

Dann kann aus diesen zusammengefassten simultanen lokalen Konsistenzregeln der Wert des Lösungs-Constraints von P berechnet werden:

$$\mathcal{S} = Range(\underbrace{\ldots}_{n} Range(Range(\mathcal{T} \circ \widehat{\mathcal{R}}_k \circ \widehat{\mathcal{R}}_{k-1}) \circ \widehat{\mathcal{R}}_{k-2}) \underbrace{\circ \ldots}_{n} \circ \widehat{\mathcal{R}}_1) \tag{4.31}$$

Hier muss die abstrakte Multiplikation des c-Semirings idempotent sein, da die Gewichtungen der Regelanwendungen sowohl beim Berechnen der Zwischenergebnisse als auch beim Ermitteln der Lösung abstrakt aufmultipliziert werden.

Um multiplikativ nichtidempotente c-Semiringe verwenden zu können, kann eine zusammengefasste simultane lokale Konsistenzregel $\overline{\mathcal{R}}_i$ mit $1 \leq i \geq k$ wie folgt definiert werden:

$$\overline{\mathcal{R}}_i = \begin{cases} \mathcal{R}_i \circ L(\bar{1}_{con}) & \text{falls } i = 1 \\ \mathcal{R}_i \circ Support(\overline{\mathcal{R}}_{i-1}) & \text{sonst} \end{cases} \tag{4.32}$$

Über die Support-Operation wird gewährleistet, dass bei den Zwischenergebnissen die Gewichtungen der einzelnen Regelanwendungen nicht abstrakt aufmultipliziert werden. Die Aufmultiplizierung soll ausschließlich bei der Berechnung der Lösung geschehen. So kann mithilfe dieser zusammengefassten simultanen lokalen Konsistenzregeln der Wert des Lösungs-Constraints von P auch für nicht idempotente c-Semiringe berechnet werden:

$$\mathcal{S} = Range(\underbrace{\ldots}_{n} Range(Range(\mathcal{T} \circ \overline{\mathcal{R}}_k \circ \overline{\mathcal{R}}_{k-1}) \circ \overline{\mathcal{R}}_{k-2}) \underbrace{\circ \ldots}_{n} \circ \overline{\mathcal{R}}_1) \tag{4.33}$$

In Yli-Jyrä (2012) werden über die Zwischenergebnisse ausschließlich die besten Analysen berechnet. Dies kann hier über die Berechnung der abstrakten Lösung geschehen. Es soll aber nicht erst die komplette Lösung ermittelt werden, bevor die abstrakte Lösung berechnet wird. Daher sollen die Wertetupel, die nicht Teil der abstrakten Lösung sind, analog zu Yli-Jyrä (2012) so früh wie möglich ausgeschlossen werden. Hierzu werden sowohl $\widehat{\mathcal{R}}_1, \widehat{\mathcal{R}}_2, \ldots, \widehat{\mathcal{R}}_k$ als auch $\overline{\mathcal{R}}_1, \overline{\mathcal{R}}_2, \ldots, \overline{\mathcal{R}}_k$ herangezogen. Über die Gewichte von $\overline{\mathcal{R}}_1, \overline{\mathcal{R}}_2, \ldots, \overline{\mathcal{R}}_k$ kann das korrekte c-Semiring-Element der abstrakten Lösung

berechnet werden und über die Gewichte von $\widehat{\mathcal{R}}_1, \widehat{\mathcal{R}}_2, \ldots, \widehat{\mathcal{R}}_k$ ist es möglich, Analysen frühzeitig auszuschließen. Für ein beliebiges i mit $1 \leq i \geq k$ gilt hierbei $Support(\widehat{\mathcal{R}}_i) \circ Range(\overline{\mathcal{R}}_i) = \overline{\mathcal{R}}_i$. So müssen bezüglich $\overline{\mathcal{R}}_1, \overline{\mathcal{R}}_2, \ldots, \overline{\mathcal{R}}_k$ nur die Wertebereiche berücksichtigt werden. So kann eine Sequenz K gebildet werden, die im Folgenden als memoisierte Kaskade verstanden wird:

$$\begin{gathered}\mathcal{K} = ((\widehat{\mathcal{K}}_1, \overline{\mathcal{K}}_1),(\widehat{\mathcal{K}}_2, \overline{\mathcal{K}}_2),\ldots,(\widehat{\mathcal{K}}_k, \overline{\mathcal{K}}_k)) = \\ ((\widehat{\mathcal{R}}_1, Range(\overline{\mathcal{R}}_1)),(\widehat{\mathcal{R}}_2, Range(\overline{\mathcal{R}}_2)),\ldots,(\widehat{\mathcal{R}}_k, Range(\overline{\mathcal{R}}_k)))\end{gathered} \tag{4.34}$$

Nun lässt sich der Algorithmus 5 definieren, der im Kontext einer Optimalzerlegung bei einer ungebundenen Tiefe eine memoisierte Kaskade berechnet und dabei die Vergessen-Komposition einbezieht. Im Gegensatz zu dem Parsingansatz in Yli-Jyrä (2012) ist hier eine ungebundene Tiefe möglich. Innerhalb der While-Schleife in den Zeilen 3 bis 13 wird bottom-up die

Algorithmus 5 Berechnung einer memoisierten Kaskade über eine Optimalzerlegung.

Eingabe: Eine simultane lokale Konsistenzregel $\mathcal{R}$ für das Constraint-Problem $P = (C_{\mathcal{R}^\infty}, con)$, die eine Optimalzerlegung implementiert und der Bottom-Constraint $(A(\bar{1}_{con}), A(con))$.

Ausgabe: Eine memoisierte Kaskade $K = ((\widehat{\mathcal{K}}_1,\overline{\mathcal{K}}_1),(\widehat{\mathcal{K}}_2, \overline{\mathcal{K}}_2),\ldots,(\widehat{\mathcal{K}}_k, \overline{\mathcal{K}}_k))$, wobei k der Zerlegungstiefe entspricht.

1: $K \leftarrow ()$
2: $\mathcal{D} \leftarrow A(\bar{1}_{con})$
3: **while** true **do**
4: $\quad\widehat{\mathcal{L}} \leftarrow \mathcal{R} \circ \mathcal{D}$
5: $\quad\overline{\mathcal{L}} \leftarrow \mathcal{R} \circ Support(\mathcal{D})$
6: $\quad$**if** $\Sigma^* \{ \textcircled{G}, \textcircled{M} \} \Sigma^* \cap Dom(\widehat{\mathcal{L}}) = A(\varnothing)$ **then**
7: $\quad\quad K \leftarrow K + (\widehat{\mathcal{L}}, Range(\overline{\mathcal{L}}))$
8: $\quad\quad$**break**
9: $\quad$**else**
10: $\quad\quad K \leftarrow K + (\widehat{\mathcal{L}}, Range(\overline{\mathcal{L}}))$
11: $\quad\quad \mathcal{D} \leftarrow Dom(\widehat{\mathcal{L}})$
12: $\quad$**end if**
13: **end while**
14: **return** K

memoisierte Kaskade aufgebaut. Über die Zeilen 4 und 5 werden die beiden zusammengefassten simultanen lokalen Konsistenzregeln zu einer Ebene berechnet. In Zeile 10 wird anhand dieser beiden die memoisierte Kaskade erweitert und in Zeile 11 wird das 'Vergessen' umgesetzt. Wenn der Algorithmus eine memoisierte Kaskade der Länge k zurückgibt, kann bei einem Constraint-Problem $(C_{\mathcal{R}^\infty}, con)$ auf das konkretere Constraint-Problem $(C_{\mathcal{R}^k}, con)$ geschlossen werden.

Nun lässt sich der Algorithmus 6 definieren, der anhand der memoisierten Kaskade top-down die abstrakte Lösung berechnet. Innerhalb der For-Schleife

---

Algorithmus 6 Berechnung der abstrakten Lösung eines Constraint-Problems anhand einer memoisierten Kaskade.

Eingabe: Eine memoisierte Kaskade $K = ((\widehat{\mathcal{K}}_1, \overline{\mathcal{K}}_1),(\widehat{\mathcal{K}}_2, \overline{\mathcal{K}}_2),\ldots,(\widehat{\mathcal{K}}_k, \overline{\mathcal{K}}_k))$ zu einem Constraint-Problem $P = (C_{\mathcal{R}^k}, con)$ und der Wert $\mathcal{T}$ des Top-Constraints.

Ausgabe: Der Constraint für die abstrakte Lösung von P.

1: $\mathcal{L}_1 \leftarrow Support(Best(\mathcal{T} \circ \widehat{\mathcal{K}}_k))$
2: $\mathcal{L}_2 \leftarrow Range(\mathcal{L}_1 \circ \overline{\mathcal{K}}_k)$
3: **for** $i \leftarrow k-1$ **downto** 1 **do**
4: $\quad \mathcal{L}_1 \leftarrow Support(Best(Range(\mathcal{L}_1) \circ \widehat{\mathcal{K}}_i))$
5: $\quad \mathcal{L}_2 \leftarrow Range(\mathcal{L}_2 \circ \mathcal{L}_1 \circ \overline{\mathcal{K}}_i)$
6: **end for**
7: **return** $(\mathcal{L}_2, A(con))$

---

in den Zeilen 3 bis 6 werden sukzessive die Teile der memoisierten Kaskade top-down zur abstrakten Lösung verrechnet. In Zeile 1 und 4 werden die Zeichenketten mit dem besten Gewicht ermittelt. Hierdurch sind in Zeile 2 und 5 die Wertetupel ausgeschlossen, die nicht Teil der abstrakten Lösung sind.

4.5.8 Aufspalten simultaner lokaler Konsistenzregeln

Bei einer adäquaten Grammatik kann eine simultane lokale Konsistenzregel eine sehr hohe Übergangs- und Zustandsanzahl aufweisen, obwohl im Vergleich zu einer FSIG ein kleinerer Beschränkungsbereich angenommen ist. Dies kann beispielsweise dann auftreten, wenn viele verletzbare Beschränkungen involviert sind, da diese zu einer starken Kombinatorik führen können.

In diesem Abschnitt wird einem Ansatz nachgegangen, bei dem eine simultane lokale Konsistenzregel über mehrere Regelbestandteile repräsentiert ist, die separat Beschränkungen der Grammatik umsetzen. So müssen die Constraints nicht initial in einen monolithischen WFST kompiliert werden. Um den Ansatz darzulegen, wird hier der Fall der Optimalzerlegung betrachtet. Der Fall einer Nicht-Optimalzerlegung ist aber analog zu verstehen.

Es wird angenommen, dass die Beschränkungen der Grammatik über mehrere Constraint-Werte umgesetzt sind, sodass mit einem bestimmten Constraint-Typ mehrere Constraint-Werte verknüpft sind. Die entsprechenden Constraint-Werte müssen innerhalb eines Constraint-Problems zusammengefasst sein. Dies geschieht hier während des Problemlösens. Dies ist über die Assoziativität der Kompositionsoperation möglich. So kann der Suchraum bereits eingeschränkt werden, bevor die Werte zu einem Constraint-Typ zusammengefasst werden. Wenn die Beschränkungen der Grammatik geschickt auf mehrere Constraint-Werte verteilt sind und wenn anhand dieser eine simultane lokale Konsistenzregel erstellt wird, die aus mehreren separaten Bestandteilen besteht, kann eine unhandhabbare monolithische simultane lokale Konsistenzregel vermieden werden.

Es sei angenommen, dass die Beschränkungen der Grammatik durch die Constraint-Werte $\mathcal{Q}_1, \mathcal{Q}_2, \ldots, \mathcal{Q}_n$ abgebildet werden. Des Weiteren seien die Filter $\mathcal{F}_u$ und $\mathcal{F}_l$ aus Abschnitt 4.5.6 gegeben. Dann kann eine simultane lokale Konsistenzregel wie folgt definiert werden:

$$\mathcal{R} = \left(\begin{array}{c} \epsilon \leftarrow \mathcal{U} \\ \circ \\ \mathcal{F}_u \\ \circ \\ (\mathcal{I}^* \spadesuit \mathcal{Q}_1 \spadesuit)^* \mathcal{I}^* \\ \circ \\ (\mathcal{I}^* \spadesuit \mathcal{Q}_2 \spadesuit)^* \mathcal{I}^* \\ \circ \\ \vdots \\ \circ \\ (\mathcal{I}^* \spadesuit \mathcal{Q}_n \spadesuit)^* \mathcal{I}^* \\ \circ \\ \mathcal{F}_l \end{array} \right) \tag{4.35}$$

Im Folgenden soll eine simultane lokale Konsistenzregel in mehrere Teile aufgespalten werden. Dieses Vorgehen orientiert sich an der Aufspaltung des Constraint-Werts $\mathcal{Q}$ in $\mathcal{Q}_1, \mathcal{Q}_2, \ldots, \mathcal{Q}_n$ mit $n > 1$. Hierbei ist es immer möglich, einen Constraint-Wert einzubeziehen, der keine Beschränkungen umsetzt und als Dummy-Constraint-Wert fungiert. Gegeben die Constraint-Werte $\mathcal{Q}_1 \circ \mathcal{Q}_2 \circ \cdots \circ \mathcal{Q}_n$ mit $n > 1$. Dann ist für $\mathcal{B}_1, \mathcal{B}_2, \ldots, \mathcal{B}_n$ der Regelteil $\mathcal{B}_1$ definiert durch:

$$\mathcal{B}_1 = \left(\begin{array}{c} \epsilon \leftarrow\!\!\!\shortmid \mathcal{U} \\ \circ \\ \mathcal{F}_u \\ \circ \\ (\mathcal{I}^* \spadesuit \mathcal{Q}_1 \spadesuit)^* \mathcal{I}^* \end{array} \right) \tag{4.36}$$

und der Regelteil $\mathcal{B}_n$ durch:

$$\mathcal{B}_n = \left(\begin{array}{c} (\mathcal{I}^* \spadesuit \mathcal{Q}_n \spadesuit)^* \mathcal{I}^* \\ \circ \\ \mathcal{F}_l \end{array} \right) \tag{4.37}$$

und der Regelteil $\mathcal{B}_k$ mit $1 < k > n$ durch:

$$\mathcal{B}_k = \left(\begin{array}{c} (\mathcal{I}^* \spadesuit \mathcal{Q}_k \spadesuit)^* \mathcal{I}^* \end{array} \right) \tag{4.38}$$

sodass $\mathcal{B}_1 \circ \mathcal{B}_2 \circ \cdots \circ \mathcal{B}_n = \mathcal{R}$.

Anhand dieser Regelbestandteile lässt sich der Bottom-Up-Lösungsalgorithmus 7 definieren, der eine Optimalzerlegung behandelt und der als eine Anpassung des Algorithmus 4 aus Abschnitt 4.5.6 anzusehen ist. Hier wird zunächst der Fall betrachtet, bei dem keine Memoisierung einbezogen ist. Innerhalb der Repeat-Schleife in den Zeilen 2 bis 6 werden die Bestandteile nacheinander angewendet. Die Reihenfolge der Anwendungen kann hierbei einen Einfluss auf die Größe der Zwischenergebnisse haben. Dies verhält sich ähnlich zu den Constraint-Anwendungen einer FSIG.

Das Aufspalten simultaner lokaler Konsistenzregeln kann mit der Memoisation von Zwischenergebnissen kombiniert werden. Hierzu ist der Algorithmus 8 definiert, der eine memoisierte Kaskade unter Berücksichtigung der separaten Bestandteile einer simultanen lokalen Konsistenzregel berechnet. Anhand dieser Kaskade kann dann wie in Abschnitt 4.5.7 beschrieben über den Algorithmus 6 die Lösung ermittelt werden. Innerhalb der While-Schleife

Algorithmus 7 Simultane perfekte Relaxierung mit Bottom-Up-Lösen, Optimalzerlegung und ungebundener Tiefe und einer Sequenz von Regelbestandteilen.

Eingabe: Eine Sequenz von Bestandteilen $S = (\mathcal{B}_1, \mathcal{B}_2, \ldots, \mathcal{B}_n)$ mit $|S| > 1$ zu einer simultanen lokalen Konsistenzregel $\mathcal{R}$ für das Constraint-Problem $P = (C_{\mathcal{R}^\infty}, con)$, die eine Optimalzerlegung implementiert. Der Wert $\mathcal{T}$ des Top-Constraints und der Bottom-Constraint $(A(\bar{1}_{con}), A(con))$.

Ausgabe: Der Lösungs-Constraint von P.

1: $\mathcal{L} \leftarrow A(\bar{1}_{con})$
2: **repeat**
3: **for** $i \leftarrow n$ **downto** 1 **do**
4: $\mathcal{L} \leftarrow \mathcal{B}_i \circ \mathcal{L}$
5: **end for**
6: **until** $\Sigma^* \{ⓖ, ⓜ\} \Sigma^* \cap Dom(\mathcal{L}) = A(\varnothing)$
7: $\mathcal{L} \leftarrow \mathcal{T} \circ \mathcal{L}$
8: **return** $(Range(\mathcal{L}), A(con))$

wird das Anwenden der Regelbestandteile in den Zeilen 4 bis 9 umgesetzt. In Zeile 11 und 14 wird hier im Gegensatz zu Algorithmus 5 aus Abschnitt 4.5.7 nicht der Wertebereich von $\overline{\mathcal{L}}$ ermittelt. Daher sind die Kaskadenbestandteile $\overline{\mathcal{K}}_1, \overline{\mathcal{K}}_2, \ldots, \overline{\mathcal{K}}_n$ nun Transduktoren. So ist dort das Trennzeichen ♠ immer noch enthalten. Aus diesem Grund besteht weiterhin der Zugriff über die Position der Dependenzknoten-Teilprobleme. Dieser Sachverhalt ist im folgenden Abschnitt für die verzögerte Anwendung von Constraints relevant.

4.5.9 Verzögertes Anwenden von Constraints

Bei der Berechnung der Problemzerlegungen werden bisher sowohl harte als auch weiche Constraints einbezogen. Hierbei sind die weichen für die Problemzerlegungsberechnung aber nicht notwendig. In diesem Abschnitt liegt die grundlegende Idee darin, erst die Problemzerlegungen anhand der harten Constraints zu berechnen und danach die weichen verzögert auf Basis der Zerlegung anzuwenden. Für das verzögerte Anwenden wird vorausgesetzt, dass eine memoisierte Kaskade vorliegt, die über den Algorithmus 8 aus Abschnitt 4.5.8 ermittelt wurde. So sind die Positionen der einzelnen Dependenzknoten-Teilprobleme immer noch zugänglich.

Algorithmus 8 Berechnung einer memoisierten Kaskade über eine Optimalzerlegung und einer Sequenz von Regelbestandteilen.

Eingabe: Eine Sequenz von Bestandteilen $S = (\mathcal{B}_1, \mathcal{B}_2, \ldots, \mathcal{B}_n)$ mit $|S| > 1$ zu einer simultanen lokalen Konsistenzregel $\mathcal{R}$ für das Constraint-Problem $P = (C_{\mathcal{R}^\infty}, con)$, die eine Optimalzerlegung implementiert und der Bottom-Constraint $(A(\bar{1}_{con}), A(con))$.

Ausgabe: Eine memoisierte Kaskade $K = ((\widehat{\mathcal{K}}_1, \overline{\mathcal{K}}_1), (\widehat{\mathcal{K}}_2, \overline{\mathcal{K}}_2), \ldots, (\widehat{\mathcal{K}}_k, \overline{\mathcal{K}}_k))$, wobei k der Zerlegungstiefe entspricht.

$K \leftarrow ()$
$\mathcal{D} \leftarrow A(\bar{1}_{con})$
while true **do**
  $\overline{\mathcal{L}} \leftarrow Support(\mathcal{D})$
  for $i \leftarrow n$ **downto** 2 **do**
    $\overline{\mathcal{L}} \leftarrow \mathcal{B}_i \circ \overline{\mathcal{L}}$
  end for
  $\widehat{\mathcal{L}} \leftarrow \mathcal{B}_1 \circ Dom(\overline{\mathcal{L}} \circ \mathcal{D})$
  $\overline{\mathcal{L}} \leftarrow Range(\mathcal{B}_1) \circ \overline{\mathcal{L}}$
  if $\Sigma^* \{ \textcircled{G}, \textcircled{M} \} \Sigma^* \cap Dom(\widehat{\mathcal{L}}) = \varnothing$ **then**
    $K \leftarrow K + (\widehat{\mathcal{L}}, \overline{\mathcal{L}})$
    break
  else
    $K \leftarrow K + (\widehat{\mathcal{L}}, \overline{\mathcal{L}})$
    $\mathcal{D} \leftarrow Dom(\widehat{\mathcal{L}})$
  end if
end while
return K

Wenn ausschließlich die abstrakte Lösung berechnet werden soll und wenn bezüglich des c-Semirings verschiedene streng voneinander getrennte Bewertungsbereiche angenommen werden, können weiche Constraints nach einer bestimmten Relevanzordnung in Blöcken nacheinander angewendet werden. Nach den Anwendungen kann dann zwischendurch die abstrakte Lösung ermittelt werden, um das Anwachsen von Zwischenergebnissen abzumildern.

Im Folgenden wird zuerst auf den lexikografischen n-dimensionalen Semiring eingegangen, der es ermöglicht, streng voneinander getrennte Bewertungsbereiche mit einer bestimmten Relevanzordnung zu definieren. Darauf aufbauend wird gezeigt, wie mithilfe der Ordnung sukzessive anhand von Constraint-Mengen (abstrakte) Lösungen berechnet werden können. Abschließend wird dargelegt, wie diese Technik in den Problemzerlegungsansatz integriert werden kann.

Lexikografischer n-dimensionaler Semiring

In Didakowski (2008a) wird im Kontext des FSIG-Parsings mit Gewichten eine komplexe Gewichtsstruktur in Form eines lexikografischen n-dimensionalen Semirings vorgeschlagen, um verschiedene (linguistische) Kriterien zu implementieren, die nach einer bestimmten Relevanz geordnet sind. Die einzelnen Kriterien sind separat über linear geordnete Semiringe implementiert und zu einem n-dimensionalen Semiring zusammengefasst, der die Relevanzordnung einbezieht. Auf diese Weise herrscht eine strikte Trennung zwischen den verschiedenen Bewertungsbereichen, die über die Kriterien widergespiegelt werden. In dieser Arbeit soll der lexikografische n-dimensionale Semiring herangezogen werden, um weiche Constraints mit getrennten Bewertungsbereichen konzeptuell zu erarbeiten. Mengen von Constraints sind dann mit bestimmten linear geordneten Kriterien und somit mit einer Relevanzordnung verknüpft.

Gegeben die linear geordneten c-Semiringe $S_1 = (A_1, \oplus_1, \otimes_1, \bar{0}_1, \bar{1}_1)$,$S_2 = (A_2, \oplus_2, \otimes_2, \bar{0}_2, \bar{1}_2)$,...,$S_n = (A_n, \oplus_n, \otimes_n, \bar{0}_n, \bar{1}_n)$ und die folgende Relevanzordnung: $S_1 \succ S_2 \succ \cdots \succ S_n$. Dann ergibt die Semiring-Komposition $S_1 \circ S_2 \circ \cdots \circ S_n$ einen neuen c-Semiring der Form $S = ((A_1, A_2, \ldots, A_n), \oplus, \otimes, (\bar{0}_1, \bar{0}_2, \ldots, \bar{0}_n), (\bar{1}_1, \bar{1}_2, \ldots, \bar{1}_n))$. Es seien zwei beliebige Elemente $(a_1, a_2, \ldots, a_n) \in S$ und $(b_1, b_2, \ldots, b_n) \in S$ angenommen, dann ist die abstrakte Multiplikation von S

definiert durch:

$$(a_1, a_2, \ldots, a_n) \otimes (b_1, b_2, ..., b_n) = (a_1 \otimes_1 b_1, a_2 \otimes_2 b_2, \ldots, a_n \otimes_n b_n) \quad (4.39)$$

Die Trägermenge entspricht der Vektorisierung der einzelnen individuellen Trägermengen und die abstrakte Multiplikation der Vektorisierung der individuellen abstrakten Multiplikationsoperationen. Dies folgt so weit der Definition eines n-dimensionalen c-Semirings (siehe Bistarelli et al., 1997, Abschnitt 7).

Die abstrakte Addition, die den lexikografischen n-dimensionalen Semiring ausmacht, ist folgend definiert. Gegeben zwei beliebige c-Semiring-Elemente $(a_1, a_2, ..., a_n) \in S$ and $(b_1, b_2, ..., b_n) \in S$, dann ist die abstrakte Addition von S definiert durch (für den binären Fall siehe auch Sproat et al., 2014):

$$(a_1, a_2, ..., a_n) \oplus (b_1, b_2, ..., b_n) = \begin{cases} (a_1, a_2, ..., a_n) & \text{wenn } (a_1, a_2, ..., a_n) = (b_1, b_2, ..., b_n) \\ (a_1, a_2, ..., a_n) & \text{wenn } a_1 = b_1 \text{ und } a_2 = b_2 \text{ und } \ldots \text{ und } a_{k-1} = b_{k-1} \text{ und} \\ & a_k \oplus_k b_k = a_k \text{ mit } k \leq n \text{ und } a_k \neq b_k \\ (b_1, b_2, ..., b_n) & \text{sonst} \end{cases} \quad (4.40)$$

Wenn die einzelnen zugrunde liegenden c-Semiringe linear geordnet sind, dann gilt dies auch für den lexikografischen n-dimensionalen Semiring. In Seeker und Quernheim (2008) wird im Bereich FSMs im Kontext der Optimalitätstheorie ein *tropischer Vektorsemiring* (tropical vector semiring) vorgeschlagen, um ähnlich zur Relevanzordnung von Kriterien ein Ranking von Constraints umzusetzen.

Sukzessive Berechnung der abstrakten Lösung

In diesem Abschnitt wird vorausgesetzt, dass bestimmte Bewertungsbereiche existieren, die strikt voneinander getrennt sind und die über Kriterien als lexikografischer n-dimensionaler c-Semiring umgesetzt sind. Im Folgenden seinen sowohl die c-Semiringe $S_1 = (A_1, \oplus_1, \otimes_1, \bar{0}_1, \bar{1}_1)$,$S_2 = (A_2, \oplus_2, \otimes_2, \bar{0}_2, \bar{1}_2)$,...,$S_n = (A_n, \oplus_n, \otimes_n, \bar{0}_n, \bar{1}_n)$, die die einzelnen Kriterien implementieren, als auch die Kriterienordnung $S_1 \succ S_2 \succ \cdots \succ S_n$ gegeben. Zudem sei angenommen, dass die c-Semiringe zu einem lexikografischen n-dimensionalen

Semiring zusammengefasst sind und damit einer Relevanzordnung unterliegen.

Außerdem wird angenommen, dass sich ein weicher Constraint auf genau ein gegebenes Kriterium bezieht und dass er im Rahmen des lexikografischen n-dimensionalen c-Semirings ausschließlich innerhalb der für ihn relevanten Komponente eine Bewertung vergibt, die ungleich dem Nullelement ist. Innerhalb der anderen Komponenten weist er hingegen das Einselement zu. Wenn ein beliebiges c-Semiring-Element $(a_1, a_2, ..., a_n)$ gegeben ist, das von einem Constraint vergeben wird, der sich auf die Komponente k und somit auf das k-te Kriterium bezieht mit $0 \leq k \leq n$. Dann gilt für alle a_r mit $0 \leq r < k$ oder $k < r \leq n$, dass $a_r = \bar{1}_r$ und für a_r, dass $a_k \neq \bar{0}$. So kann die Relevanzordnung der Kriterien ausgenutzt werden, um die Zwischenergebnisse während des Problemlösens klein zu halten. Die Idee dahinter ist, erst die abstrakte Lösung bezüglich des relevantesten Kriteriums, dann hinsichtlich des zweitrelevantesten usw. zu berechnen.

Für jedes Kriterium sei eine Constraint-Menge gegeben: bezüglich S_1 eine Constraint-Menge M_1, hinsichtlich S_2 eine Constraint-Menge M_2 usw. Für diese Mengen herrscht die gleiche Relevanzordnung wie für die Kriterien: $M_1 \succ M_2 \succ \cdots \succ M_n$. Anstatt die Lösung über $Sol((M_1 \cup M_2 \cup \cdots \cup M_n, con))$ zu berechnen, kann sie nun über $Sol((M_n \cup Sol((M_{n-1} \cup \ldots Sol((M_1, con)) \ldots, con)), con))$ ermittelt werden. Es wird erst die Lösung anhand von M_1 berechnet. Dann werden die Constraints von M_2 hinzugezogen und es wird erneut die Lösung ermittelt usw. Das Problem ist also in mehrere Teilprobleme aufgespalten. Gleiches lässt sich auf die Ermittlung der abstrakten Lösung übertragen. Anstatt diese über $ASol((M_1 \cup M_2 \cup \cdots \cup M_n, con))$ zu berechnen, kann sie über $ASol((M_n \cup ASol((M_{n-1} \cup \ldots ASol((M_1, con)) \ldots, con)), con))$ ermittelt werden. Dies ist möglich, da über die Kriterien die einzelnen Beschränkungsbereiche geordnet und strikt voneinander getrennt sind. So wird über das sukzessive Berechnen der abstrakten Lösungen der lexikografische Charakter der abstrakten Addition nachempfunden. Dieses Vorgehen wird im Folgenden als *verzögerte Anwendung von Constraints* bezeichnet.

Wenn zusätzlich harte Constraints verzögert angewendet werden sollen, können für diese eigene harte Constraint-Mengen angenommen werden. Diese sind in der Relevanzordnung stets vor den weichen Constraint-Mengen $M_1, M_2, \ldots, M_n$, da harte Constraints bezüglich des c-Semirings die schlech-

teste Bewertung vergeben. Somit sind sie als am relevantesten einzustufen. Es seien die harten Constraint-Mengen $M_{0_1}, M_{0_2}, \ldots M_{0_m}$ gegeben und es sei eine beliebige Relevanzordnung über diese Mengen definiert $M_{0_1} \succ M_{0_2} \succ \cdots \succ M_{0_m}$. Dann ergibt sich mit Hinzunahme der weichen Constraint-Mengen die Relevanzordnung $M_{0_1} \succ M_{0_2} \succ \cdots \succ M_{0_m} \succ M_1 \succ M_2 \succ \cdots \succ M_n$ anhand derer sukzessive die abstrakte Lösung berechnet werden kann. Hierbei ist die Berechnung der abstrakten Lösung über harte Constraints gleichbedeutend mit der Berechnung der Lösung. Hierbei ist interessant, dass auch ohne einen lexikografischen n-dimensionalen Semiring eine Relevanzordnung zwischen harten und weichen Constraints besteht. So kann erst die (abstrakte) Lösung bezüglich der harten Constraints berechnet werden und daraufhin die abstrakte Lösung mit Hinzunahme der weichen Constraints. Zudem ist die Reihenfolge, in der die harten Constraint-Mengen angewendet werden, nicht lösungsrelevant. Um die Anzahl an Constraint-Mengen zu reduzieren, können eine oder mehrere harte Constraint-Mengen mit der adjazenten weichen Constraint-Menge zusammengefasst werden. So können beispielsweise alle harten Constraint-Mengen und die adjazente weiche Constraint-Menge M_1 vereinigt werden: $(M_{0_1} \cup M_{0_2} \cup \cdots \cup M_{0_m} \cup M_1) \succ M_2 \succ \cdots \succ M_n$.

Den Algorithmus 8 aus Abschnitt4.5.8 vorausgesetzt, besteht innerhalb einer Problemzerlegung über die memoisierten Zwischenergebnisse ein direkter Zugriff auf die Positionen und die relativen Positionsinformationen der einzelnen Dependenzknoten-Teilprobleme. So können Constraints nach der Berechnung der Zerlegung gezielt auf diese Teilprobleme angewendet werden. Zudem lässt sich anhand einer memoisierten Kaskade für jede Zerlegungsebene die abstrakte Lösung ermitteln. Diese wird hierbei über alle Constraint-Systeme und Probleme hinweg berechnet, die simultan repräsentiert sind (vgl. Abschnitt 3.4.2).

Für eine memoisierte Kaskade ist der Algorithmus 9 gegeben, der es ermöglicht, für jede Kaskadenebene die abstrakte Lösung zu ermitteln und das Ergebnis als neue eingeschränktere memoisierte Kaskade zurückzugeben. Innerhalb der For-Schleife in den Zeilen 6 bis 11 werden die abstrakten Lösungen top-down propagiert. Über die Zeilen 2 und 7 wird die abstrakte Lösung zu einer Ebene ermittelt. Mithilfe der Zeilen 4 und 9 wird die neue Kaskade aufgebaut. In Zeile 5 und 10 wird die Gewichtsinformation entfernt, da hier im Gegensatz zu Algorithmus 5 aus Abschnitt 4.5.7 nicht das Gewicht eines Lösungs-Constraint berechnet werden soll, weil in dieser Arbeit auf die Ver-

Algorithmus 9 Berechnung der abstrakten Lösung für die Ebenen einer memoisierten Kaskade.

Eingabe: Eine memoisierte Kaskade $K = ((\widehat{\mathcal{K}}_1, \overline{\mathcal{K}}_1),(\widehat{\mathcal{K}}_2, \overline{\mathcal{K}}_2),\ldots,(\widehat{\mathcal{K}}_k, \overline{\mathcal{K}}_k))$ zu einem Constraint-Problem $P = (C_{\mathcal{R}^k}, con)$ und der Wert $\mathcal{T}$ des Top-Constraints.

Ausgabe: Eine memoisierte Kaskade der Länge k.

1: $K' = ()$
2: $\widehat{\mathcal{L}} \leftarrow Support(Best(\mathcal{T} \circ \widehat{\mathcal{K}}_k))$
3: $\overline{\mathcal{L}} \leftarrow Range(\widehat{\mathcal{L}}) \circ \overline{\mathcal{K}}_k$
4: $K' = K' + (\widehat{\mathcal{L}}, \overline{\mathcal{L}})$
5: $\mathcal{D} \leftarrow Support(Range(\overline{\mathcal{L}}))$
6: **for** $i \leftarrow k-1$ **downto** 1 **do**
7: $\quad \widehat{\mathcal{L}} \leftarrow Support(Best(\mathcal{D} \circ \widehat{\mathcal{K}}_i))$
8: $\quad \overline{\mathcal{L}} \leftarrow Range(\widehat{\mathcal{L}}) \circ \overline{\mathcal{K}}_i$
9: $\quad K' = K' + (\widehat{\mathcal{L}}, \overline{\mathcal{L}})$
10: $\quad \mathcal{D} \leftarrow Support(Range(\overline{\mathcal{L}}))$
11: **end for**
12: **return** $reverse(K')$

wendung eines lexikografischen n-dimensionalen Semirings verzichtet wird. In Zeile 12 wird die aufgebaute Kaskade abschließend umgedreht, da diese in umgekehrter Reihenfolge aufgebaut wurde.

Nun soll ein Algorithmus vorgestellt werden, der es ermöglicht, Constraints verzögert auf die Positionen bereits gelöster Dependenzknoten-Teilprobleme anzuwenden. Dies wird über die Regelbestandteile einer simultanen lokalen Konsistenzregel möglich. Das bedeutet, dass hier initial nicht alle Bestandteile bei der Berechnung der Zerlegung einbezogen werden (vgl. Abschnitt 4.5.8). Wenn $\mathcal{Q}_k$ ein Constraint ist, der verzögert angewendet werden soll, dann hat ein Regelbestandteil $\mathcal{R}_k$ die Form:

$$\mathcal{R}_k = \Big(\ (\mathcal{I}^* \ \spadesuit \ \mathcal{Q}_k \ \spadesuit \ )^* \ \mathcal{I}^* \ \Big) \tag{4.41}$$

Dies ist analog zu dem Ausdruck eines mittleren Regelbestandteils in 4.38 aus Abschnitt 4.5.8. Nun kann der Algorithmus 10 definiert werden, der Constraints verzögert anwendet und eine neue eingeschränktere memoisierte Kaskade berechnet. Über die For-Schleife in den Zeilen 8 bis 15 wird die memoi-

Algorithmus 10 Verzögerte Anwendung von Constraints auf eine memoisierte Kaskade.

Eingabe: Eine memoisierte Kaskade $K = ((\widehat{\mathcal{K}}_1, \overline{\mathcal{K}}_1), (\widehat{\mathcal{K}}_2, \overline{\mathcal{K}}_2), \ldots, (\widehat{\mathcal{K}}_k, \overline{\mathcal{K}}_k))$ zu einem Constraint-Problem $P = (C_{\mathcal{R}^k}, con)$. Eine nachgeschaltete Sequenz von Bestandteilen $S = (\mathcal{R}_1, \mathcal{R}_2, \ldots, \mathcal{R}_n)$ einer simultanen lokalen Konsistenzregel $\mathcal{R}$ für das Constraint-Problem $P = (C_{\mathcal{R}^\infty}, con)$, die eine Optimalzerlegung implementiert.

Ausgabe: Eine memoisierte Kaskade der Länge k.

1: $K' \leftarrow ()$
2: $\overline{\mathcal{L}} \leftarrow \overline{\mathcal{K}}_1$
3: **for** $i \leftarrow 1$ **to** n **do**
4: $\quad \overline{\mathcal{L}} \leftarrow \mathcal{R}_i \circ \overline{\mathcal{L}}$
5: **end for**
6: $\widehat{\mathcal{L}} \leftarrow \widehat{\mathcal{K}}_1 \circ Dom(\overline{\mathcal{L}})$
7: $K' \leftarrow K' + (\widehat{\mathcal{L}}, \overline{\mathcal{L}})$
8: **for** $i \leftarrow 2$ **to** k **do**
9: $\quad \overline{\mathcal{L}} \leftarrow \overline{\mathcal{K}}_i$
10: $\quad$ **for** $j \leftarrow 1$ **to** n **do**
11: $\quad\quad \overline{\mathcal{L}} \leftarrow \mathcal{R}_j \circ \overline{\mathcal{L}}$
12: $\quad$ **end for**
13: $\quad \widehat{\mathcal{L}} \leftarrow \widehat{\mathcal{K}}_i \circ Dom(\overline{\mathcal{L}} \circ \widehat{\mathcal{L}})$
14: $\quad K' \leftarrow K' + (\widehat{\mathcal{L}}, \overline{\mathcal{L}})$
15: **end for**
16: **return** K'

sierte Kaskade bottom-up durchlaufen und Es werden analog zu Algorithmus 5 in Abschnitt 4.5.7 Gewichtungen sukzessive nach oben weitergereicht. Innerhalb der For-Schleife in den Zeilen 3 bis 5 und der in den Zeilen 10 bis 12 werden die Regelbestandteile angewendet. In den Zeilen 6 und 13 werden die Kaskadenbestandteile $\widehat{\mathcal{K}}_1, \widehat{\mathcal{K}}_2, \ldots, \widehat{\mathcal{K}}_n$ in Bezug zu ihrer Gewichtsmasse aktualisiert, um später die abstrakten Lösungen der Zerlegungsebenen oder die abstrakte Lösung des Problems berechnen zu können. Über die Zeilen 7 und 14 wird letztlich die neue memoisierte Kaskade aufgebaut.

Über die Algorithmen 9 und 10 lässt sich wie folgt ein verzögertes Anwenden von Constraint-Mengen und eine sukzessive Berechnung der abstrakten Lösung umsetzen. Initial wird auf Grundlage der relevantesten ersten Constraint-Menge über Algorithmus 5 aus Abschnitt 4.5.8 eine Problemzerlegung und somit eine memoisierte Kaskade berechnet. Daraufhin wird über Algorithmus 9 die abstrakte Lösung für die einzelnen Kaskadenebenen ermittelt. Ausgehend hiervon geschieht die verzögerte Anwendung der übrigen Constraint-Mengen unter Berücksichtigung der Relevanzordnung jeweils auf die folgende Weise: Als Erstes werden die Constraints der darauffolgenden Constraint-Menge über Algorithmus 10 auf die Positionen der Dependenzknoten-Teilprobleme angewendet. Danach wird über Algorithmus 9 erneut die abstrakte Lösung für die einzelnen Kaskadenebenen berechnet. Ausgehend von der resultierenden memoisierten Kaskade wird die nächste Constraint-Menge herangezogen und auf die Positionen der Dependenzknoten-Teilprobleme angewendet. Nachdem auch hier die abstrakte Lösung für die einzelnen Kaskadenebenen ermittelt wurde, wird die darauffolgende Constraint-Menge einbezogen usw. Im Fall der letzten Constraint-Menge muss die abstrakte Lösung für die Ebenen nicht berechnet werden. Hier kann wie in Abschnitt 4.5.7 beschrieben über Algorithmus 6 final die abstrakte Lösung des Problems ermittelt werden.

4.5.10 Affixorientierte Zerlegung

In Abschnitt 4.5.3 wurde gezeigt, dass mit einer festen Zerlegungstiefe Parsing in linearer Zeit möglich ist. Die natürliche Sprache scheint einige Eigenschaften zu besitzen, die für solche Beschränkungen sprechen. Karlsson (2010) zeigt, dass ein absolutes Limit für Zentraleinbettungen existiert, sowohl in geschriebener als auch gesprochener Sprache. Hierzu betrachtet er

unterschiedliche Typen von Rekursionen und untersucht diese empirisch in verschiedenen Sprachen. Yli-Jyrä (2003) zeigt für das Dänische, dass dort die *eingebettete Kreuzungstiefe* (nested crossing depth) generell sehr gering ist. Hierzu untersucht er Konstruktionen empirisch auf Basis einer dänischen Baumbank.

Dies gibt Motivation für eine Parametrisierung der Komplexität der Dependenzstrukturen: z. B. die maximale Tiefe eines Dependenzbaums oder eine maximale eingebettete Kreuzungstiefe (im Kontext parametrisierter geklammerter Dependenzbäume siehe auch Yli-Jyrä, 2004). Des Weiteren kann innerhalb der Grammatik rechtsseitige oder linksseitige Rekursion durch Iteration ersetzt werden, um die Tiefe der Dependenzbäume generell zu verringern (im Kontext der FSIG vgl. Koskenniemi, 1990, Abschnitt „Word and clause boundaries“). Letztlich wäre man aber damit beschäftigt, ein *hartnäckiges Constraint-Optimierungsproblem* (intractable constraint optimization problem) in ein *einfacheres Constraint-Optimierungsproblem* (tractable constraint optimization problem) umzuformulieren. So würde wieder die FSIG-Problematik unhandhabbarer Zwischenergebnisse auftreten, obwohl die Parsingkomplexität linear ist.

In diesem Abschnitt wird ein anderer Weg beschritten. Es wird ein Ansatz vorgestellt, bei dem die Zerlegungstiefe künstlich erhöht wird, um das starke Anwachsen von Zwischenergebnissen abzumildern. Hierbei besteht die Annahme, das ein geschicktes Erhöhen der Zerlegungstiefe eine kompaktere Repräsentation der einzelnen Zerlegungsebenen und dadurch eine Beschleunigung der Analyse ermöglicht. Um die Zerlegungstiefe zu erhöhen, wird der Idee des Präfixparsings in Foth et al. (2000a) gefolgt. Dort wird im Kontext der WCDG ein Ansatz vorgestellt, bei dem ein initiales Satzpräfix sukzessive erweitert und analysiert wird, bis es den gesamten Satz umfasst. Die Analysen der einzelnen Satzpräfixe werden dabei für das Ausschließen und Generieren von Analysehypothesen herangezogen. Für verschiedene Heuristiken zeigt sich dort insbesondere für längere Sätze eine Beschleunigung der Laufzeit. In dieser Arbeit sollen sowohl Satzpräfixe als auch Satzsuffixe betrachtet werden. So ist im Folgenden mit Affix entweder ein Präfix oder ein Suffix gemeint. Es wird ein Ansatz verfolgt, der das sukzessive Analysieren und Erweitern von Satzaffixen in das Parsing einbezieht. Jedoch soll hier die Beschleunigung nicht über das Beschneiden des Suchraums erreicht werden, sondern durch die kompaktere Repräsentation von Zwischenergebnissen.

Das grundlegende Vorgehen ist wie folgt: Es wird erst eine Zerlegung auf Grundlage des initialen Affixes berechnet. Daraufhin werden sukzessive neue Variablen einbezogen, indem das Affix erweitert wird. Auf Grundlage dieser neu hinzugekommenen Variablen werden sukzessive weitere Zerlegungen berechnet, bis das Affix den ganzen Satz umfasst. Ob eine Variable zu einem Affix gehört, wird durch eine entsprechende verknüpfte Information bestimmt. Über diese wird festgelegt, welche Variablen in den Dependenzknoten-Teilproblemen involviert sein dürfen. Dies bedeutet, dass stets nur die Variablen in Dependenzknoten-Teilprobleme eingebunden werden, die Teil eines Affixes sind. Alle anderen werden als Füller-Variablen weitergereicht. Somit sind initial alle Variablen involviert.

Um die Tiefe einer Zerlegung über das Affixparsing künstlich erweitern zu können, wird eine abgeschwächte Form der Optimalzerlegung betrachtet, die hier als *affixorientierte Zerlegung* bezeichnet wird. Diese ist wie folgt definiert:

Definition 34 (Affixorientierte Zerlegung)**.** *Angenommen ein Satzaffix wird entweder als Satzpräfix von links nach rechts oder als Satzsuffix von rechts nach links sukzessive erweitert. Dann gilt bei einer affixorientierten Zerlegung für ein gegebenes Affix β, welches die direkte Erweiterung eines Affixes α darstellt und um die Menge an Wörtern W erweitert wurde oder welches als initiales Affix die Menge an Wörtern W umfasst, dass*

(i) ein nichtunäres Dependenzknoten-Teilproblem stets eine Variable, die sich in einem Dependenzknoten-Teilproblem des direkt vorhergehenden Schicht-Teilproblems befindet, umfasst;

(ii) sich alle unären Dependenzknoten-Teilprobleme, die Wörter aus W enthalten, in dem ersten Schicht-Teilproblem befinden, falls es sich um das initiale Affix handelt;

(iii) sich alle unären Dependenzknoten-Teilprobleme, die Wörter aus W enthalten, in dem Schicht-Teilproblem befinden, das direkt auf das letzte Schicht-Teilproblem von α folgt.

Bei dieser Definition ist interessant, dass Dummy-Schichtteilprobleme auch innerhalb der Zerlegung auftreten können und zwar immer genau zwischen

den Zerlegungsebenen der einzelnen Affixe. Im Folgenden sind mit den Schichtteilproblemen zu einem Affix die Schichtteilprobleme gemeint, die sukzessive auf Grundlage des entsprechenden Affixes berechnet wurden.

Im Folgenden liegt bei der affixorientierten Zerlegung der Fokus auf Algorithmen, die die Memoisation von Zwischenergebnissen einbeziehen. Die Berechnung der Lösung ohne Memoisation kann aber analog umgesetzt werden. Um die Zerlegungstiefe unter der Berücksichtigung einer affixorientierten Zerlegung zu erhöhen, wird die relative Positionsinformation ❶ herangezogen. Diese besagt, dass eine entsprechende Variable aus Top-Down-Sicht in einer folgenden Zerlegungsebene in keinem Dependenzknoten-Teilproblem enthalten ist und dort somit als Füller-Variable fungiert (siehe Tabelle 4.1 in Abschnitt 4.5.1). Über dieses Markierungszeichen wird verhindert, dass bestimmte Variablen in Dependenzknoten-Teilprobleme eingebunden werden. Da über die simultanen lokalen Konsistenzregeln eine Top-Down-Lösungsabfolge implementiert ist, wird initial von einem Affix ausgegangen, dass alle Wörter des Satzes umfasst. Dieses wird dann sukzessive verkleinert. Dass heißt, dass im Folgenden im Gegensatz zur Definition 34 eine Top-Down-Perspektive eingenommen wird. Für das sukzessive Verkleinern kann beispielsweise der folgende Ausdruck herangezogen werden, der ein Präfix um ein Wort verkleinert:

$$\mathcal{A}_p = (\triangleleft \text{❶} \diamond L(D) \triangleright)^* \triangleleft \epsilon : \text{❶} \diamond L(D) \triangleright (\triangleleft ((\Gamma - \text{❶}) \cup \epsilon) \diamond L(D) \triangleright)^* \quad (4.42)$$

Für den Suffix-Fall würde dann folgender Ausdruck bestehen:

$$\mathcal{A}_s = (\triangleleft ((\Gamma - \text{❶}) \cup \epsilon) \diamond L(D) \triangleright)^* \triangleleft \epsilon : \text{❶} \diamond L(D) \triangleright (\triangleleft \text{❶} \diamond L(D) \triangleright)^* \quad (4.43)$$

Hierbei ist das Alphabet für die relativen Positionsinformationen wie in den vorherigen Abschnitten durch Γ gegeben. Der Teilausdruck $(\triangleleft ((\Gamma - \text{❶}) \cup \epsilon) \diamond L(D) \triangleright)^*$ meint, dass Variablen überlesen werden, die nicht die relative Position ❶ besitzen. Hier wäre aber auch ein Ausdruck möglich, der eine Verkürzung um zwei oder mehr Wörter umsetzt. Zudem könnte für die Verkürzung auf linguistische Informationen wie die Wortart Bezug genommen werden, falls solche zur Verfügung stehen. Letztlich kann über dieses sukzessive Verkleinern das initiale Affix auf die Länge null verkürzt werden. Um final die relative Positionsinformation ❶ zu entfernen, kann der folgende

Ausdruck herangezogen werden:

$$\overline{\mathcal{A}} = (\triangleleft \text{❶} : \epsilon \diamond L(D) \triangleright)^* \tag{4.44}$$

Durch diesen Ausdruck wird die relative Positionsinformation ❶ auf Epsilon umgeschrieben. Wenn $L(\bar{1}_{con})$ mit $|con| > 2$ der Wert des Bottom-Constraint ist, dann repräsentiert beispielsweise die rationale Sprache $Dom(\mathcal{A}_p \circ \mathcal{A}_p \circ \mathcal{A}_p \circ \overline{\mathcal{A}} \circ L(\bar{1}_{con}))$ ein Präfix der Länge 3.

Nun kann der Algorithmus 11 definiert werden, der bottom-up eine memoisierte Kaskade auf Basis einer affixorientierten Zerlegung erstellt, die Zerlegungstiefe künstlich erhöht und auf dem Algorithmus 8 aus Abschnitt 4.5.8 aufbaut. Innerhalb der While-Schleife in den Zeilen 3 bis 21 wird sukzessive das Affix erweitert und stückweise die Problemzerlegung und somit die memoisierte Kaskade berechnet. In Zeile 2 wird die relative Positionsinformation ❶ initial eingeführt. In Zeile 2 und 15 wird über den FST $\mathcal{A}$ das Affix erweitert. Inwieweit es sich hier um Präfix- oder Suffixparsing handelt und auf welche Weise eine Erweiterung umgesetzt ist, hängt von dem FST ab. In Zeile 11 wird geprüft, ob eine weitere Affixerweiterung möglich ist. Die maximale Tiefe der Zerlegung eines Affixes, wenn es in Isolation betrachtet wird, entspricht der Affixlänge. Im ungünstigsten Fall kommt zu dieser Zerlegungstiefe stets die maximale Zerlegungstiefe des vorherigen Affixes hinzu. Die künstliche Erhöhung der Problemzerlegungstiefe kann demnach erheblich sein.

Ausgehend hiervon kann mithilfe des Algorithmus 6 aus Abschnitt 4.5.7 anhand der memoisierten Kaskade die Lösung berechnet werden. Alternativ können aber auch, wie in Abschnitt 4.5.9 beschrieben, Constraints verzögert angewendet werden. Wenn jedoch Algorithmus 9 aus Abschnitt 4.5.9 zur Berechnung der abstrakten Lösung für die Ebenen einer memoisierten Kaskade herangezogen wird, kann es zu einem unerwünschten Effekt kommen. Dies betrifft die Variablen, die sich in einer Zerlegungsebene in keinem Affix befinden. Über den Algorithmus werden innerhalb der einzelnen Zerlegungsebenen für diese Variablen Belegungen berechnet. Hierdurch können die Zwischenergebnisse unnötig anwachsen. Es soll demnach eine Belegung dieser Variablen in den entsprechenden Zerlegungsebenen verhindert werden. Das grundlegende Problem liegt in der Propagierung der Constraints über die verschiedenen Ebenen hinweg. Für die Variablen, die nicht im Affix einer Zerlegungsebene

Algorithmus 11 Berechnung einer memoisierten Kaskade über eine affix-orientierte Zerlegung.

Eingabe: Eine Sequenz von Bestandteilen $S = (\mathcal{B}_1, \mathcal{B}_2, \ldots, \mathcal{B}_n)$ mit $|S| > 1$ zu einer simultanen lokalen Konsistenzregel $\mathcal{R}$ für das Constraint-Problem $P = (C_{\mathcal{R}^\infty}, con)$, die eine Optimalzerlegung implementiert und der Bottom-Constraint $(A(\bar{1}_{con}), A(con))$. Der FST A, der ein gegebenes Affix aus Bottom-Up-Sicht erweitert.

Ausgabe: Eine memoisierte Kaskade $K = ((\widehat{\mathcal{K}}_1, \overline{\mathcal{K}}_1), (\widehat{\mathcal{K}}_2, \overline{\mathcal{K}}_2), \ldots, (\widehat{\mathcal{K}}_k, \overline{\mathcal{K}}_k))$, wobei k der Zerlegungstiefe entspricht.

1: $K \leftarrow ()$
2: $\mathcal{D} \leftarrow \mathcal{A} \circ (\triangleleft \; \mathtt{I} : \epsilon \diamond L(D) \triangleright)^* \circ A(\bar{1}_{con})$
3: **while** true **do**
4: $\quad \overline{\mathcal{L}} \leftarrow Support(\mathcal{D})$
5: $\quad$ **for** $i \leftarrow n$ **downto** 2 **do**
6: $\quad\quad \overline{\mathcal{L}} \leftarrow \mathcal{B}_i \circ \overline{\mathcal{L}}$
7: $\quad$ **end for**
8: $\quad \widehat{\mathcal{L}} \leftarrow \mathcal{B}_1 \circ Dom(\overline{\mathcal{L}} \circ \mathcal{D})$
9: $\quad \overline{\mathcal{L}} \leftarrow Range(\mathcal{B}_1) \circ \overline{\mathcal{L}}$
10: $\quad$ **if** $\Sigma^* \; \{ \textcircled{G}, \textcircled{M} \} \; \Sigma^* \cap Dom(\widehat{\mathcal{L}}) = \varnothing$ **then**
11: $\quad\quad$ **if** $\Sigma^* \; \mathtt{I} \; \Sigma^* \cap Dom(\mathcal{L}) = \varnothing$ **then**
12: $\quad\quad\quad K \leftarrow K + (\widehat{\mathcal{L}}, \overline{\mathcal{L}})$
13: $\quad\quad\quad$ **break**
14: $\quad\quad$ **else**
15: $\quad\quad\quad \mathcal{D} \leftarrow \mathcal{A} \circ \mathcal{D}$
16: $\quad\quad$ **end if**
17: $\quad$ **else**
18: $\quad\quad K \leftarrow K + (\widehat{\mathcal{L}}, \overline{\mathcal{L}})$
19: $\quad\quad \mathcal{D} \leftarrow Dom(\widehat{\mathcal{L}})$
20: $\quad$ **end if**
21: **end while**
22: **return** K

vorhanden sind, soll im Folgenden die Propagierung aussetzen. Die Idee ist hierbei, innerhalb einer Ebene die Wertetupel auf die Variablen zu projizieren, die in dem aktuellen Affix enthalten sind. Daraufhin können die Tupel wieder auf die vorherige Variablenmenge zurückprojiziert werden. Hierdurch werden konkrete Belegungen für Variablen außerhalb des Affixes verhindert. Um dies umzusetzen, ist die folgende rationale Zeichenkettentransduktion definiert:

$$\widetilde{\mathcal{A}} = (\triangleleft \, \textcircled{\texttt{I}} \diamond (L(D) \times L(D)) \, \triangleright \cup \triangleleft \, (\Gamma - \textcircled{\texttt{I}}) \diamond L(D) \triangleright)^* \tag{4.45}$$

Wenn die beiden Constraint-Werte $\mathcal{L}_1$ und $\mathcal{L}_2$ angenommen werden, deren Supports eine Teilmenge der rationalen Sprache $(\triangleleft \, \textcircled{\texttt{I}} \diamond L(D) \triangleright)^5 \, (\triangleleft (\Gamma - \textcircled{\texttt{I}}) \diamond L(D) \triangleright)^5$ sind und die sich somit auf ein Präfix der Länge 5 beziehen, dann werden durch $Range(\mathcal{L}_1 \circ \widetilde{\mathcal{A}}) \cap \mathcal{L}_2$ beide Constraint-Werte kombiniert, wobei eventuelle Belegungen von Variablen außerhalb des Präfixes nicht berücksichtigt werden. Dies wird hier über das gezielte Entfernen von konkreten Variablenbelegungen realisiert $(L(D) \times L(D))$. Im Fall von Informationen, die mit den Variablen verknüpft sind, können diese auf die gleiche Weise behandelt werden. Denn auch diese können dazu führen, dass sich im Fall von verschiedenen Constraint-Systemen die Zwischenergebnisse unnötig aufblähen.

Für die Berechnung der abstrakten Lösung für die Ebenen einer memoisierten Kaskade mit Berücksichtigung der Präfix- oder Suffix-Analyse ist der Algorithmus 12 definiert, der auf dem Algorithmus 9 aus Abschnitt 4.5.9 aufbaut und das gezielte Aussetzen der Propagierung einbezieht. Dort wird in der Zeile 7 das Entfernen von konkreten Variablenbelegungen umgesetzt, sodass sie nicht mehr top-down propagiert werden.

4.5.11 Weitere Aufspaltung von Teilproblemen

Beschränkungen sind bei dem Problemzerlegungsansatz für den Bereich der Dependenzknoten-Teilprobleme formuliert. Jedoch kann dieser bereits zu weit gefasst sein, sodass die Formulierung von Beschränkungen schwierig wird oder zu WFSTs von enormer Größe führt. Aus diesem Grund soll es möglich sein, Dependenzknoten-Teilprobleme weiter aufzuspalten. Hierzu soll ein Beispiel im Kontext der Nominalisierung von Partizipien gegeben werden. In

Algorithmus 12 Berechnung der abstrakten Lösung für die Ebenen einer memoisierten Kaskade mit lokalem Aussetzen der Propagierung.

Eingabe: Eine memoisierte Kaskade $K = ((\widehat{\mathcal{K}}_1, \overline{\mathcal{K}}_1), (\widehat{\mathcal{K}}_2, \overline{\mathcal{K}}_2), \ldots, (\widehat{\mathcal{K}}_k, \overline{\mathcal{K}}_k))$ zu einem Constraint-Problem $P = (C_{\mathcal{R}^k}, con)$ und der Wert $\mathcal{T}$ des Top-Constraints.

Ausgabe: Eine memoisierte Kaskade der Länge k.

1: $K' = ()$
2: $\widehat{\mathcal{L}} \leftarrow Support(Best(\mathcal{T} \circ \widehat{\mathcal{K}}_k))$
3: $\overline{\mathcal{L}} \leftarrow Range(\widehat{\mathcal{L}}) \circ \overline{\mathcal{K}}_k$
4: $K' = K' + (\widehat{\mathcal{L}}, \overline{\mathcal{L}})$
5: $\mathcal{D} \leftarrow Support(\overline{\mathcal{L}})$
6: **for** $i \leftarrow k - 1$ **downto** 1 **do**
7: $\quad \mathcal{D} \leftarrow Range(\mathcal{D} \circ (\lhd \text{I} \diamond (L(D) \times L(D)) \rhd \cup \lhd (\Gamma - \text{I}) \diamond L(D) \rhd)^*)$
8: $\quad \widehat{\mathcal{L}} \leftarrow Support(Best(\mathcal{D} \circ \widehat{\mathcal{K}}_i))$
9: $\quad \overline{\mathcal{L}} \leftarrow Range(\widehat{\mathcal{L}}) \circ \overline{\mathcal{K}}_i$
10: $\quad K' = K' + (\widehat{\mathcal{L}}, \overline{\mathcal{L}})$
11: $\quad \mathcal{D} \leftarrow Support(Range(\overline{\mathcal{L}}))$
12: **end for**
13: **return** $reverse(K')$

Abbildung 4.20 ist die syntaktische Analyse des Fragments „die die Schweiz Liebende des Englischkurses“ zu sehen. Hier fungiert das Partizip 1 einerseits

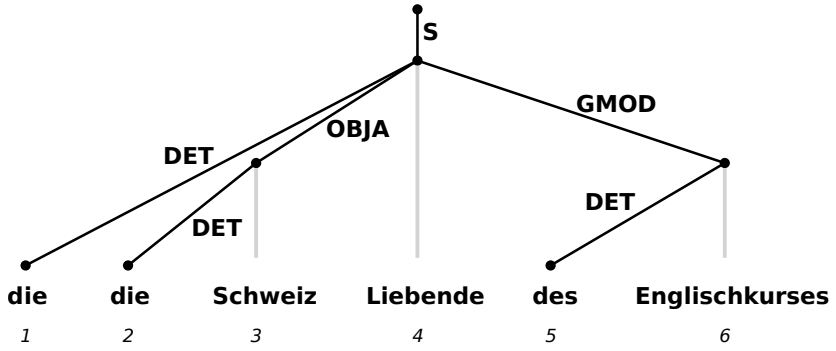


Abbildung 4.20: Nominalisierung von Partizipien.

als Verb, sodass dem Wort ein Akkusativobjekt (`OBJA`) untergeordnet ist, und andererseits als Nomen, sodass es einen Artikel (`DET`) und ein Genitivmodifizierer (`GMOD`) besitzt. Bei solch einer Nominalisierung können die Variablen, die die syntaktischen verbalen Eigenschaften und die Variablen, die die syntaktischen nominalen Eigenschaften betreffen, gut voneinander abgetrennt werden. So kann ein Dependenzknoten-Teilproblem, dass sich auf ein nominalisiertes Partizip bezieht, analog in zwei Teilprobleme aufgespalten werden, die im Folgenden als *Sub-Dependenzknoten-Teilprobleme* bezeichnet werden.

Ein Dependenzknoten-Teilproblem P, das die Governor-Variable g und die Modifier-Variablen $m_{1_1}, m_{1_2},\ldots,m_{1_h},\ldots,m_{2_1}, m_{2_2},\ldots,m_{2_i},\ldots m_{k_1}, m_{k_2},\ldots,$ m_{k_j} und die Variablen für kreuzende Wörter $x_1, x_2,\ldots,x_p$ umfasst, kann in die Sub-Dependenzknoten-Teilprobleme $\overline{P_1}, \overline{P_2}, \ldots, \overline{P_k}$ aufgespalten werden, wobei das Teilproblem $\overline{P_1}$ die Variablen $g, m_{1_1}, m_{1_2},\ldots,m_{1_h}$, das Teilproblem $\overline{P_2}$ die Variablen $g, m_{2_1}, m_{2_2},\ldots,m_{2_i}$, ... und das Teilproblem $\overline{P_k}$ die Variablen $g, m_{k_1}, m_{k_2},\ldots,m_{k_h}$ und die Variablen $x_1, x_2, \ldots, x_p$ enthält. Über die Sub-Dependenzknoten-Teilprobleme ist unter der Annahme einer Bottom-Up-Lösungsabfolge die Ordnung $\overline{P_1} \prec \overline{P_2} \prec \cdots \prec \overline{P_k}$ angenommen, sodass $\overline{P_1}$ das erste und $\overline{P_k}$ das letzte Teilproblem darstellt. Des Weiteren darf ein Sub-Dependenzknoten-Teilproblem nicht unär sein. Es darf also nicht nur aus einer Governor-Variable bestehen. Diese Aufspaltung ist in Abbildung 4.21 schematisch dargestellt. Dort gibt der blaue Pfeil die Bottom-Up- und der rote die Top-Down-Lösungsabfolge an. Kreuzende Wörter sind ausschließlich im letzten Sub-Dependenzknoten-Teilproblem erlaubt. So werden Über-

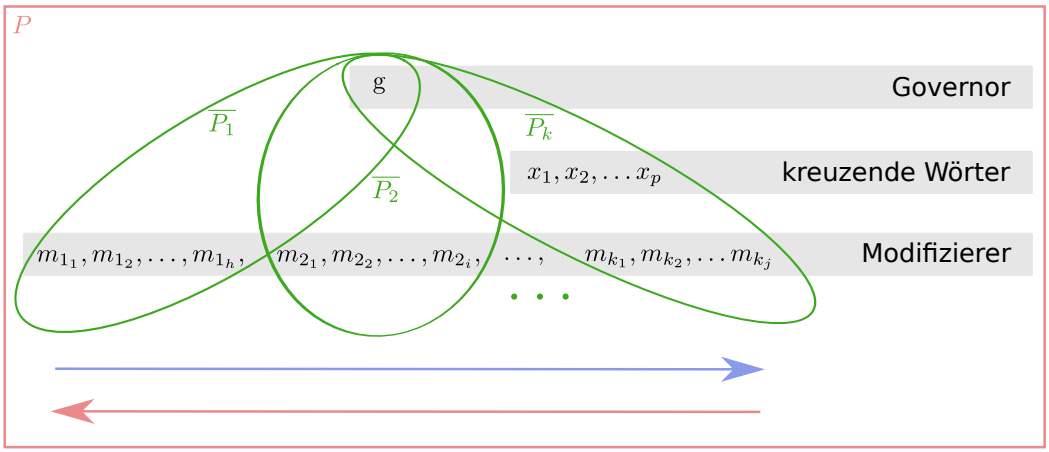


Abbildung 4.21: Weitere Aufspaltung eines Dependenzknoten-Teilproblems.

schneidungen der Teilprobleme über die kreuzenden Wörter ausgeschlossen. Für das Beispielfragment „die die Schweiz Liebende des Englischkurses“ ist in Abbildung 4.22 eine mögliche Zerlegung in die Dependenzknoten-Teilprobleme $P_1, P_2, \ldots, P_5$ und in die Sub-Dependenzknoten-Teilprobleme $\overline{P_1}$ und $\overline{P_2}$ mit $\overline{P_1} \prec \overline{P_2}$ gegeben. Auf Basis der Zerlegung in Beispiel 4.22 kann eine

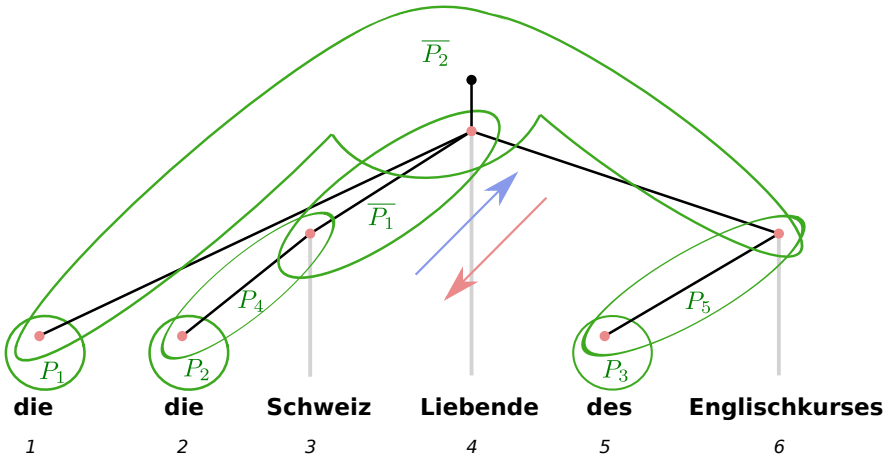


Abbildung 4.22: Problemzerlegung mit Sub-Dependenzknoten-Teilproblemen.

Zerlegung in Schicht-Teilprobleme angenommen werden. Diese ist in Abbildung 4.23 dargestellt. Hier befinden sich die beiden Sub-Dependenzknoten-Teilprobleme $\overline{P_1}$ und $\overline{P_2}$ in zwei unterschiedlichen Schichten und zwar in den Schicht-Teilproblemen P'_3 und P'_4.

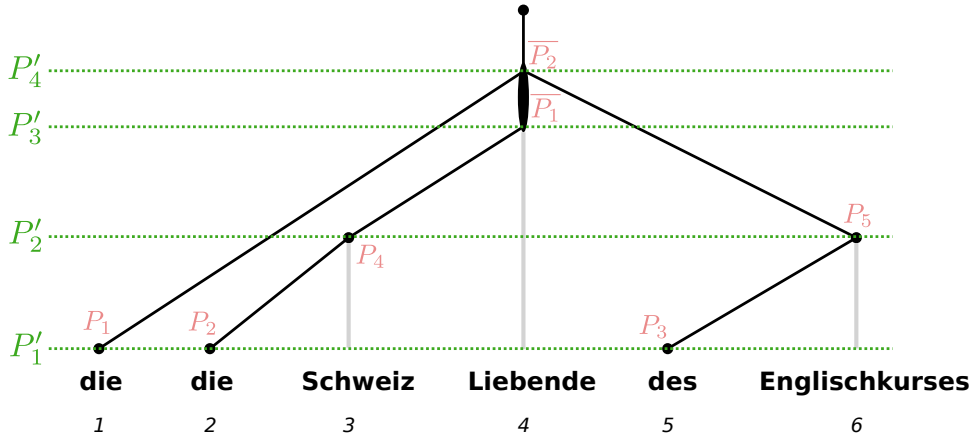


Abbildung 4.23: Zuordnung von Sub-Dependenzknoten-Teilproblemen zu Schicht-Teilproblemen.

Bei der Integration in den Problemzerlegungsansatz werden die Sub-Dependenzknoten-Teilprobleme ähnlich wie normale Dependenzknoten-Teilprobleme behandelt. Es muss lediglich erlaubt werden, dass der Governor eines Sub-Dependenzknoten-Teilproblems wieder als Governor innerhalb eines anderen Sub-Dependenzknoten-Teilproblems fungieren kann. So darf innerhalb des Filters F_u aus Abschnitt 4.5.6 für die aus Top-Down-Sicht nichtinitialen Sub-Dependenzknoten-Teilprobleme die Umschreibung Ⓜ ↤ Ⓖ nicht vollzogen werden.

Um die vorgegebene Lösungsabfolge der Sub-Dependenzknoten-Teilprobleme umzusetzen und um zu fordern, dass ein Governor eines aus Top-Down-Sicht nichtinitialen Sub-Dependenzknoten-Teilproblems wiederum in einem Sub-Dependenzknoten-Teilproblem involviert ist, müssen entsprechende Informationen mit den Governor-Variablen verknüpft werden. Solche Informationen können über die simultanen lokalen Konsistenzregeln dynamisch eingefügt werden. So muss beispielsweise in Beispiel 4.22 der Governor in $\overline{P_1}$ entsprechend markiert werden und $\overline{P_2}$ muss genau diese Governor-Markierung verlangen.

4.5.12 Überlokale Constraints

In McCrae et al. (2008) werden im Kontext des WCDG-Ansatzes überlokale Constraints einbezogen, die über Hilfsconstraints einen erweiterten Zugriff auf Informationen besitzen. In Abschnitt 4.2 wird vorgeschlagen, überlokale

Constraints über einen erweiterten Wertebereich umzusetzen, über den die Anwendungsergebnisse von Hilfsconstraints abgelegt und zugänglich gemacht werden. In diesem Abschnitt wird gezeigt, wie das Ablegen von Anwendungsergebnissen über *dynamische Informationen* realisiert werden kann, die mit den Variablen verknüpft werden. Es muss kein vollständiger Dependenzbaum vorliegen, um die Hilfsconstraints auszuwerten, und der Ansatz ist nicht auf unäre Hilfsconstraints beschränkt. Im Folgenden wird zudem gezeigt, dass überlokale Constraints auch über *dynamische Merkmale* umgesetzt werden können, bei denen die Überlokalität direkt und nicht über den Umweg der Hilfsconstraints implementiert wird.

Die dynamischen Informationen werden ausschließlich für den Informationsaustausch zwischen den einzelnen Dependenzknoten-Teilproblemen verwendet. Wenn beispielsweise alle simultanen lokalen Konsistenzregeln zusammengefasst werden würden, dann enthielte die resultierende Regel keine dynamischen Informationen mehr. Im Folgenden wird initial auf die Umsetzung überlokaler Constraints mithilfe von Hilfsconstraints eingegangen. Danach werden die dynamischen Merkmale eingeführt.

Hilfsconstraints

In diesem Abschnitt wird gezeigt, wie Hilfsconstraints umgesetzt werden können. Hierzu wird eine endliche Menge an Hilfsconstraints angenommen, die jeweils eine eindeutige ID besitzen. Eine endliche Menge dieser IDs kann analog zu einer endlichen Menge natürlicher Zahlen auf Zeichenkettenebene repräsentiert werden (vgl. Abschnitt 3.2.1). Zudem wird angenommen, dass die IDs in dem Alphabet Δ enthalten sind und so als Information mit einzelnen Variablen verknüpft werden können. Wenn die IDs $i_1, i_2, \ldots, i_n \in \Delta$ mit $i_1 \prec i_2 \prec \cdots \prec i_n$ gegeben sind, können alle möglichen Mengen über diesen IDs, also $\mathcal{P}(\{i_1, i_2, \ldots, i_n\})$, durch die folgende rationale Sprache repräsentiert werden: $\mathcal{I} = (i_1 \cup \epsilon)(i_2 \cup \epsilon)\ldots(i_n \cup \epsilon)$. Bei dieser Repräsentationsform wird eine totale Ordnung über den IDs vorausgesetzt. Wenn eine bestimmte ID in einer ID-Menge enthalten ist, soll dies bedeuten, dass der Check des entsprechenden Hilfsconstraints erfolgreich ist. Wenn sich die ID hingegen nicht in der Menge befindet, ist der Check gescheitert. Wenn $m \in \mathcal{I}$ gegeben ist, kann das Prüfen auf Enthaltensein einer ID i_k mit $1 \leq k \leq n$ wie folgt realisiert werden: $m \cap \Delta^* \; i_k \; \Delta^*$, wobei die ID genau dann enthalten ist, wenn das Ergebnis nicht der leeren Sprache entspricht. Die ID-Mengen

können Zerlegungsebene für Zerlegungsebene dynamisch mit einzelnen Variablen verknüpft werden. Wenn in einer Zerlegungsebene mehrere ID-Mengen für eine Variable möglich sind, führt dies zu unterschiedlichen Constraint-Systemen, die simultan und kompakt repräsentiert werden können.

Es wird zwischen den Hilfsconstraint-Typen *Has*, *Is*, *XHas* und *XIs* unterschieden und für jeden dieser Typen wird eine separate ID-Menge angenommen. Über die verschiedenen Typen kann auf unterschiedliche Art und Weise der lokale Beschränkungsbereich eines Constraints erweitert werden. Dies geschieht über die vier zweistelligen Prädikate $Has/2$, $Is/2$, $XHas/2$ und $XIs/2$, die über die Hilfsconstraint-ID-Mengen der einzelnen Constraint-Typen definiert sind. Das erste Argument der Prädikate ist eine Constraint-Variable und das zweite eine Hilfsconstraint-ID. Die Prädikate selbst beschreiben die Eigenschaft einer Variable, eine bestimmte Hilfsconstraint-ID als Information zu besitzen. Da die Prädikate auch innerhalb der Hilfsconstraints verwendet werden können, ist eine rekursive Prüfung von Beschränkungen möglich. Im Folgenden werden die einzelnen Prädikate und die dazugehörigen Hilfsconstraints genauer erläutert. Hierbei wird im Rahmen der Problemzerlegung eine Bottom-Up-Sicht eingenommen, da die Prädikate auf Basis dieser Perspektive benannt sind.

Has Das zweistellige Prädikat $Has/2$ ermöglicht es, ähnlich wie das gleichnamige Prädikat in McCrae et al. (2008), innerhalb eines Dependenzknoten-Teilproblems über eine Variable, die als Modifier oder kreuzendes Wort fungiert, Bedingungen auszudrücken, die für ein Dependenzknoten-Teilproblem gelten, in dem diese Variable Governor ist. Angenommen $A[\star]$ steht für die grundlegende Bedingung, die geprüft werden soll. Des Weiteren sei die Hilfsconstraint-ID i_k gegeben und es sei angenommen, dass das Prädikat $Gov/1$ die Eigenschaft einer Variable beschreibt, ein Governor zu sein. Dann lässt sich der Wert eines Has-Hilfsconstraints wie folgt definieren:

$$\begin{gathered}(\forall x\ A[x] \rightarrow (Gov(x) \rightarrow Has(x, i_k))) \\ \wedge \\ (\forall x\ Gov(x) \wedge Has(x, i_k) \rightarrow A[x])\end{gathered} \tag{4.46}$$

Innerhalb eines Dependenzknoten-Teilproblems besitzt eine Variable, die als Governor fungiert, nur genau dann die Hilfsconstraint-ID i_k, wenn die Bedingung $A[x]$ erfüllt ist. So kann innerhalb eines folgenden Dependenzknoten-

Teilproblems, das diese Variable als Modifier oder kreuzendes Wort enthält, die Beschränkung überlokal über die ID i_k geprüft werden.

Is Das zweistellige Prädikat $Is/2$ ermöglicht es, ähnlich wie das gleichnamige Prädikat in McCrae et al. (2008), innerhalb eines Dependenzknoten-Teilproblems über eine Variable, die als Governor oder als kreuzendes Wort fungiert, Bedingungen auszudrücken, die für ein Dependenzknoten-Teilproblem gelten, in dem diese Variable Modifier ist. Angenommen $A[\star]$ steht für die grundlegende Bedingung, die geprüft werden soll. Des Weiteren sei die Hilfsconstraint-ID i_k gegeben und es sei angenommen, dass das Prädikat $Mod/1$ die Eigenschaft einer Variable beschreibt, ein Modifier zu sein. Dann lässt sich der Wert eines Is-Hilfsconstraints wie folgt definieren:

$$\begin{gathered}(\forall x\ A[x] \rightarrow (Mod(x) \rightarrow Is(x, i_k))) \\ \wedge \\ (\forall x\ Mod(x) \wedge Is(x, i_k) \rightarrow A[x])\end{gathered} \tag{4.47}$$

Innerhalb eines Dependenzknoten-Teilproblems besitzt eine Variable, die als Modifier fungiert, nur genau dann die Hilfsconstraint-ID i_k, wenn die Bedingung $A[x]$ erfüllt ist. So kann innerhalb eines vorhergehenden Dependenzknoten-Teilproblems, das diese Variable als Governor enthält, die Beschränkung überlokal über die ID i_k geprüft werden.

XHas Das zweistellige Prädikat $XHas/2$ ermöglicht es, innerhalb eines Dependenzknoten-Teilproblems über eine Variable, die als Modifier oder als kreuzendes Wort fungiert, Bedingungen auszudrücken, die für ein vorhergehendes Dependenzknoten-Teilproblem gelten, in dem diese Variable ein kreuzendes Wort ist. Bei der Spezifizierung eines XHas-Hilfsconstraints müssen zwei Hilfsconstraint-ID-Mengen betrachtet werden: eine temporäre, die die XHas-Information über das aktuelle Dependenzknoten-Teilproblem widerspiegelt, und eine nichttemporäre, die die XHas-Information über ein Teilproblem einer vorhergehenden Zerlegungsebene liefert. Über Umschreibungen wird die nichttemporäre ID-Menge aber durch die temporäre ersetzt. So wird stets nur eine XHas-Information weitergereicht. Für die Abfrage der temporären Information wird zusätzlich das zweistellige Prädikat $XHas'/2$ herangezogen.

Angenommen $A[\star]$ steht für die grundlegende Bedingung, die geprüft

werden soll. Des Weiteren sei die Hilfsconstraint-ID i_k gegeben und es sei angenommen, dass das Prädikat $Xing/1$ die Eigenschaft einer Variable beschreibt, ein kreuzendes Wort zu sein. Dann lässt sich der Wert eines XHas-Hilfsconstraints wie folgt definieren:

$$\begin{gathered}(\forall x\ A[x] \rightarrow (Xing(x) \rightarrow XHas'(x, i_k))) \\ \wedge \\ (\forall x\ Xing(x) \wedge XHas'(x, i_k) \rightarrow A[x])\end{gathered} \tag{4.48}$$

Innerhalb eines Dependenzknoten-Teilproblems besitzt eine Variable, die als kreuzendes Wort fungiert, nur genau dann die Hilfsconstraint-ID i_k, wenn die Bedingung $A[x]$ erfüllt ist. So kann innerhalb eines folgenden Dependenzknoten-Teilproblems, das diese Variable als kreuzendes Wort, als Modifier oder als Governor enthält, die Beschränkung überlokal über die ID i_k geprüft werden.

XIs Das zweistellige Prädikat $XIs/2$ ermöglicht es, innerhalb eines Dependenzknoten-Teilproblems über eine Variable, die als Modifier oder ein kreuzendes Wort fungiert, Bedingungen auszudrücken, die für ein folgendes Dependenzknoten-Teilproblem gelten, in dem diese Variable ein kreuzendes Wort ist. Wie bei XHas müssen auch hier zwei Hilfsconstraint-ID-Mengen betrachtet werden: eine temporäre, die die XIs-Information über das aktuelle Dependenzknoten-Teilproblem widerspiegelt, und eine nichttemporäre, die XIs-Informationen über ein Teilproblem einer folgenden Zerlegungsebene liefert. Auch hier wird angenommen, dass über Umschreibungen die nicht temporäre ID-Menge durch die temporäre ersetzt wird. So wird stets nur eine XHas-Information weitergereicht. Für die Abfrage der temporären Information wird zusätzlich das zweistellige Prädikat $XIs'/2$ herangezogen.

Angenommen $A[\star]$ steht für die grundlegende Bedingung, die geprüft werden soll. Des Weiteren sei die Hilfsconstraint-ID i_k gegeben und es sei angenommen, dass das Prädikat $Xing/1$ die Eigenschaft einer Variable beschreibt, ein kreuzendes Wort zu sein. Dann lässt sich der Wert eines XIs-Hilfsconstraints wie folgt definieren:

$$\begin{gathered}(\forall x\ A[x] \rightarrow (Xing(x) \rightarrow XIs'(x, i_k))) \\ \wedge \\ (\forall x\ Xing(x) \wedge XIs'(x, i_k) \rightarrow A[x])\end{gathered} \tag{4.49}$$

Innerhalb eines Dependenzknoten-Teilproblems besitzt eine Variable, die als kreuzendes Wort fungiert, nur genau dann die Hilfsconstraint-ID i_k, wenn die Bedingung $A[x]$ erfüllt ist. So kann innerhalb eines vorhergehenden Dependenzknoten-Teilproblems, das diese Variable als kreuzendes Wort oder als Governor enthält, die Beschränkung überlokal über die ID i_k geprüft werden.

Dynamische Merkmale

Hier werden dynamische Merkmale vorgeschlagen, die mit bestimmten Variablen verknüpft sein können, um eine feingliedrigere Formulierung von Beschränkungen zu ermöglichen. Die Bezeichnung 'dynamisch' meint, dass die Merkmale erst während des Bottom-Up-Lösens zugewiesen werden.

Die Merkmale können sich auf bestimmte linguistische Phänomene beziehen. Es soll ein Beispiel hierfür im Kontext des Genus Verbi gegeben werden. Eine Verbalgruppe kann entweder das Aktiv oder verschiedene Passivformen widerspiegeln. So könnten mehrere Has- und Is-Hilfsconstraints herangezogen werden, um im Kontext einer Verbalgruppe die Einigkeit über das Genus Verbi herzustellen. Dies ist jedoch sehr umständlich, da das gegenseitige Ausschließen der einzelnen Genus Verbi formuliert werden müsste. Alternativ kann ein dynamisches Merkmal für das Genus Verbi eingeführt werden. Dieses wird den Verben zugewiesen, die in der betreffenden Verbalgruppe involviert sind. Constraints können dann über Dependenzknoten-Teilprobleme hinweg Bezug auf dieses Merkmal nehmen, um auf die Übereinstimmung bezüglich des Genus Verbi zu prüfen. Im Grunde lässt sich der Effekt von dynamischen Merkmalen auch durch eine feingliedrigere Dependenz-Label-Menge umsetzen. Das verhält sich ähnlich zu den Hilfsconstraints, die theoretisch auch über den Wertebereich der Constraint-Variablen umgesetzt werden können (siehe Abschnitt 4.2.1). Der große Vorteil der dynamischen Merkmale im Gegensatz zu feingliedrigeren Dependenz-Labeln ist, dass sie nur punktuell im Rahmen der Kommunikation zwischen den Dependenzknoten-Teilproblemen bestehen.

Es sei ein dynamisches Merkmal β mit den Merkmalswerten $\spadesuit_1, \spadesuit_2, \ldots, \spadesuit_n$ gegeben. Dann wird für jedes Paar aus Merkmalsname und Merkmalswert $(\beta, \spadesuit_k)$ mit $1 \leq k \leq n$ eine eindeutige ID $i_{(\beta,\spadesuit_k)}$ angenommen. Wenn mehrere Merkmale gegeben sind über die eine totale Ordnung definiert ist, kann eine Menge von Merkmal-IDs wie eine Menge von Hilfsconstraint-IDs repräsentiert werden. So können die IDs über das Alphabet Δ als Information mit

einzelnen Variablen verknüpft werden.

Die ID-Mengen können Zerlegungsebene für Zerlegungsebene dynamisch mit einzelnen Variablen verknüpft werden. Wenn in einer Zerlegungsebene mehrere ID-Mengen für eine Variable möglich sind, führt dies zu unterschiedlichen Constraint-Systemen, die simultan und kompakt repräsentiert werden können. Der Zugriff auf die dynamischen Merkmale geschieht über das zweistellige Prädikat $DFeat/2$, das die Eigenschaft einer Variable beschreibt, eine bestimmte Merkmal-ID als Information zu besitzen. Das erste Argument des Prädikats ist eine Constraint-Variable und das zweite eine Merkmal-ID.

Bei den dynamischen Merkmalen muss berücksichtigt werden, dass sich die Merkmalswerte gegenseitig ausschließen. Wenn die Merkmale β_1,β_2,..., β_n gegeben sind, und wenn die Menge M definiert ist durch $\{\{i_1, i_2, \ldots, i_n\} \mid i_k = i_{(\beta_k,\spadesuit)}\ mit\ 1 \leq k \leq n$ und $\spadesuit$ ist ein Merkmalswert von $\beta_k\}$, dann sind alle gültigen Merkmal-ID-Mengen durch $\bigcup_{N \epsilon M} \mathcal{P}(N)$ gegeben. Es sei das dynamische Merkmal β mit den Merkmalswerten $\spadesuit_1$,$\spadesuit_2$,...,$\spadesuit_n$ angenommen. Dann ist die rationale Sprache $\mathcal{I}$, die alle IDs zu einem Merkmal enthält, definiert durch: $i_{(\beta,\spadesuit_1)} \cup i_{(\beta,\spadesuit_2)} \cup \ldots i_{(\beta,\spadesuit_n)}$. So kann eine Teilformel $B[\star]$ definiert werden, die ausdrückt, dass sich die einzelnen Merkmalswerte gegenseitig ausschließen:

$$B[\star] = \left(\begin{array}{c} DFeat(\star, i_{(\beta,\spadesuit_1)}) \cap \neg DFeat(\star, \mathcal{I} - i_{(\beta,\spadesuit_1)}) \\ \cup \\ DFeat(\star, i_{(\beta,\spadesuit_2)}) \cap \neg DFeat(\star, \mathcal{I} - i_{(\beta,\spadesuit_2)}) \\ \cup \\ \vdots \\ \cup \\ DFeat(\star, i_{(\beta,\spadesuit_n)}) \cap \neg DFeat(\star, \mathcal{I} - i_{(\beta,\spadesuit_n)}) \end{array} \right) \tag{4.50}$$

Zudem muss bei den dynamischen Merkmalen berücksichtigt werden, dass sie nur mit bestimmten Variablen verknüpft werden sollen. Dies und das gegenseitige Ausschließen der Merkmalswerte werden über einen quantifizierten regulären Constraint umgesetzt. Es sei die Beschränkung für die Zuweisung des dynamischen Merkmals durch $A[\star]$ gegeben. Dann ist der Wert des Con-

straints definiert durch:

$$\begin{array}{c}(\forall x\ Var(x) \wedge DFeat(x,\mathcal{I}) \leftrightarrow Var(x) \wedge A[x]) \\ \wedge \\ (\forall x\ Var(x) \wedge DFeat(x,\mathcal{I}) \rightarrow Var(x) \wedge B[x])\end{array} \tag{4.51}$$

Innerhalb eines Dependenzknoten-Teilproblems besitzt eine Variable nur genau dann die Merkmal-ID i_k, wenn die Bedingung $A[x]$ erfüllt ist. So kann innerhalb eines anderen Dependenzknoten-Teilproblems, das diese Variable enthält, die Beschränkung überlokal über die ID i_k geprüft werden.

4.5.13 Dependenzstruktur und Verletzungsnachverfolgung

Es ist generell möglich, beim Parsing zusätzlich zu den Labeln die Dependenzstruktur über die Variablenwerte aufzubauen. Hierzu muss die Position des übergeordneten Wortes über die Variablen behandelt werden und es muss die Information über die Position eines Wortes im Satz mit einer Variable verknüpft sein. Alternativ kann die Struktur der Zerlegung durch spezielle Zeichen angezeigt werden. Im Parsing-Ansatz von Oflazer (2003) wird dies über Kanäle realisiert (siehe Abschnitt 4.1.1). In Yli-Jyrä (2005b) wird hingegen ein Klammerungsschema vorgestellt, um projektive Dependenzstrukturen darzustellen und in Yli-Jyrä (2004) wird ein gefärbtes Klammerungsschema vorgestellt, um nichtprojektive Dependenzstrukturen zu behandeln. Grundsätzlich geschieht das Anzeigen der Strukturen über matchende Zeichenpaare. Hier wird eine Darstellungsvariante vorgestellt, die ein Mix aus den oben genannten Darstellungsformen ist.

Es wird ausgenutzt, dass eine Zerlegung in Ebenen unterteilt ist und das die Dependenzknoten-Teilprobleme innerhalb der Ebenen nicht überlappen. So lässt sich innerhalb einer Ebene auf einfache Weise eine Zuordnung der Modifier zu den Governor herstellen, indem mit den Variablen bestimmte Unterordnungsinformationen verknüpft werden. Es werden die Zeichen $\gg, \ll, \square \in \Gamma$ verwendet, um die lokalen Unterordnungen anzuzeigen. Die Kodierungen der Unterordnungen bezüglich der Variablen sind in Tabelle 4.3 aufgeführt, wobei ein Constraint-System $CS = (S, D, V)$ angenommen wird. Ein linksseitiger Modifier ist stets dem nächsten Governor zur Rechten untergeordnet, ein rechtsseitiger Modifier hingegen stets dem nächsten Governor

Mitspieler	Darstellung
linksseitiger Modifier	$\lhd \gg \diamond L(D) \rhd$
rechtsseitiger Modifier	$\lhd \ll \diamond L(D) \rhd$
Governor	$\lhd \square \diamond L(D) \rhd$

Tabelle 4.3: Darstellung von Unterordnungen innerhalb einer Ebene.

zur Linken. So wird beispielsweise durch die drei nacheinander folgenden Variablen $\lhd \gg \diamond L(D) \rhd \lhd \gg \diamond L(D) \rhd \lhd \square \diamond L(D) \rhd$ ein Governor mit zwei direkt untergeordneten linksseitigen Modifiern repräsentiert.

Um eine eindeutige Zuordnung über die verschiedenen Zerlegungsebenen hinweg zu gewährleisten, kann für jede Ebene eine eindeutige Färbung angenommen werden. Es sei eine Zerlegungstiefe k gegeben. Dann können die Zeichen $\gg_1, \gg_2, \ldots, \gg_k \in \Gamma$, $\ll_1, \ll_2, \ldots, \ll_k \in \Gamma$ und $\square_1, \square_2, \ldots, \square_k \in \Gamma$ angenommen werden. Nun sollen die Auszeichnungen der verschiedenen Ebenen zusammengebracht werden. Eine Variable, die in einer Ebene Governor ist, ist auch stets in einer höheren Ebene Modifier oder die Variable ist dem Dummy-Wurzelknoten direkt untergeordnet. Für den letzteren Fall wird das Zeichen $\top \in \Gamma$ herangezogen. Die verschiedenen Codierungsmöglichkeiten der Unterordnungsbeziehungen zu einer Variable, die in der Ebene r Governor ist, sind in Tabelle 4.4 zusammengefasst.

Unterordnung	Darstellung
linksseitiger Modifier eines Governors der i-ten Ebene mit $i > r$	$\lhd \gg_i \square_r \diamond L(D) \rhd$
rechtsseitiger Modifier in der i-ten Ebene mit $i > r$	$\lhd \ll_i \square_r \diamond L(D) \rhd$
dem Dummy-Wurzelknoten direkt untergeordnet	$\lhd \top \square_r \diamond L(D) \rhd$

Tabelle 4.4: Darstellung ebenenübergreifender Unterordnungen.

Als Beispiel ist für den Satz „Die Köchin macht Raclette“ in Abbildung 4.24 die Codierung des entsprechenden gelabelten Dependenzbaums auf Zeichenkettenebene gegeben. Die einzelnen drei Zerlegungsebenen sind durch die grünen horizontalen gestrichelten Linien angezeigt.

Die Codierung der Unterordungsinformation könnte auch über die Variablenwerte behandelt werden. Dann wäre jedoch ein Maximum für die Zerlegungstiefe gesetzt, da die Anzahl der Färbungen gebunden wäre. Hier wird

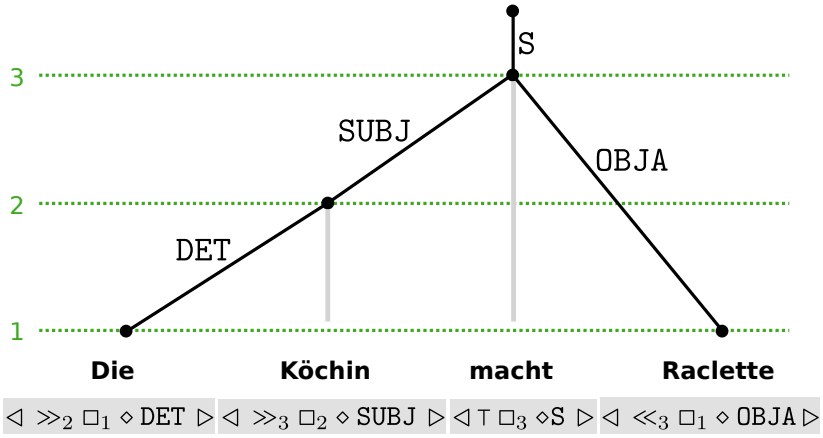


Abbildung 4.24: Dependenzbaumcodierung auf Zeichenkettenebene.

hingegen angenommen, dass die Struktur als Seiteneffekt aufgebaut wird. So können die ebenenbezogenen Unterordnungen beim Parsing über die simultanen lokalen Konsistenzregeln eingeführt werden. Die Färbung kann dann über Zeichensubstitutionen innerhalb der entsprechenden Ebenen umgesetzt werden. Hierbei wird angenommen, dass ein ausreichender Vorrat an Markierungszeichen existiert.

Zusätzlich zum Aufbauen einer Dependenzstruktur können auch Verletzungen von weichen Constraints den einzelnen Dependenzknoten zugeordnet werden. Hierdurch wir ein Nachverfolgen von Constraint-Verletzungen möglich. Wenn jedem weichen Constraint eine eindeutige ID zugeordnet ist, können diese IDs im Falle einer Verletzung auf Zeichenkettenebene eingefügt werden. Hierbei sollen die Verletzungen stets dem Governor zugeschrieben werden. Dieses Einfügen kann beispielsweise geschehen, nachdem die weichen Constraints bereits angewendet wurden. Dann werden final nochmals die weichen Constraints anhand der Problemzerlegung angewendet, wobei keine Strafgewichte vergeben, sondern die Constraint-IDs eingefügt werden. Über die so ausgezeichnete Dependenzstruktur ist dann eine Zuordnung von den Constraint-IDs zu den Dependenzknoten möglich.

Kapitel 5

Praktische Umsetzung

5.1 Endliche-Maschine-Bibliothek

Für Teile der Implementierung des neuen Ansatzes wurde die an der Universität Potsdam entwickelte Bibliothek *FSM<2.0>* herangezogen, für die auch ein Skriptspracheninterpreter angeboten wird (Hanneforth, 2010). Diese ist in C++ implementiert und ermöglicht wie die bekanntere frei verfügbare Bibliothek *OpenFst* (Allauzen et al., 2007) die Erstellung und Manipulation von gewichteten FSMs. Im Rahmen des Projekts *das Digitale Wörterbuch der deutschen Sprache* (DWDS, *http://www.dwds.de*) wird diese Bibliothek für die Eigennamenerkennung (siehe Didakowski und Drotschmann, 2008), die morphologische Analyse (siehe Geyken und Hanneforth, 2005) und die syntaktische Analyse (siehe Didakowski, 2008b) verwendet. Jedoch unterstützt der enthaltene Reguläre-Ausdruck-Compiler weder Unicode-Zeichen für mathematische Operatoren und Symbole noch reguläre prädikatenlogische Formeln. Daher wurde ein Compiler implementiert, der auf der FSM<2.0>-Bibliothek basiert und zusätzlich die Syntax der regulären Ausdrücke und regulären prädikatenlogischen Formeln umsetzt, wie sie in Abschnitt 2.2.2 definiert wurde. Für die Berechnung der Problemzerlegung und der (abstrakten) Lösung wurde hingegen eine eigene C++-Implementierung herangezogen, die für die Komposition von gewichteten FSMs und für Äquivalenztransformationen über azyklische Maschinen optimiert ist.

5.2 Lexikalische Ressource

Für die Ermittlung der lexikalischen Informationen wird auf das Vollformlexikon zurückgegriffen, auf dem die deutsche Grammatik der WCDG basiert (Foth, 2006a). Dieses unterstützt Subkategorisierungsrahmen für Verben und Nomen und gibt Auskunft über mögliche Verbpräfixe und semantische und grammatische Merkmale. Die Informationen sind hierbei für die Parsingaufgabe abgestimmt.

Im Rahmen dieser Arbeit wurde das Lexikon in einen FST kompiliert, der eine Vollform auf mögliche Lemmaformen und Wortarten in Form von STTS-Tags und auf lexikalische Merkmale abbildet (für das STTS-Tagset siehe Schiller et al., 1995). In Abbildung 5.1 ist als Beispiel ein Analyseautomat für das Wort 'Absicht' gegeben. In dem Übergangsdiagramm ist die

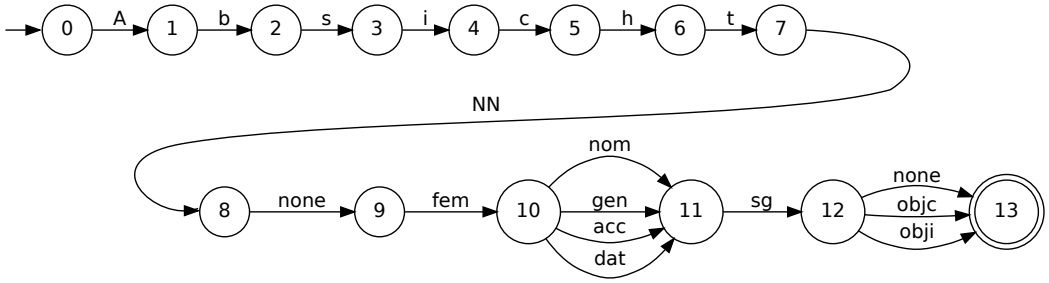


Abbildung 5.1: Wortanalyse.

Lemmaform, gefolgt von dem STTS-Tag `NN`, das für ein Nomen steht, zu sehen. Der Wortart sind lexikalische Merkmale nachgestellt, die analog zur Repräsentation innerhalb des FSIG-Ansatzes implementiert sind. Hierbei ist jede Zeichenkettenposition wortartspezifisch einem bestimmten Merkmal zugeordnet. In dem gegebenen Beispiel ist die Merkmalsabfolge: semantischer Typ ≺ Genus ≺ Kasus ≺ Numerus ≺ Valenz. Dem Nomen ist kein semantischer Typ zugeordnet (`none`) und der Subkategorisierungsrahmen fordert optional ein Satzobjekt (`objc`) oder ein Infinitivobjekt (`obji`). Falls ein Wort verschiedene Wortarten und Merkmale besitzt, sind die Analysen kompakt repräsentiert.

Um eine robuste Analyse zu ermöglichen, wird der Ansatz von Lindén (2009) angewendet, um auch für unbekannte Inhaltswörter Lexikoneinträge erhalten zu können. Hierzu wird aus einem Lexikon-FST ein Rater-WFST

konstruiert, mit dessen Hilfe über längste übereinstimmende Suffixe Einträge erraten werden können. So kann beispielsweise auch für komplexe Komposita ein Lexikoneintrag gefunden werden, wenn das Erstglied nicht bekannt ist. Die lexikalischen Merkmale werden bei Nomen und Adjektiven anhand der Deklination und bei Verben anhand der Konjugation erraten, wobei bestimmte Informationen wie die Subkategorisierungsrahmen als undefiniert markiert werden. Bei komplett unbekannten Wörtern werden ausgewählte STTS-Tags zusammen mit undefinierten lexikalischen Merkmalen zugewiesen.

Der Lexikon-FST und der Rater-WFST sind so konstruiert, dass eine Vollform bezüglich eines bestimmten STTS-Tags nachgeschlagen werden kann. Dies ist dann der Fall, wenn der Eingabesatz automatisch über einen Wortarten-Tagger oder als Goldstandard getaggt ist. Letzteres wird in der vorliegenden Arbeit angenommen.

5.3 Constraint-System

5.3.1 Wertebereich

Es werden grundsätzlich die Dependenz-Label herangezogen, die auch innerhalb der WCDG-Grammatik verwendet werden. In der Tabelle 5.1 sind die einzelnen Label zusammengefasst. Für deren genauere Beschreibung sei auf die WCDG-Annotationsrichtlinien in Foth (2006a) verwiesen, in der die zusätzlichen Label `NP2` und `EXPL` enthalten sind. Das Label `NP2` wird dort verwendet, um ein Subjekt in einer elliptischen Koordination einzubinden. Hier ist solch eine Einbindung über leere Elemente umgesetzt (siehe Abschnitt 5.6.7). Das Label `EXPL` wird dort herangezogen, um expletive Pronomen parallel zu ihren Korrelaten einzubinden. Hier werden die Korrelate den expletiven Pronomen untergeordnet (siehe Abschnitt 5.6.3). Ein weiterer Unterschied besteht in dem Label `PAR`. Dieses dient in den WCDG-Annotationsrichtlinien dazu, parenthetische Matrixsätze dem Objektsatz unterzuordnen, wenn es zu Nichtprojektivität kommen würde. In der vorliegenden Arbeit wird hingegen nicht versucht, solche nichtprojektiven Strukturen zu vermeiden (siehe Abschnitt 5.6.3).

Label	Funktion	Beispiel
”	Anbindung von Satzzeichen	Hallo **!**
ADV	adverbialer Modifikator	Ich schlafe **heute**
APP	Apposition	die Frau, eine **Heldin**, ...
ATTR	pränominales Attribut	die **schöne** Frau
AUX	nicht finiter Verbgruppenbestandteil	Er will **heiraten**.
AVZ	abgetrenntes Verbpräfix	Er legt den Hörer **auf**.
CJ	koordiniertes Element	die Frau und das **Kind**
DET	Determiner	**die** Frau
ETH	ethischer Dativ	Du bist **mir** ein feiner Kerl.
GMOD	Genitivattribut	der Mann der **Tat**
GRAD	nominale Maßangabe	zwei **Jahre** älter
KOM	vergleichende Konjunktion	weiß **wie** Schnee
KON	koordinierende Konjunktion	die Frau **und** das Kind
KONJ	subordinierende Konjunktion	Ich arbeite, **wenn** sie schläft.
NEB	Nebensatz	Ich arbeite, **wenn** sie schläft.
OBJA	Akkusativobjekt	Die Frau liebt den **Mann**
OBJA2	zweites Akkusativobjekt	Er nennt die Frau eine **Heldin**
OBJD	Dativobjekt	Huldigt dem **König**!
OBJG	Genitivobjekt	Er beschuldigt ihn eines **Vergehens**.
OBJC	Satzobjekt	Ich weiß, dass er **lügt**.
OBJI	Infinitivobjekt	Ich versuche zu **schlafen**.
OBJP	Präpositionalobjekt	Ich zweifle **an** der Gerechtigkeit.
PART	Partikel	Ich versuche **zu** schlafen.
PN	Komplement einer Präposition	in der **Nacht**.
PP	präpositionaler Modifikator	Sie lebt **in** der Stadt.
PRED	prädikative Ergänzung	Sie ist eine **Heldin**.
REL	Relativsatz	die Frau, die **schläft**, ...
S	Hauptsatz/Fragment	die Frau **schläft**
SUBJ	nominales Subjekt	die **Frau** schläft
SUBJC	Subjektsatz	Wer **schläft**, ist eine Schlafmütze.
VOK	Vokativ	**Herr**, es ist Zeit.
ZEIT	nominale Zeitangabe	Sie ging **1979** ins Kino.

Tabelle 5.1: Dependenz-Label.

5.3.2 Gewichtsstruktur

Die syntaktische Analyse geschriebener natürlicher Sprache ist eine realitätsnahe Situation, die Verletzbarkeit, Präferenzen und Unsicherheit beinhaltet. Zentral ist hierbei das Phänomen der Gradation, die dort allgegenwärtig anzutreffen ist (siehe Schröder, 2002, Abschnitt 1.1 und Fanselow, 2004). Diese wird innerhalb der WCDG über eine Kostenstruktur abgebildet, die negative Präferenzen über reelle Zahlen von 0 bis 1 implementiert. Über diesen Zahlenbereich sind bestimmte gerankte Bewertungsbereiche definiert (Foth, 2006a, Abschnitt 2.1.3): ein Bereich für Universal-Constraints, mit denen Debugging realisiert werden kann; ein Bereich für Präferenz-Constraints; ein Bereich für heuristische Regeln, die öfter verletzt werden; ein Bereich für minderschwere Fehler; ein Bereich für schwere Fehler; ein Bereich für harte Constraints, die die Wohlgeformtheitsregeln der Grammatik umsetzen. Bei dieser Umsetzung kann bei Mehrfachverletzungen von weichen Constraints innerhalb eines Bewertungsbereichs ein Bewertungselement hervorgehen, das sich in einem anderen Bewertungsbereich befindet. Es ist also keine klare Trennung zwischen den einzelnen Bereichen gewährleistet (siehe Schröder, 2002, Abschnitt 3.5.7).

Um eine klare Trennung zwischen den einzelnen Bewertungsbereichen umzusetzen, kann ein lexikografischer n-dimensionaler Semiring herangezogen werden. In Abschnitt 4.5.9 wird gezeigt, dass über die verzögerte Anwendung von Constraints der lexikografische Charakter solch eines Semirings nachempfunden werden kann. Da hier nicht die Bewertungselemente zu den Analysen relevant sind, sondern ausschließlich ihre Ordnung, wird innerhalb der Implementierung auf einen lexikografischen n-dimensionalen Semiring verzichtet. So werden die Gewichtungen über ein gewichtetes CSP, also über den c-Semiring $(\mathbb{R}_0^+ \cup \{+\infty\}, min, +, +\infty, 0)$ umgesetzt. Über die verschiedenen numerischen Bewertungsbereiche können dann Constraint-Mengen gebildet werden, über die das gewünschte Ranking mithilfe der verzögerten Anwendung von Constraints eingebracht wird.

Es werden in der Implementierung ähnliche Bewertungsbereiche wie in der deutschen WCDG-Grammatik verwendet. Diese sind in Tabelle 5.2 aufgeführt. Hier ist der nummerische Bereich für die Präferenzen und für die schweren Fehler etwas größer gewählt als bei der WCDG.

Bewertungsbereich	Bewertungstyp
0,9999 bis 0,7	Präferenzen
0,7 bis 0,5	heuristische Regeln
0,5 bis 0,2	minder schwere Fehler
0,2 bis 0,001	schwere Fehler

Tabelle 5.2: Verwendete Bewertungsbereiche.

5.3.3 Verknüpfte Informationen

Jedes Wort eines Eingabesatzes wird durch eine Constraint-Variable repräsentiert. Damit ein Zugriff auf tokenspezifische Informationen möglich ist, werden mit den einzelnen Variablen Wortinformationen verknüpft. Diese sind auf Zeichenkettenebene repräsentiert und umfassen Folgendes:

- **Wortform**: Form des Wortes im Satz
- **Analysestatus**:
  - Das Wort steht im Lexikon.
  - Es wurde ein Lexikoneintrag erraten.
  - Es kann kein Lexikoneintrag gefunden oder erraten werden.
- **Lemmaform**: die Lemmaform des Wortes
- **Wortart**: das STTS-Tag des Wortes
- **Lexikalische Merkmale**: zusätzliche wortartbezogene Informationen

Der Analysestatus wird durch eine einelementige Zeichenkette über dem Alphabet $\{\star_{lexicon},\star_{guessed},\star_{unknown}\}$ repräsentiert. Die Lemmaform und die Oberflächenform werden jeweils durch eine eindeutige Identifizierungsnummer dargestellt, die über eine Zeichenkette über dem Alphabet $\{\star_0,\star_1,\star_2,\ldots,\star_9\}$ codiert ist. Hierbei wird einer Lemmaform oder einer Oberflächenform, die nicht in der Grammatik verwendet werden, die Identifikationsnummer '0' zugewiesen. Die Repräsentation der Wortart und der lexikalischen Merkmale ist analog zu der eines Lexikoneintrags (siehe Abschnitt 5.2).

Im Folgenden wird ein konkretes Beispiel für eine Constraint-Variable mit verknüpften Informationen gegeben. Hierzu wird eine Variable betrachtet, die das Nomen 'Absicht' repräsentiert. Diese ist in Abbildung 5.2 in Form

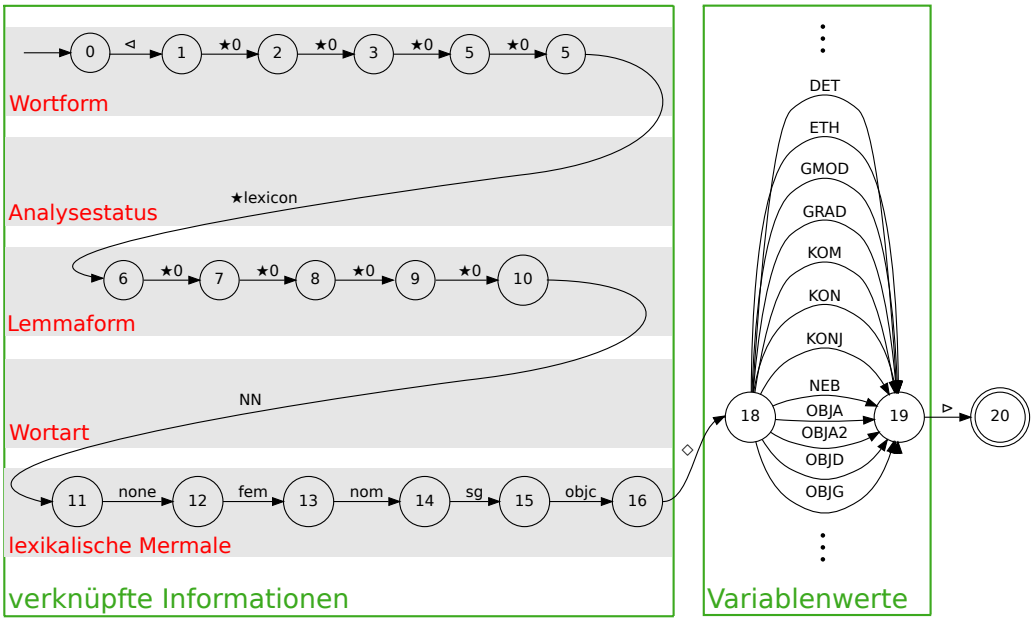


Abbildung 5.2: Constraint-Variable.

eines FSA abgebildet. Wenn ein Wort lexikalisch ambig ist, werden die einzelnen Lesarten über ein simultanes Constraint-System kompakt repräsentiert. Wenn für dieses genau eine Lösung gefunden werden kann, heißt dies auch, dass die lexikalischen Merkmale vollständig disambiguiert werden können.

5.3.4 Virtuelle Knoten

In Beuck und Menzel (2013) werden im Kontext des inkrementellen WCDG-Parsings virtuelle Knoten vorgeschlagen, um während des Parsing strukturelle Vorhersagen machen zu können. Diese repräsentieren Pseudowörter, die erwartungsgemäß innerhalb eines späteren Inkrements vorkommen, und ermöglichen auch für Satzpräfixe eine verbundene Dependenzstruktur.

In der vorliegenden Arbeit werden virtuelle Knoten herangezogen, um linguistische Phänomene wie Strukturellipsen (siehe Abschnitt 5.6.6) und Koordinationsellipsen (siehe Abschnitt 5.6.7) zu behandeln. Zudem werden sie dazu verwendet, Informationen in mehreren Dependenzknoten-Teilproblemen verfügbar zu machen (siehe Abschnitt 5.6.1 und 5.6.4). Es wird aber auch gezeigt, dass sie für die Modellierung von Koordination herangezogen werden können (siehe Abschnitt 5.6.5).

Hier sind virtuelle Knoten nichts anderes als zusätzliche Variablen, die für Wörter stehen, die nicht in der Eingabe vorkommen und mit denen bestimmte Informationen verknüpft sind. Sie können initial an unterschiedlichen Positionen des Eingabesatzes angenommen werden, was über verschiedene simultan repräsentierte Constraint-Systeme umgesetzt werden kann. Hierbei kann die Anzahl an Constraint-Systemen sogar ungebunden sein. Hierfür soll ein Beispiel gegeben werden. Es sei ein Satz mit den Wörtern $w_1, w_2, \ldots, w_n$ gegeben, die über die Variablen $v_{w_1}, v_{w_2}, \ldots, v_{w_n}$ repräsentiert sind. Des Weiteren sei die rationale Sprache $\mathcal{N}$ angenommen, die für einen virtuellen Knoten steht. Dann hat die simultane Variablenmenge, die deren Auftreten uneingeschränkt behandelt, folgende Form: $\mathcal{N}^*\ L(v_{w_1})\ \mathcal{N}^*\ L(v_{w_2})\ \mathcal{N}^*\ \ldots\ \mathcal{N}^*\ L(v_{w_n})\ \mathcal{N}^*$.

Hier ist jedoch zu beachten, dass die Repräsentation der Variablen über eine unendliche rationale Sprache dazu führen kann, dass der Lösungsalgorithmus letztlich nicht terminiert. Zudem müssen für die Epsilonentfernung, für die Suche des besten Pfades und für die Minimierung spezielle Algorithmen für zyklische FSMs herangezogen werden, die aufwendiger sind als die für azyklische Maschinen. Daher wird innerhalb der Implementierung gefordert, dass die Anzahl an virtuellen Knoten gedeckelt ist. Wenn beispielsweise angenommen wird, dass maximal zwei solcher Knoten optional zwischen zwei Wörtern der Eingabe stehen dürfen, hat die entsprechende simultane Variablenmenge die Form: $(\mathcal{N} \cup \mathcal{N}\mathcal{N} \cup \epsilon)\ L(v_{w_1})\ (\mathcal{N} \cup \mathcal{N}\mathcal{N} \cup \epsilon)\ L(v_{w_2})\ (\mathcal{N} \cup \mathcal{N}\mathcal{N} \cup \epsilon)\ \ldots\ (\mathcal{N} \cup \mathcal{N}\mathcal{N} \cup \epsilon)\ L(v_{w_n})\ (\mathcal{N} \cup \mathcal{N}\mathcal{N} \cup \epsilon)$.

Es ist jedoch nicht immer notwendig, an jeder Position des Eingabesatzes virtuelle Knoten anzunehmen. So kann deren Auftreten auch weiter eingeschränkt werden. Ein entsprechender Filter kann beispielsweise über den Kontextbeschränkungsoperator oder eine reguläre prädikatenlogische Formel kompiliert und über die Schnittbildung angewendet werden. Dies ist in der vorliegenden Implementierung der Fall. Alternativ kann die Variablenmenge aber auch initial bereits mit den entsprechenden Positionseinschränkungen konstruiert werden.

5.4 Grundlegendes Grammatikdesign

Initial war die Idee, sich an der WCDG-Grammatik für das Deutsche (Foth, 2006a) zu orientieren und Regeln zu übernehmen. Jedoch ist dies nur bedingt

möglich. Das liegt hierbei eher an der Philosophie des Grammatikkonzeptes als an dem unterschiedlichen Beschränkungsbereich. Innerhalb der WCDG-Grammatik sind die Regeln eher locker definiert und folgen einer Default-Strategie. Solch eine Umsetzung ist innerhalb des Zerlegungsansatzes nicht realisierbar. Für die Berechnung der Zerlegung muss ein grober Rahmen über harte Beschränkungen gegeben sein. Das bedeutet aber nicht, dass darunter die Robustheit leidet. Es werden durch den harten Rahmen lediglich die Analysen ausgeschlossen, die generell zu keinem sinnvollen Ergebnis führen würden. Der Rahmen muss demnach bezüglich der Realisierbarkeit der Problemzerlegung und der Robustheit gesteckt werden.

Grundlegend orientiert sich die Grammatik an den Annotationsrichtlinien in Foth (2006a). Sie wurde in der ersten Entwicklungsphase ohne die Hinzunahme eines annotierten Testkorpus entwickelt. Hier lag der Fokus auf dem Grammatikdesign und darauf, wie dynamische Merkmale und Hilfsconstraints am besten einzusetzen sind und wie spezielle sprachliche Phänomene wie beispielsweise Ellipsen oder die Koordination am geeignetsten zu behandeln und modellieren sind. Die dynamischen Merkmale erwiesen sich hierbei als sehr hilfreich, um präzise Regeln zu formulieren und um die Grammatik zu strukturieren. Nachdem ein grundlegendes Regelwerk erstellt worden war, wurde dieses anhand von tausend Sätzen eines annotierten Testkorpus weiterentwickelt. Hierbei wurden vermehrt weiche Constraints hinzugefügt und es wurden die Gewichtungen der Regeln optimiert. Hierzu diente die Dependenzversion des NEGRA-Korpus, die über das Werkzeug *DEPSY* (Dependency Synthesizer) erzeugt wurde (Daum et al., 2004).

Grundsätzlich ist die Grammatik nicht für die Disambiguierung von Wortarten entwickelt worden. So wurden bei der Grammatikentwicklung bereits disambiguierte STTS-Tags angenommen (Gold Part-of-Speech Tags). Die Grammatik ist über mehrere Dokumente organisiert, die separat bestimmte Label oder Satz- und Wortgruppierungstypen behandeln. Insgesamt werden 26 dynamische Merkmale verwendet, die zusammen 142 Merkmalswerte besitzen. Die Anzahl an verschiedenen Constraints, die über die Grammatik kompiliert werden, ist in Tabelle 5.3 zusammengefasst. Da es aus praktischen Gründen schwierig sein kann, komplexere Bedingungen in einen FSM zu überführen, wurde eher versucht, viele kleine Regeln zu formulieren.

Es wurden nicht alle Constraints in einen monolithischen WFST kompiliert. Die Berechnung war auf der vorliegenden Hartware mit 16 Gigabyte

Typ	Anzahl
harter Constraint	1.764
weicher Constraint	935
Has-Constraint	64
Is-Constraint	41
XHas-Constraint	6
XIs-Constraint	3

Tabelle 5.3: Constraint-Anzahl.

RAM nicht möglich. Das Zusammenfassen der harten Constraints und der Hilfsconstraints wäre aber auf einer größeren Rechenmaschine denkbar. So wurde gemäß Abschnitt 4.5.8 im Rahmen des Aufspaltens simultaner lokaler Konsistenzregeln nur ein Teil der Constraints in einen monolithischen WFST zusammengefasst. Die entsprechende simultane lokale Konsistenzregel hat eine Größe von 5.341.864 Zuständen und 29.327.719 Übergängen. Der Rest wurde in separate WFSTs in Form von Regelbestandteilen kompiliert. Die Bestandteile umfassen insgesamt 706 harte Constraints und Hilfsconstraints mit durchschnittlich 9.058 Zuständen und 182.071 Übergängen und 935 weiche Constraints mit durchschnittlich 24.972 Zuständen und 460.396 Übergängen.

5.5 Grammatikformat

In diesem Abschnitt wird das Format beschrieben, das der Grammatik, die für die Erprobung des neuen Ansatzes entwickelt wurde, zugrunde liegt. Hierbei wird auf Einzelheiten in der Grammatikentwicklung Bezug genommen.

Eine Grammatik besteht aus XML-Dokumenten. Über diese können Constraints mithilfe von speziellen XML-Container-Elementen definiert werden. Ihr grundlegender Aufbau ist in Abbildung 5.3 gegeben. Ein XML-Dokument enthält als Wurzel das XML-Container-Element '*grammar*' und erlaubt über das XML-Container-Element '*include*' die Einbindung anderer XML-Dokumente. Auf diese Weise ist es möglich, die Grammatikregeln auf mehrere Dateien aufzuteilen. Im Folgenden wird erst auf die Prädikate eingegangen, die innerhalb von regulären prädikatenlogischen Formeln verwendet werden können. Danach werden die XML-Container beschrieben, die die Definitionen von weichen und harten Constraints, von Hilfsconstraints, von Teilaus-

```
<grammar>
  <include> Datei_1 </include>
  <include> Datei_2 </include>
       ⋮
  <include> Datei_n </include>
  ...
</grammar>
```

Abbildung 5.3: Grundlegender Aufbau einer Grammatik-Spezifikation.

drücken und von aufgespaltenen Dependenzknoten-Teilproblemen ermöglichen.

5.5.1 Prädikate

Innerhalb der Grammatik werden mehrere Prädikate verwendet. Diese beziehen sich auf Oberflächeninformationen, auf lexikalische Informationen, auf Informationen über Hilfsconstraints und dynamische Merkmale und auf syntaktische Informationen, d.h. ob ein Wort beispielsweise ein Governor oder ein Modifier ist. Die einzelnen Prädikate sind in Tabelle 5.4 aufgeführt.

5.5.2 Harte und weiche Constraints

Innerhalb der Grammatik werden die harten und weichen Constraints über das XML-Container-Element *rule* angegeben.

Ein Beispiel für die Definition eines harten Constraints ist in Abbildung 5.4 gegeben. Dort wird über das XML-Attribut *name* der Name der Regel

```
<rule name="DET-STTS-Kategorie" score="0.0">
  ∀x Gov(x) ∧ Lab(x,DET) →
    Cat(x,{ART,PWAT,PPOSAT,PDAT,PIAT,PRELAT})
</rule>
```

Abbildung 5.4: Beispiel für die Definition eines harten Constraints.

angegeben und über das XML-Attribut *score* das Strafgewicht bei einer Verletzung. Das Strafgewicht beträgt hier 0,0 und definiert somit einen harten Constraint. In diesem Beispiel ist eine Beschränkung umgesetzt, die besagt, dass das `DET`-Label nur bei Wörtern mit einer bestimmten STTS-Kategorie auftritt.

Prädikat	**Beschreibung**
Allgemeine Informationen	
$Var(x)$	x ist eine Variable
Wortforminformationen	
$Word(x)$	x ist ein Wort des Eingabesatzes
$Empty(x)$	x ist ein leeres Wort
$Dummy(x)$	x ist ein Dummy-Wort
$Cap(x)$	x ist ein kapitalisiertes Wort
$Word(x, \alpha)$	x besitzt die Wortform α
Lexikalische Informationen	
$Lex(x)$	x besitzt einen Lexikoneintrag
$LGuess(x)$	x besitzt einen erratenen Lexikoneintrag
$Unnown(x)$	x besitzt keinen Lexikoneintrag
$Cat(x, \alpha)$	x besitzt den STTS-Tag α
$CFeat(x, \alpha)$	x besitzt das lexikalische Merkmal α
$CFeat(x, \alpha, \beta)$	x besitzt das lexikalische Merkmal α mit dem Wert β
$Base(x, \alpha)$	x besitzt die Lemmaform α
Syntaktische Informationen	
$Gov(x)$	x ist ein Governor
$Mod(x)$	x ist ein Modifier
$Xing(x)$	x ist ein kreuzendes Wort
$XGov(x)$	x ist ein kreuzendes Wort, das in dem direkt folgendem Dependenzknoten-Teilproblem Governor ist
$XMod(x)$	x ist ein kreuzendes Wort, das in dem direkt folgendem Dependenzknoten-Teilproblem Modifier ist
$Root(x)$	x ist dem Dummy-Wurzelknoten direkt untergeordnet
$Crossed(x)$	die Kante unterhalb von x wurde gekreuzt
Hilfsconstraints	
$Is(x, \alpha)$	Prüfen des Is-Constraints α oberhalb von x
$Has(x, \alpha)$	Prüfen des Has-Constraints oberhalb von x
$XIs(x, \alpha)$	Prüfen des XIs-Constraints α oberhalb von x
$XHas(x, \alpha)$	Prüfen des XHas-Constraints unterhalb von x
Dynamische Merkmale	
$DFeat(x, \alpha)$	x besitzt das dynamische Merkmal β
$DFeat(x, \alpha, \beta)$	x besitzt das dynamische Merkmal α mit dem Wert β
Aufgespaltenes Dependenzknoten-Teilproblem	
$Cut(x)$	x ist/war Governor innerhalb eines zweiteilig aufgespaltenen Dependenzknoten-Teilproblems

Tabelle 5.4: Prädikate.

Bei der Definition von weichen Constraints kann zusätzlich angegeben werden, ob das Verletzen oder das Nichtverletzen nachverfolgt werden soll. Ersteres wird bei der Umsetzung von verletzbaren Regeln angewendet und Letzteres bei der Modellierung von Präferenzen.

Verletzbare Regeln werden über reguläre prädikatenlogische Formeln ausgedrückt, die einen initialen Allquantor besitzen. Ein Beispiel hierfür ist in Abbildung 5.5 gegeben. Dort beträgt das Strafgewicht 0,2 und definiert somit

```
1 <rule name="DET-Num-Kongruenz" score="0.2" trace="negative">
2   ∀xy Gov(x) ∧ CFeat(x,Number) ∧
3       Mod(y) ∧ Lab(y,DET) ∧ CFeat(y,Number) →
4         CFeat(x,Number,sg) ∧ CFeat(y,Number,sg) ∨
5         CFeat(x,Number,pl) ∧ CFeat(y,Number,pl)
6 </rule>
```

Abbildung 5.5: Weicher Constraint: verletzbare Regel.

einen weichen Constraint. Mithilfe des XML-Attributs *trace* und dem Wert *negative* ist angegeben, dass ein Verletzen nachverfolgt werden soll. Über die Bedingung wird geprüft, ob eine Numerus-Kongruenz zwischen dem Determiner und dem übergeordneten Wort besteht. Wenn diese verletzt wird, dann wird auch eine Verletzung angezeigt.

Präferenzen werden hingegen über reguläre prädikatenlogische Formeln mit initialem Existenzquantor ausgedrückt. Ein Beispiel hierfür ist in Abbildung 5.6 gegeben. Dort beträgt das Strafgewicht 0,8 und definiert einen

```
1 <rule name="präferiere DET" score="0.8" trace="positive">
2   ∃x Gov(x) ∧ ¬Root(x) ∧ Lab(x,DET)
3 </rule>
```

Abbildung 5.6: Weicher Constraint: Präferenz.

weichen Constraint. Mithilfe des XML-Attributs *trace* und dem Wert *positive* ist angegeben, dass ein Nichtverletzen nachverfolgt werden soll. Über die Bedingung wird gefordert, dass der Governor die Funktion eines Determiners besitzt und nicht an den Dummy-Wurzelknoten angebunden ist. Wenn diese nicht verletzt wird, dann wird eine Präferierung angezeigt.

5.5.3 Hilfsconstraints

Innerhalb der Grammatik wird ein Hilfsconstraint über das XML-Container-Element *ancilla* definiert, wobei der Hilfsconstraint-Typ über ein XML-Attribut angegeben wird.

Ein Beispiel für einen Hilfsconstraint ist in Abbildung 5.7 gegeben. Dort

```
1 <ancilla name="Determiner" type="has">
2     Cat(⋆,{NN,NE}) ∧ ∃x Mod(x) ∧ Lab(x,DET)
3 </ancila>
```

Abbildung 5.7: Beispiel für die Definition eines Hilfsconstraints.

wird über das XML-Attribut *type* und dem Wert *has* ein Has-Constraint definiert und über das XML-Attribut *name* der Name des Constraints angegeben. Die Bedingung ist in Form einer Teilformel gegeben. Das Zeichen $\star$ dient hierbei als Platzhalter für die Variable, von der aus der Hilfsconstraint aufgerufen wird (siehe hierzu Abschnitt 4.5.12). Über den Hilfsconstraint wird abwärts geprüft, ob die entsprechende Variable für ein Nomen oder einen Eigennamen steht und einen Determiner besitzt.

Es ist auch möglich, Hilfsconstraints rekursiv zu definieren. Hierfür ist in Abbildung 5.8 für einen Is-Constraint ein Beispiel gegeben. Dort wird

```
1 <ancilla name="Subjekt" type="is">
2     (Lab(⋆,AUX) ∧ ∃x Mod(x) ∧ Lab(x,{SUBJ,SUBJC})) ∨
3     (Lab(⋆,AUX) ∧ ∃x Gov(x) ∧ Lab(x,AUX) ∧ Is(x,Subjekt))
4 </ancilla>
```

Abbildung 5.8: Beispiel für die rekursive Definition eines Hilfsconstraints.

innerhalb einer Bedingung über eine `AUX`-Kette hinweg geprüft, ob ein Satz ein Subjekt besitzt. Der rekursive Aufruf geschieht hierbei in Zeile drei ($Is(x, Subjekt)$).

Has-Constraints dienen in der vorliegenden Grammatik dafür, eine Struktur abwärts auf Determiner, Partikel und Pronomen und andere Wortarten zu testen. Es wird aber auch geprüft, ob Nichtprojektivität vorliegen. So wird beispielsweise innerhalb einer AUX-Kette überprüft, ob das Subjekt zu einer nichtprojektiven Struktur führt. Des Weiteren wird getestet, ob abwärts bestimmte rekursive Strukturen auftreten.

Is-Constraints werden beispielsweise dazu verwendet, aufwärts zu prüfen, ob sich ein Wort oder eine Wortgruppe im Vorfeld oder Nachfeld befindet oder in Hochkommata oder in Klammern steht. Es wird aber auch getestet, ob bei Präpositionalgruppen und vergleichenden Wortgruppen aus Sicht des Komplements oder des Konjunktes eine bestimmte Präposition oder vergleichende Konjunktion vorliegt.

XHas-Constraints werden herangezogen, um beispielsweise abwärts zu testen, ob eine Kantenüberkreuzung durch eine infinite Verbalgruppe verursacht wird oder ob eine Überkreuzung mit einer Kante existiert, die Nebensätze oder Infinitivobjekte, Vergleichsstrukturen oder Appositionen anbinden. Über die XIs-Constraints wird dies hingegen aufwärts geprüft.

5.5.4 Dynamische Merkmale

Innerhalb der Grammatik werden dynamische Merkmale über das XML-Container-Element *feature* angegeben. Ein Beispiel hierfür ist in Abbildung 5.9 gegeben. Dort wird über die XML-Attribute *name* und *value* der Merkmals-

```
1 <feature name="NT" value="np ap pp cp avp vp ip ns s undef">
2   Var(⋆) ∧ ¬Lab(⋆,'')
3 </feature>
```

Abbildung 5.9: Beispiel für die Definition eines dynamischen Merkmals.

name mit seinen möglichen Werten angegeben. Die Bedingung für die Zuweisung des Merkmals ist über eine Teilformel definiert (siehe hierzu Abschnitt 4.5.12). Das Zeichen ⋆ dient als Platzhalter für die Variable der das Merkmal zugewiesen werden soll. Hier wird das Merkmal `NT` allen Token zugeordnet, die keine Satzzeichen darstellen. Mögliche Merkmalswerte sind Nominalgruppe (`np`), Adjektivgruppe (`ap`), Präpositionalgruppe (`pp`), vergleichende Wortgruppe (`cp`), Adverbgruppe (`avp`), infinite Verbalgruppe (`vp`), Infinitivgruppe (`ip`), Nebensatz (`ns`) und Satz (`s`). Über dieses Merkmal wird in der Grammatik eine grobe Strukturierung der Dependenzknoten-Teilprobleme umgesetzt.

Bei der Definition eines Merkmals können andere dynamische Merkmale einbezogen werden. Ein Beispiel hierfür ist in Abbildung 5.10 gegeben. Dort wird das Merkmal `AdpoType`, das die Wortart des untergeordneten Wortes einer Adposition angibt, ausschließlich mit Präpositionalgruppen verknüpft

```
1 <feature name="AdpoType" value="adj adv prep pron sub">
2   DFeat(*,NT,pp)
3 </feature>
```

Abbildung 5.10: Beispiel für die Definition eines dynamischen Merkmals, die ein anderes dynamisches Merkmal einbezieht.

(`pp`). Die möglichen Merkmalswerte sind Adjektiv (`adj`), Adverb (`adv`), Präposition (`prep`), Pronomen (`pron`) und Substantiv (`sub`). Analog zu diesem wird innerhalb der Grammatik auch ein Merkmal für vergleichende Wortgruppen (als und wie) herangezogen.

Da über die dynamischen Merkmale in gewisser Weise feinkörnigere Dependenz-Label implementiert werden, wird im Folgenden auch von *verfeinerten Dependenz-Labeln* gesprochen. Hierdurch können Beschränkungen gezielt für die einzelnen Phänomene formuliert werden. Neben den oben genannten Merkmalen werden innerhalb der vorliegenden Grammatik noch weitere Merkmale einbezogen. Die wichtigsten sind im Folgenden kurz aufgeführt.

Über ein spezielles Merkmal wird jedem Wort eine STTS-unabhängige grobkörnige Wortartenkategorie zugewiesen. Mögliche Kategorien sind beispielsweise Vollverb, Hilfsverb, Modalverb, Kopulaverb, Substantiv, Adposition und Adjektiv. Hierbei wird im Gegensatz zum STTS zwischen Hilfsverb und Kopulaverb unterschieden.

Das Label für die Anbindung von Konjunkten `CJ` enthält keine Information darüber, welche Funktion die gesamte koordinierte Struktur besitzt. Aus diesem Grund wird ein Merkmal herangezogen, das diese bereitstellt.

Im Kontext der topologischen Felder wird ein Merkmal für die Satzstellung verwendet. Dieses unterscheidet zwischen Verberst-, Verbzweit- und Verbletztsatz.

Um die Subkategorisierungsrahmen von Verben angemessen behandeln zu können, wird in der Grammatik ein Merkmal für das Genus Verbi verwendet. Die möglichen Merkmalswerte umfassen das Lassen-, Werden- und Bekommen-Passiv, das modale Passiv und das Aktiv.

Es wird ein Merkmal herangezogen, das den Typ eines Nebensatzes angibt. Hierbei wird zwischen Konjunktionalsatz mit subordinierender Konjunktion, Relativsatz mit Relativpronomen oder Relativadverb, Interrogativsatz mit Interrogativpronomen oder Interrogativadverb und uneingeleitetem Nebensatz unterschieden.

Im Kontext der Infinitivkonstruktionen wird ein Merkmal herangezogen, dass zwischen einem Infinitiv mit zu, einem AcI (accusativus cum infinitivo) und einem phraseologisch festgelegten Infinitiv in Fügungen wie „stehen bleiben“ oder „einkaufen gehen“ differenziert.

Es existieren verschiedenste Phänomene, die unter dem Label `APP` zusammengefasst sind. Daher wird ein Merkmal herangezogen, das zwischen diesen unterscheidet. Dies sind beispielsweise verschiedene Typen des Nomen varians und invarians oder die klassische Apposition.

5.5.5 Weiter aufgespaltene Teilprobleme

Ein Dependenzknoten-Teilproblem kann in Sub-Dependenzknoten-Teilprobleme aufgespalten werden (siehe hierzu Abschnitt 4.5.11). Hier ist die Aufspaltung auf zwei Teilprobleme begrenzt. Das aus Bottom-Up-Sicht initiale wird über das XML-Containter-Element *cut* angegeben. Ein Beispiel hierfür ist in Abbildung 5.11 gegeben. Dort wird der Fall betrachtet, dass innerhalb

```
1 <cut>
2   ∃x Gov(x) ∧ DFeat(x,NT,np) ∧ Cat(x,ADJA)
3 </cut>
```

Abbildung 5.11: Initiales Sub-Dependenzknoten-Teilproblem.

einer Nominalgruppe in bestimmten Konstruktionen das Nomen reduziert werden kann. Dies ist möglich, wenn ein attributives Adjektiv dessen Rolle einnimmt. Ein konkretes Beispiel ist der Satz „Wir lieben den roten Stift und hassen den ziemlich blauen“. Die Definition besagt, dass das Adjektiv (`ADJA`) innerhalb der Nominalgruppe (`np`) die Funktion des Governors einnimmt. Aus Bottom-Up-Sicht werden anhand dieser Definition alle Modifier des entsprechenden Teilproblems ‘abgeschnitten’, sodass sie für die Folgeprobleme nicht mehr sichtbar sind.

Über das Prädikat $Cut/1$ kann innerhalb einer Regel geprüft werden, ob ein Governor vorliegt, der auch innerhalb eines initialen Sub-Konten-Teilproblems Governor ist. Falls ein Teilproblem solch einen enthält, ist es automatisch das finale Sub-Konten-Teilproblem.

5.5.6 Teilausdrücke

Es ist möglich, innerhalb der Grammatik Teilausdrücke für die regulären prädikatenlogischen Formeln zu definieren. Hierdurch ist ein übersichtlicheres und strukturierteres Formulieren von Constraints möglich und es können oft verwendete Bedingungen gekapselt und wiederverwendet werden. Ein Teilausdruck wird über das XML-Element *expression* angegeben. Ein Beispiel hierfür ist in Abbildung 5.12 gegeben. Dort wird über das XML-Attribut

```
1 <expression name="Final">
2   ¬∃x Var(x) ∧ ⋆1 ≺ x
3 </expression>
```

Abbildung 5.12: Beispiel für die Definition eines Teilausdrucks.

name der Name des Teilausdrucks angegeben. Der Ausdruck ist einstellig und das Zeichen $\star_1$ dient als Platzhalter für eine einzusetzende Variable. Er besagt, dass sich ein gegebenes Wort an der rechtesten Position eines Dependenzknoten-Teilproblems befindet. Über die Platzhalter $\star_1, \star_2, \ldots, \star_n$ sind auch mehrstellige Teilausdrücke möglich.

Interessant sind Teilausdrücke in Kombination mit dynamischen Merkmalen. Ein Beispiel hierfür ist in Abbildung 5.13 gegeben. Dort ist über den

```
1 <expression name="@ObjiMitZu">
2   Lab(⋆1,OBJI) ∧ DFeat(⋆1,NT,ip) ∧ DFeat(⋆1,IpType,inf_mit_zu)
3 </expression>
```

Abbildung 5.13: Teilausdruck mit dynamischen Merkmalen.

Teilausdruck ein verfeinertes `OBJI`-Label umgesetzt. Hierbei wird innerhalb der Definition über das dynamische Merkmal `IpType` ein konkreter Infinitiv-Typ angegeben. Teilausdrücke dieser Art werden in der Grammatik ausgiebig dazu verwendet, verfeinerte Label einzubinden.

5.6 Modellierung sprachlicher Phänomene

Da innerhalb des Zerlegungsansatzes ein anderer Beschränkungsbereich angenommen wird als bei dem WCDG-Ansatz, müssen manche sprachliche Phänomene auch anders behandelt werden. Innerhalb des WCDG-Ansatzes kön-

nen zwei Kanten angefasst werden, die in keiner Unterordnungs- oder Schwesterbeziehung stehen und sich nicht überkreuzen. Innerhalb des Zerlegungsansatzes ist dies nicht möglich. Zudem steht nur eine sprachliche Ebene zur Verfügung, und zwar die syntaktische. So fehlt beispielsweise eine referenzielle Ebene, um ein Relativpronomen mit seinem Bezugswort direkt in Verbindung zu bringen. In diesem Abschnitt wird für bestimmte sprachliche Phänomene gezeigt, wie diese trotz dieser Einschränkungen angemessen behandelt werden können. Hierbei können einige sogar adäquater angegangen werden als innerhalb der WCDG-Grammatik für das Deutsche. Dies betrifft beispielsweise diskontinuierliche Strukturen, Koordinationsellipsen und strukturelle Ellipsen. Im Folgenden werden die einzelnen sprachlichen Phänomene aufgeführt, bei denen in ihrer grundlegenden Modellierung eine Abweichung von der WCDG-Grammatik besteht.

5.6.1 Satzzeichen

Innerhalb der WCDG-Grammatik für das Deutsche werden Satzzeichen dem Dummy-Wurzelknoten direkt untergeordnet. Dies ist innerhalb des Problemzerlegungsansatzes jedoch problematisch, da die Satzzeicheninformation für die Formulierung von Beschränkungen innerhalb der Teilprobleme benötigt wird. Aus diesem Grund werden Satzzeichen, die in eine Koordination eingebunden sind, den Konjunkten untergeordnet (siehe Abschnitt 5.6.5) und solche, die einen Nebensatz oder eine Wortgruppierung abtrennen, dem Wort, an das auch der abgetrennte Teil angebunden ist. Eine Ausnahme bilden Matrixsätze, die in ihrem Objektsatz eingebettet sind. Dort werden sie direkt dem Objektsatz untergeordnet (siehe Abschnitt 5.6.3). Des Weiteren werden Satzschlusszeichen an das finite Verb des Hauptsatzes angebunden. Falls dies nicht möglich ist, an den Dummy-Wurzelknoten. Generell werden Satzzeichen analog zur WCDG-Grammatik über das leere Label behandelt. Ein Beispiel hierfür ist in Abbildung 5.14 für den Satz „Sie liebt einen Appenzeller, weil er niedlich ist.“ gegeben.

Ein Satzzeichen ist genau einem Teilproblem zugeordnet. Manchmal wird dieselbe Satzzeicheninformation aber gleichzeitig in verschiedenen Dependenzknoten-Teilproblemen benötigt. Ein Beispiel hierfür ist der Satz „Sie liebt ihn, einen Appenzeller.“ Ohne Hinzunahme überlokaler Constraints ist es hier nicht möglich zu fordern, dass eine weite Apposition durch ein Satzzeichen

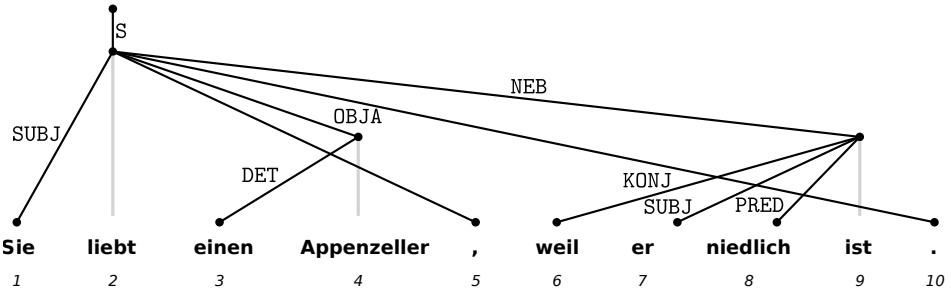


Abbildung 5.14: Satzzeichenanbindung.

nach rechts hin abgetrennt sein muss, da das finale Satzzeichen bereits dem finiten Verb des Hauptsatzes untergeordnet ist. Dasselbe Problem tritt auch bei verschachtelten Nebensätzen auf.

In der initialen Grammatikentwicklung wurden Hilfsconstraints herangezogen, um Satzzeicheninformationen gleichzeitig in mehreren Dependenzknoten-Teilproblemen verfügbar zu machen. Jedoch erwies sich die Regelformulierung insbesondere im Kontext von nichtprojektiven Strukturen als sehr aufwendig und umständlich. Eine andere Möglichkeit ist die Einbeziehung virtueller Knoten in Form von Dummy-Knoten. Über diese ist es möglich, einzelne Satzzeichen zu 'kopieren', sodass innerhalb verschiedener Dependenzknoten-Teilprobleme auf diese Kopien zugegriffen werden kann. Hierzu wird optional ein ausreichender Vorrat an Dummy-Knoten direkt vor oder nach den einzelnen Satzzeichen angenommen. Mit jedem Dummy-Knoten ist hierbei die Information verknüpft, ob es sich dabei um ein Komma, ein Punkt, ein Fragezeichen und so weiter handelt. Um dieses Vorgehen zu verdeutlichen, ist für den obigen Beispielsatz „Sie liebt ihn, einen Appenzeller." in Abbildung 5.15 eine Beispieldependenzstruktur gegeben. Dort wird direkt vor dem finalen Satzpunkt eine Dummy-Variable angenommen, die durch $\theta_{.}$ gekennzeichnet ist. Hierdurch kann ohne Hinzunahme von überlokalen Constraints gefordert werden, dass einer weiten Apposition stets ein Satzzeichen folgt.

5.6.2 Appositionen

Unter dem Label `APP` vereinen sich innerhalb der WCDG-Grammatik für das Deutsche verschiedene Phänomene, die im weiteren Sinne Appositionen darstellen. Dort werden sie über eine rechtsverzweigende `APP`-Kette modelliert.

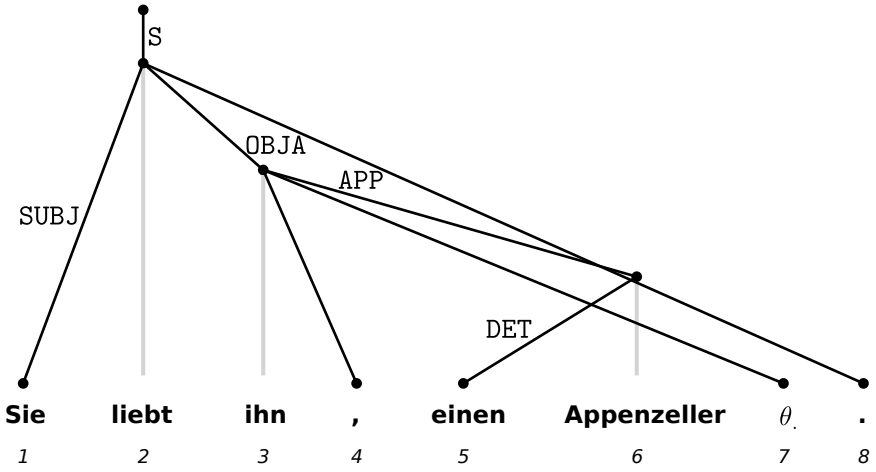


Abbildung 5.15: Satzzeichenanbindung und Dummy-Knoten.

Jedoch ist das erste Wort in einer APP-Kette nicht immer der syntaktische Kopf. Dies ist beispielsweise beim vorangestellten Nomen invarians der Fall. Aus diesem Grund kann bei einer rechtsverzweigenden Kette von außen nicht direkt auf die relevanten lexikalischen Informationen des syntaktischen Kopfes zugegriffen werden.

In der hier erarbeiteten Grammatik werden die verschiedensten Phänomene, die sich unter dem APP-Label vereinen, über ein dynamisches Merkmal unterschieden. Hierdurch wird eine gezieltere Formulierung von Beschränkungen möglich. Je nach Phänomen steht der Governor dann rechts oder links, wobei nicht immer eine Kette modelliert wird. Ein Beispiel hierfür ist das Nomen invarians in „Bundeskanzler Walter Thurnherr“ und das durch Leerzeichen getrennte (ungrammatische) Kompositum „Sechs Korn Müsli“. Beide Dependenzstrukturen sind zur Verdeutlichung in Abbildung 5.16 gegeben. Dort ist die Modellierung, wie sie in der WCDG-Grammatik angenommen wird, ausgegraut mit gestrichelten Linien gekennzeichnet. Die klassische Apposition und das nachgestellte Nomen invarians und varians werden hingegen wie in der WCDG-Grammatik von rechts angebunden.

5.6.3 Nichtprojektivität

Innerhalb der WCDG-Grammatik für das Deutsche werden nichtprojektive Strukturen als Ausnahme behandelt und nur für bestimmte sprachliche Phänomene erlaubt. Des Weiteren wird Nichtprojektivität in manchen Fällen

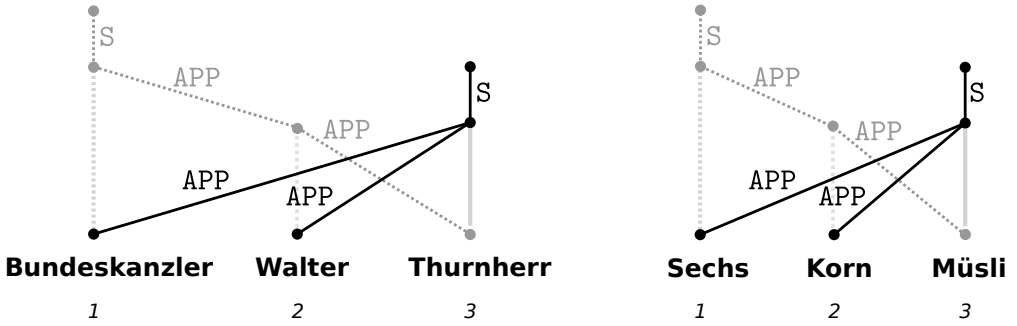


Abbildung 5.16: Linksanbindung über das APP-Label.

dadurch vermieden, dass Wörter höher im Baum angebunden oder einem vorangehenden Terminalknoten untergeordnet werden. Das bedeutet aber nicht, dass der WCDG-Formalismus mit den entsprechenden nichtprojektiven Strukturen nicht umgehen könnte. Dies ist lediglich eine Design-Entscheidung. In der hier entwickelten Grammatik ist im Vergleich zur WCDG-Grammatik mehr Nichtprojektivität einbezogen. Dies hat folgende Gründe:

- Es soll verdeutlicht werden, dass das Problemzerlegungsverfahren für die Einbeziehung von Nichtprojektivität eine gute Wahl ist.
- Es soll gezeigt werden, dass der Regelformalismus geeignet ist, Beschränkungen über nichtprojektive Strukturen zu formulieren.
- Wenn Nichtprojektivität zugelassen wird, ist es in vielen Fällen einfacher, präzise Regeln zu schreiben.

Im Folgenden sind verschiedene sprachliche Phänomene zusammengefasst, die sich auf Vergleichsstrukturen, Korrelate, Topikalisierung und Parenthesen beziehen und Nichtprojektivität verursachen.

Vergleichsstrukturen

Bei der Anbindung von vergleichenden Wortgruppen oder Vergleichssätzen über das Label KOM werden innerhalb der WCDG-Grammatik für das Deutsche nichtprojektive Strukturen systematisch vermieden. Ein konkretes Beispiel hierfür ist in Abbildung 5.17 für den Satz „Weil sie größer ist als ein Graubündener“ gegeben. Dort wird die vergleichende Wortgruppe nicht an sein Bezugswort, sondern an das Kopulaverb und somit höher angehängt.

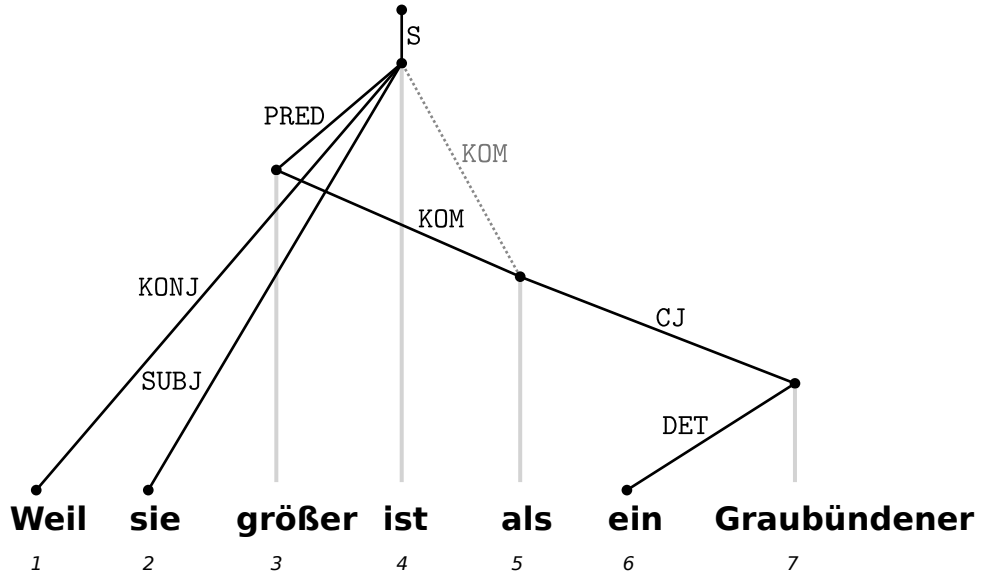


Abbildung 5.17: Nichtprojektivität bei Vergleichen.

Dies ist durch die ausgegraute gestrichelte Linie angezeigt. Analog hierzu werden in der WCDG-Grammatik vergleichende Wortgruppen oder Vergleichssätze nicht linksseitig an attribuierende Indefinitpronomen und Adjektive angebunden sondern an das übergeordnete Nomen, sodass es zu keiner nichtprojektiven Struktur kommt. In der hier entwickelten Grammatik wird solch eine Höheranbindung nicht angenommen.

Korrelate

Innerhalb der WCDG-Grammatik für das Deutsche werden Nebensätze, die durch ein Korrelat vertreten sind, nicht an dieses angebunden. Sie werden stets höher im Dependenzbaum eingehängt und dem Verb untergeordnet. Dies betrifft Korrelate wie Personalpronomen (es, das), Pronominaladverbien (darüber, darauf, damit, ...) oder Adverbien (so, derart, dermaßen, ...).

Ein Beispiel hierfür ist der Satz „Dafür kämpfe ich, dass die Kuhhörner bleiben“, bei dem das Korrelat in Form eines Pronominaladverbs Platzhalter für einen Nebensatz ist, der als Präpositionalobjekt fungiert. Innerhalb der WCDG-Grammatik würde der Teilsatz über das Label `NEB` dem finiten Verb des Matrixsatzes untergeordnet werden. Ein anderes Beispiel ist der Satz „Sie hat es geliebt, das Matterhorn zu besteigen“. Dort ist das Korrelat ein Personalpronomen und der dazugehörige Nebensatz hat die Funktion eines Ak-

kusativobjektes. In diesem Fall würde die WCDG-Grammatik das Korrelat über das Label EXPL und den Teilsatz über das Label OBJI parallel dem Vollverb unterordnen. In der hier entwickelten Grammatik wird der Nebensatz hingegen an das Korrelat angehängt. Dies ist für den obigen Beispielsatz in Abbildung 5.18 verdeutlicht. Innerhalb des Dependenzbaums wird das Korre-

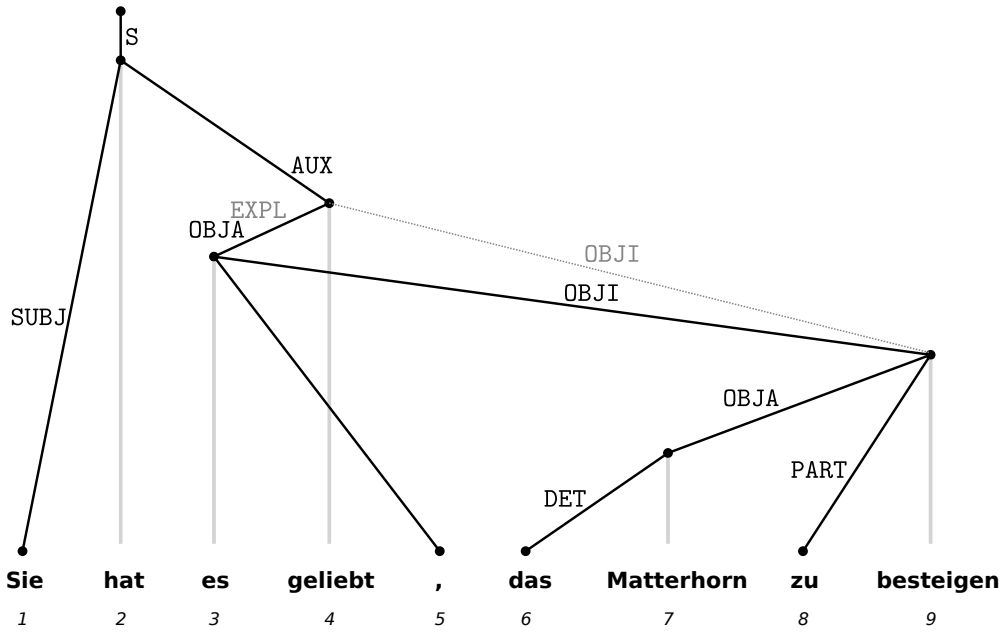


Abbildung 5.18: Nichtprojektivität bei Korrelaten.

lat über das Label OBJA dem Vollverb untergeordnet. Die Unterordnung, die innerhalb der WCDG-Grammatik angenommen wird, ist hingegen durch die ausgegraute gestrichelte Linie dargestellt. Ähnliches betrifft das Expletivum, das das Vorfeld besetzt und Platzhalter für ein normales Subjekt ist.

Hier wird generell auf das Label EXPL verzichtet. Für die Anbindung an das Korrelat wird das Dependenz-Label herangezogen, das von der WCDG-Grammatik für die Unterordnung unter das Verb vergeben würde. Hierdurch soll das Einführen neuer Label vermieden werden.

Topikalisierung

Im Rahmen der Topikalisierung werden innerhalb der WCDG-Grammatik für das Deutsche nichtprojektive Strukturen vermieden. Dies geschieht bei Modalverb- und Hilfsverbkonstruktionen. Es werden Strukturen, die keine

Objekte darstellen, wie beispielsweise Adverbiale, präpositionale Modifizierer und Vergleichsstrukturen dem finiten Verb und nicht dem Vollverb untergeordnet. Ein Beispiel hierfür ist in Abbildung 5.19 für den Satz „In der Schweiz hat sie gerne gelebt.“ gegeben. Die Unterordnung, die innerhalb der

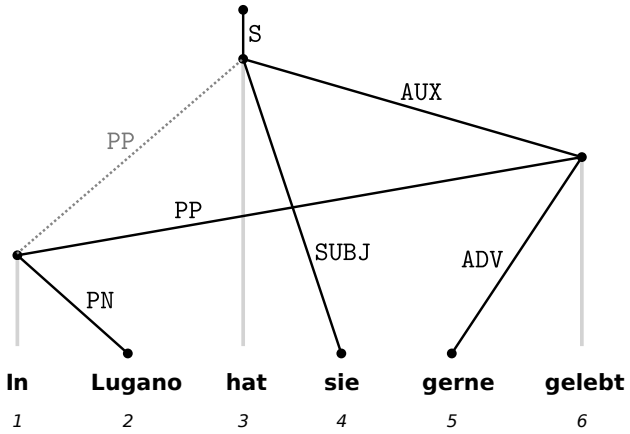


Abbildung 5.19: Nichtprojektivität bei der Topikalisierung.

WCDG-Grammatik angenommen wird, ist über die ausgegraute gestrichelte Linie dargestellt. In der hier entwickelten Grammatik wird auf solch eine Höheranbindung verzichtet.

Parenthesen

Wenn eine Einbettung eines Matrixsatzes in seinen untergeordneten Objektsatz vorliegt, wird innerhalb der WCDG-Grammatik eine nichtprojektive Struktur vermieden. Hierzu wird das Label `PAR` herangezogen, mit dem der Matrixsatz dem Wort direkt vor dem Einschub untergeordnet wird. Ein Beispiel hierfür ist der Satz „Der Hund, vermutet sie, ist ein Graubündener“. Dessen Dependenzstruktur ist in Abbildung 5.20 gegeben. Innerhalb der WCDG-Grammatik wird der Matrixsatz über das Label `PAR` an einen terminalen Knoten angehängt. Dies ist über die ausgegraute gestrichelte Linie dargestellt. In der hier entwickelten Grammatik wird hingegen eine nichtprojektive Struktur erlaubt und es wird auf das zusätzliche Label verzichtet. Hierzu wird der Objektsatz über das Label `S` an das Vollverb des Matrixsatzes angebunden. Um dies umzusetzen, wird einer Zerlegung über ein aufgespaltenes Dependenzknoten-Teilproblem nachgegangen. Der Objektsatz wird dann über das zweite Sub-Dependenzknoten-Teilproblem einbezogen.

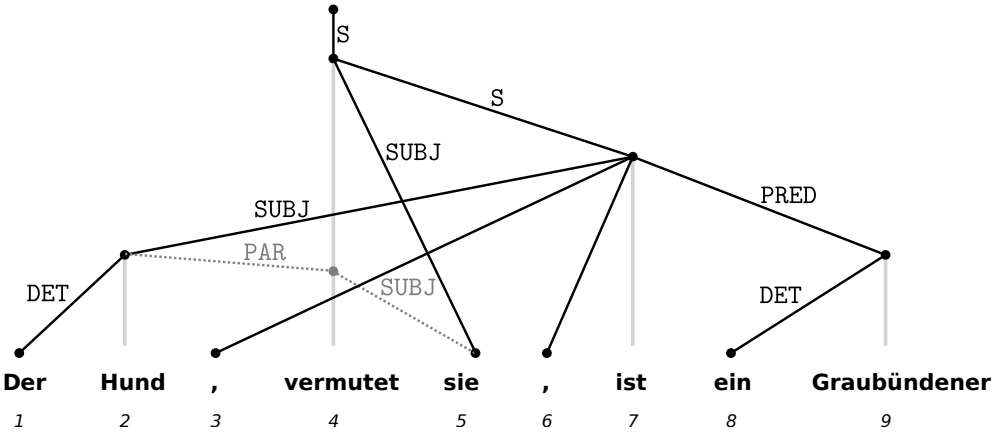


Abbildung 5.20: Nichtprojektivität bei der Einbettung eines Matrixsatzes in seinen Objektsatz.

5.6.4 Nebensatzeinleitende Pronomen und Adverbien

Interrogativpronomen und Interrogativadverbien können einen Objektsatz, einen Subjektsatz oder einen freien Relativsatz einleiten. Ein Beispiel hierfür ist der Satz „Er weiß, wo die Frau gelegen haben soll “. Hier fungiert der Nebensatz als Objekt und das Pronominaladverb ist weit in diesen eingebettet. Innerhalb des Dependenzknoten-Teilproblems, das das finite Hilfsverb des Nebensatzes enthält, besteht so kein direkter Zugriff auf die Pronominaladverb-Information. Daher wird hier vorgeschlagen, die Interrogativpronomen- oder Interrogativadverb-Information über Dummy-Knoten zu kopieren. Die Dummy-Knoten können dann dem finiten Verb des Nebensatzes untergeordnet werden. Die Dummy-Knoten werden mithilfe von virtuellen Knoten umgesetzt und initial vor jedem Interrogativpronomen oder Interrogativadverb angenommen. Diese müssen hierbei nicht zwingend adjazent sein. Es kann beispielsweise auch eine Präposition dazwischenliegen. Zur Verdeutlichung ist in Abbildung 5.21 eine Dependenzstruktur für den obigen Satz gegeben. Dort ist ein Dummy-Knoten angenommen, der durch θ_{wo} gekennzeichnet und an das finite Hilfsverb des Nebensatzes angebunden ist.

Analog zu dem obigen Vorgehen werden Dummy-Knoten auch bei Relativpronomen angenommen. Jedoch muss hier zusätzlich die Kasus-Numerus-Genus-Kongruenz mit dem Bezugswort berücksichtigt werden. Daher wird die Kasus-Numerus-Genus-Information des Relativpronomens über einen Dummy-Knoten kopiert und dem finiten Verb des Relativsatzes zugänglich ge-

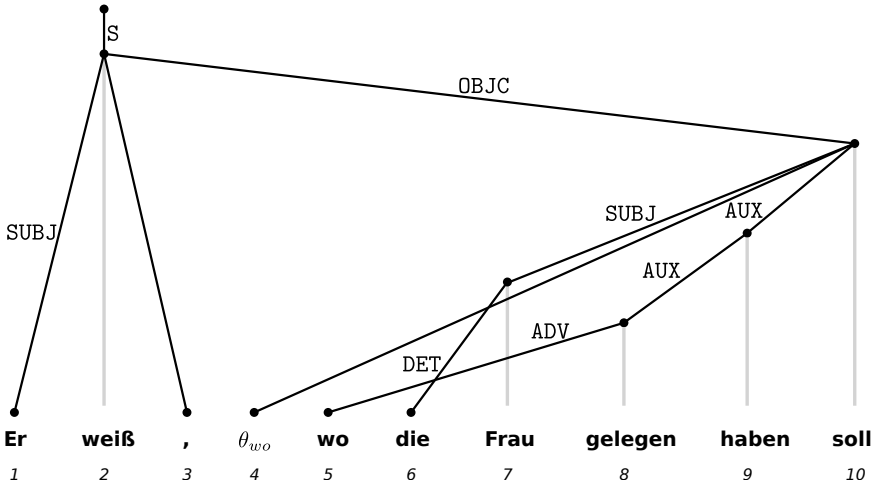


Abbildung 5.21: Behandlung von Interrogativpronomen und -adverbien über Dummy-Variablen.

macht. Die Kongruenz-Prüfung mit dem Bezugswort geschieht dann über ein entsprechendes dynamisches Merkmal, das dem finiten Verb zugeordnet ist.

5.6.5 Koordination

Innerhalb der Koordination sind die Konjunkte gleichwertig und können gleichermaßen die Konstruktion ersetzen. Dies macht eine Repräsentation über Unterordnungsbeziehungen und somit über Dependenzstrukturen schwierig. In Popel et al. (2013) werden verschiedene Repräsentationsmöglichkeiten der Koordination in unterschiedlichen Dependenzbaumbanken untersucht. Im Folgenden werden drei Repräsentationsformen vorgestellt, die für die Implementierung in Betracht gezogen wurden: rechtsverzweigende Konjunkt- und Konjunktionskette, Koordinationsblasen und rechtsverzweigende Konjunktkette. Hiervon hat sich die Letztere durchgesetzt und wurde final in der Grammatik umgesetzt.

Rechtsverzweigende Konjunkt- und Konjunktionskette

Innerhalb der WCDG-Grammatik für das Deutsche wird die Koordination als rechtsverzweigende Konjunkt- und Konjunktionskette umgesetzt. Eine entsprechende Dependenzstruktur ist für das Fragment „Er liebt weder Frau-

en und Kinder noch Schweizer“ in Abbildung 5.22 gegeben. Das Label `CJ`

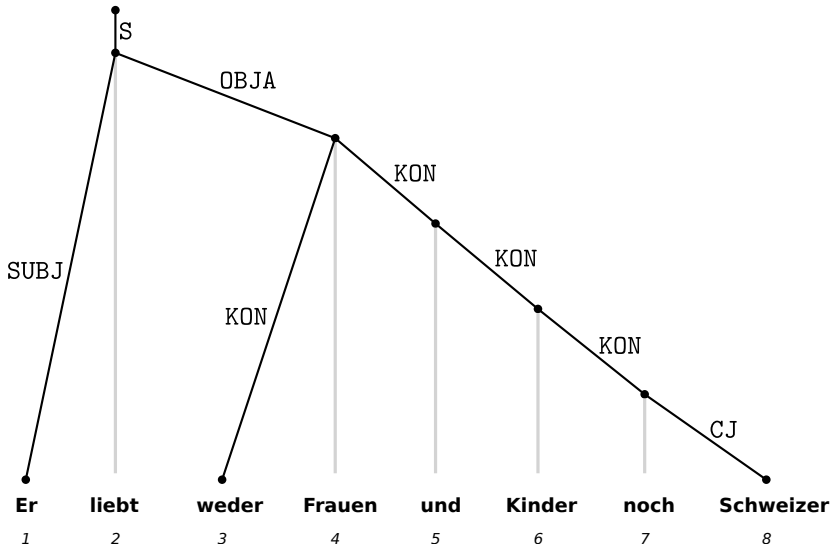


Abbildung 5.22: Koordination in den WCDG-Annotationsrichtlinien.

wird dem letzten Konjunkt der Kette zugewiesen. Das erste erhält die Funktion, die der koordinierten Struktur zuzuordnen ist. In diesem Beispiel ist dies die Akkusativobjektfunktion (`OBJA`). Die Konjunktionen und alle anderen Konjunkte erhalten das Label `KON`. Die Umsetzung dieser Darstellung ist innerhalb des Problemzerlegungsansatzes jedoch problematisch. Es ist hier ohne die Zuhilfenahme von überlokalen Constraints nicht möglich, mehrere Konjunkte gleichzeitig in ein Dependenzknoten-Teilproblem einzubeziehen. So kann beispielsweise nicht nativ geprüft werden, ob zwei Konjunkte die gleiche Wortart besitzen.

Koordinationsblasen

In Kahane (1997, Abschnitt 5.1) wird vorgeschlagen, die Elemente der Koordination innerhalb einer Dependenzstruktur zu einer *Koordinationsblase* (coordination bubble) zu gruppieren. Die Konjunkte besetzen dann über die Blase eine gemeinsame syntaktische Position. Diese kann hierbei iterativ sein. Dass heißt, dass eine unbegrenzte Anzahl an Konjunkten zu einer Blase gruppiert werden können. Sie kann aber auch rekursiv sein. Dies ist in parataktischen Konstruktionen wie „sowohl ... als auch“ oder „weder ... noch“ der Fall, bei denen eine Koordinationsblase eingebettet wird. Die Rekursivität ist hier

jedoch linguistisch stark eingeschränkt. In Beispiel 5.23 ist für den Beispielsatz aus dem vorhergehenden Abschnitt „Er liebt weder Frauen und Kinder noch Schweizer“ eine Dependenzstruktur mit zwei ineinander eingebetteten Koordinationsblasen dargestellt, wobei sowohl Rekursivität als auch Iterativität gegeben sind. In dem Beispiel fungiert die äußere Blase als Stellvertreter

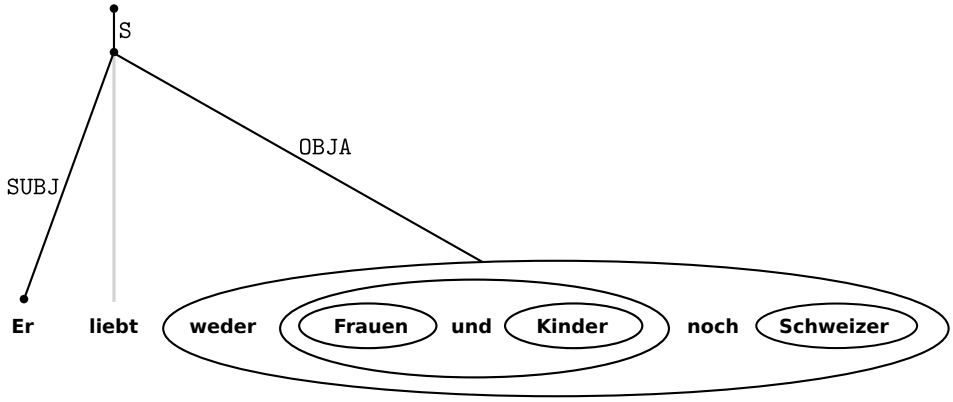


Abbildung 5.23: Iterativität und Rekursivität einer Koordinationsblase.

für die Koordinationsstruktur und besitzt die Funktion als Akkusativobjekt (`OBJA`).

Um die Koordinationsblasen über Unterordnungen darstellen zu können, wird hier vorgeschlagen, leere Knoten einzubeziehen. Diesen werden sowohl Konjunktionen als auch Konjunkte untergeordnet, wobei ein Konjunkt wiederum durch einen leeren Knoten repräsentiert sein kann. Zur Verdeutlichung dieser Repräsentationsform ist in Abbildung 5.24 eine Dependenzstruktur für das obige Koordinationsblasenbeispiel gegeben. Hierbei wird das Label `CJ` für die Anbindung der Konjunkte und das Label `KON` für die der Konjunktionen verwendet.

Die leeren Knoten werden über virtuelle Knoten repräsentiert, die initial an fast jeder Position des Eingabesatzes optional angenommen werden. Um auf die Wortinformationen der Konjunkte zugreifen zu können, muss deren lexikalische Information auf die virtuellen Knoten übertragen werden. Die entsprechenden Variablen sind dann mit Informationen verschiedener Wörter verknüpft. Dies kann dynamisch während des Bottom-Up-Lösens geschehen.

Um bei der Formulierung der Beschränkungen auf die unterschiedlichen Wortinformationen zugreifen zu können, wird das zweistellige Prädikat $Bubble/2$ vorgeschlagen. Dieses bezeichnet die Eigenschaft, eine Koordinationsblase zu

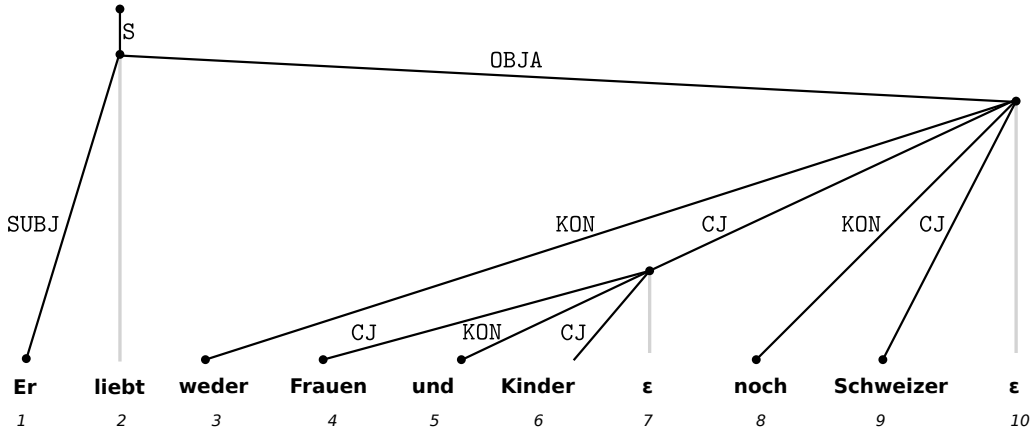


Abbildung 5.24: Repräsentation von Koordinierungsblasen über leere Knoten.

sein, die eine bestimmte lexikalische Wortinformation enthält. Die erste Stelle des Prädikats steht für den leeren Knoten und die zweite für die Information. Das bedeutet, dass hier nicht nur über Constraint-Variablen, sondern auch über Wortinformationen quantifiziert wird. Hierfür ist über den Ausdruck 5.1 ein Beispiel gegeben.

$$\forall x \forall y \; Mod(x) \wedge Bubble(x, y) \wedge Lab(x, \texttt{SUBJ}) \rightarrow Cat(y, \texttt{NN}) \tag{5.1}$$

Die Beschränkung besagt, dass eine Koordinationsblase ($Bubble/2$) ausschließlich Konjunkte der Wortart `NN` enthält ($Cat/2$), wenn sie Modifier ist ($Mod/1$) und als Subjekt ($Lab/2$) fungiert.

Diese Repräsentationsform ist zwar bezüglich der Gleichstellung der Konjunkte sehr interessant. Die Umsetzung innerhalb einer abdeckungsstarken Grammatik ist jedoch problematisch. In einer Phase der Grammatikentwicklung, die schon recht fortgeschritten war, führte die Einbeziehung der Koordinationsblasen zu sehr langen Kompilationszeiten und zu verhältnismäßig großen FSMs. Wenn zudem nicht kontinuierliche Koordinationsstrukturen einbezogen wurden (bezüglich solcher Strukturen siehe Prinzhorn und Schmitt, 2011), war eine Kompilation der Grammatik zu zeitintensiv und die Größe der resultierenden Maschinen nicht mehr praktikabel. So wurde der Koordinationsblasenansatz final nicht umgesetzt.

Rechtsverzweigende Konjunktkette

Letztlich wurde für die Koordination eine Repräsentationsform herangezogen, bei der die Konjunkte rechtsseitig verkettet werden. Hierbei erhält das erste Konjunkt die Funktion, die der gesamten koordinierten Struktur zugeschrieben wird. Allen anderen wird hingegen das Label `CJ` zugewiesen. Die Konjunktionen sind den Konjunkten linksseitig untergeordnet und erhalten das Label `KON`. Ein Beispiel hierfür ist für den Beispielsatz aus den vorhergehenden Abschnitten „Er liebt weder Frauen und Kinder noch Schweizer“ in Abbildung 5.25 gegeben. Hier besteht innerhalb eines Dependenzknoten-

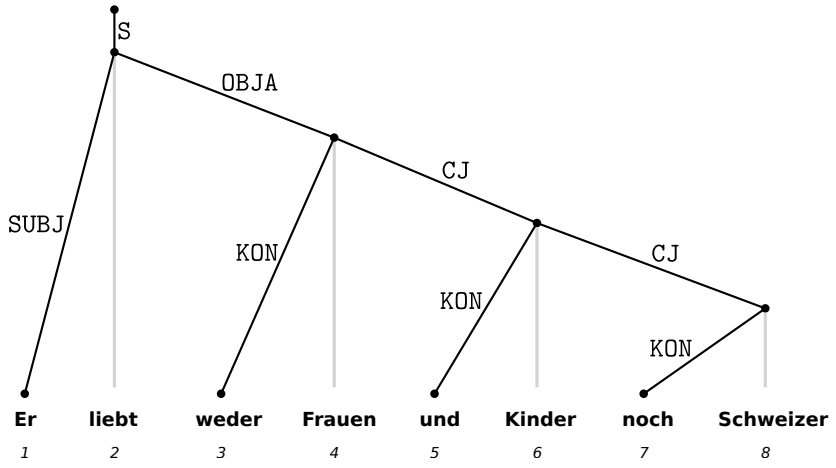


Abbildung 5.25: Koordinationsrepräsentation mit nativem Zugriff auf zwei Konjunkte.

Teilproblems ein nativer Zugriff auf zwei Konjunkte. Wenn Satzzeichen als Trenner verwendet werden, werden diese analog zu den Konjunktionen linksseitig angebunden.

5.6.6 Strukturellipsen

Eine Strukturellipse ist das Resultat der Reduktion von grammatischen Konstruktionselementen (siehe z. B. Zifonun et al., 1997, Abschnitt C4 3.5). Diese kann beispielsweise ein Adjektivattribut oder einen Determiner betreffen. Kritisch wird die Reduktion, wenn sie sich auf ein Verb bezieht. Dies geschieht beispielsweise in Schlagzeilen, die durch Ellipsen so weit reduziert werden, dass sie aus möglichst wenigen Wörtern bestehen. Ein Beispiel hierfür ist die

Strukturellipse „Frau im Tessin ermordet“. Hier wurde neben dem Determiner der Hilfsverbkomplex reduziert. In solch einem Fall wird innerhalb der WCDG-Grammatik für das Deutsche dem Partizip das Subjekt untergeordnet. Eine entsprechende Dependenzstruktur ist in Abbildung 5.26 gegeben.

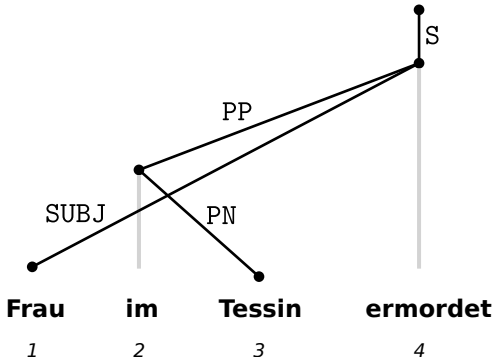


Abbildung 5.26: Hilfsverb-Reduktion.

Wenn hingegen das Vollverb reduziert ist, führt dies innerhalb der WCDG-Grammatik zu Fragmenten. Ein Beispiel hierfür ist die Strukturellipse „Auf dem Rastplatz ein Serienmörder“. Hier fehlt ein Kopulaverb, dem die Präpositionalgruppe und das Subjekt untergeordnet werden können. Dies ist in Abbildung 5.27 verdeutlicht.

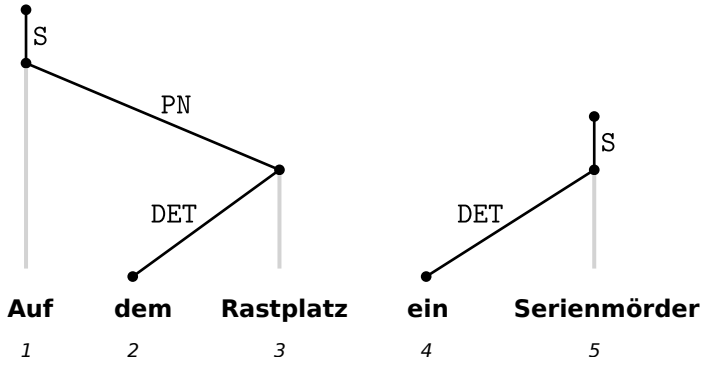


Abbildung 5.27: Vollverb-Reduktion.

In Mel’čuk (1998, Abschnitt 2.8) werden leere Knoten vorgeschlagen, um solche elliptischen Konstruktionen zu modellieren. Des Weiteren wird in Seeker und Kuhn (2012) eine Dependenzversion der Phrasenstrukturbaumbank TiGer vorgestellt, bei der Verbellipsen über leere Governor explizit gemacht

sind. Analog hierzu werden auch in dieser Arbeit leere Knoten herangezogen, die als Platzhalter für reduzierte Vollverben fungieren. Über diese kann für den obigen Beispielsatz eine fragmentlose Dependenzstruktur gebildet werden. Diese ist in Abbildung 5.28 gegeben. Das Subjekt und die Präpo-

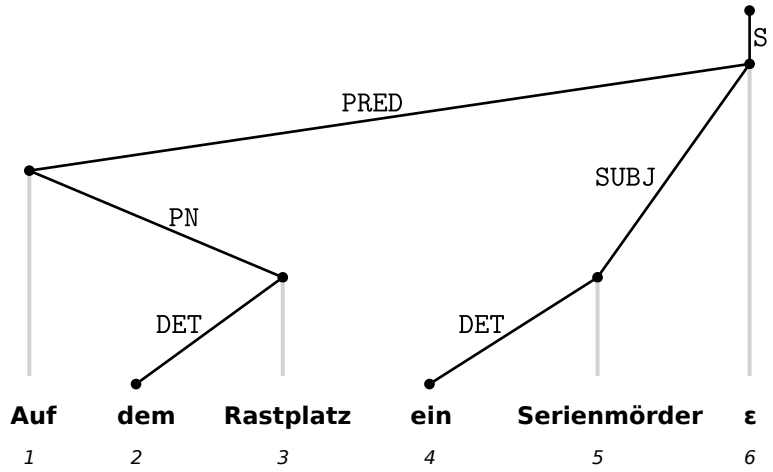


Abbildung 5.28: Vollverb-Reduktion und leerer Governor.

sitionalgruppe werden nun dem Platzhalter untergeordnet, der durch ϵ gekennzeichnet ist und der stets rechtsseitig angenommen wird.

5.6.7 Koordinationsellipsen

Ähnlich wie bei den Strukturellipsen kann es bei der Koordination zur Reduzierung von Verben kommen. Ein Beispiel hierfür ist die Koordinationsellipse „Sie liebt das Gotthardmassiv und er den Dinkelberg“. Für die Strukturzuweisung wird innerhalb der WCDG-Grammatik für das Deutsche das zusätzliche Label `NP2` herangezogen, über das das überzählige Subjekt der Konjunktion untergeordnet wird. Der Dependenzbaum hierfür ist in Abbildung 5.29 gegeben. Problematisch sind hierbei asyndetische Konstruktionen, bei denen für die Unterordnung des überzähligen Subjekts keine Konjunktion vorhanden ist. Für diesen Fall wird innerhalb der WCDG-Grammatik auf eine Lösung zurückgegriffen, die Nichtprojektivität zur Folge hat. Für Koordinationsellipsen, bei denen Vollverb und Subjekt reduziert werden, würde das zusätzliche Label jedoch nicht ausreichen.

In dieser Arbeit wird auf das `NP2`-Label verzichtet und es werden die Ellipsen über leere Knoten behandelt, die als Platzhalter für das reduzierte

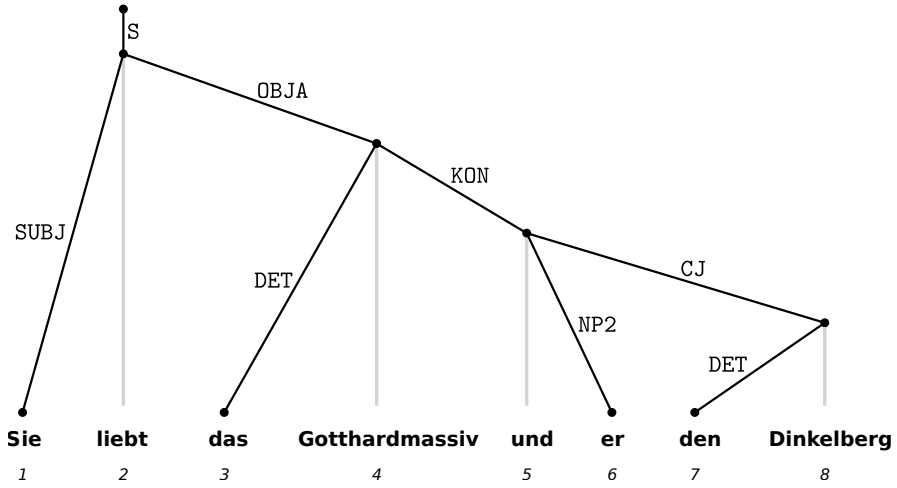


Abbildung 5.29: Koordinationsellipse mit reduziertem Vollverb.

Verb fungieren. Eine entsprechende Struktur für den Beispielsatz „Sie gibt dem Elefanten Bananen und dem Affen Zucker“ ist in Abbildung 5.30 gegeben. In dem Beispiel ist das Vollverb und das Subjekt reduziert und es wird

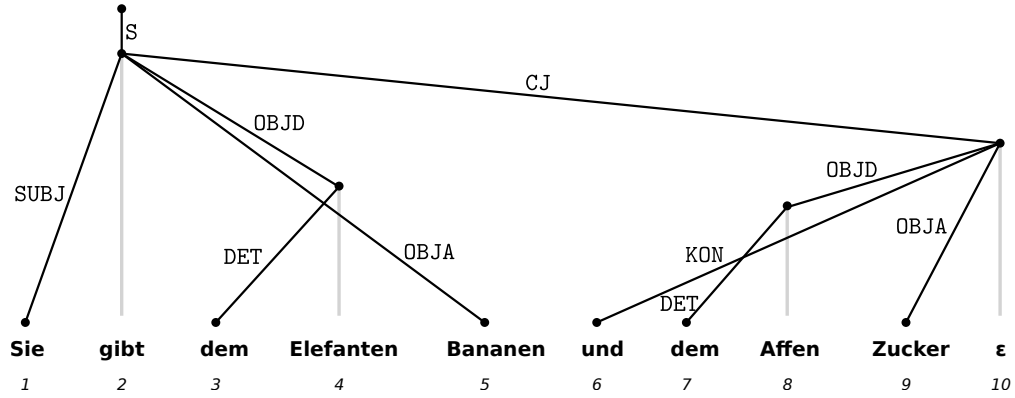


Abbildung 5.30: Repräsentation einer Koordinationsellipse über einen leeren Governor.

ein leerer Knoten herangezogen.

Wenn in der Koordinationsellipse Hilfsverben involviert sind, kann dies theoretisch zu mehreren leeren Knoten führen. Ein Beispiel hierfür ist die Ellipse „Heute sind Frauen befördert und Männer gefeuert worden“. Solche Konstruktionen können, wie in Seeker und Kuhn (2012) vorgeschlagen, über zwei leere Knoten abgebildet werden, einen für das reduzierte infinite Hilfsverb und einen für das reduzierte finite Hilfsverb. In dieser Arbeit wird hingegen

dem exhaustiven Explizitmachen der Ellipsen nicht nachgegangen. Anstelle dessen wird versucht, die leeren Knoten nur dann heranzuziehen, wenn sie für eine verbundene Dependenzstruktur notwendig sind. So wird für den obigen Beispielsatz nur ein leerer Knoten für das reduzierte finite Hilfsverb angenommen. Ein entsprechender Dependenzbaum ist in Abbildung 5.31 gegeben. Es

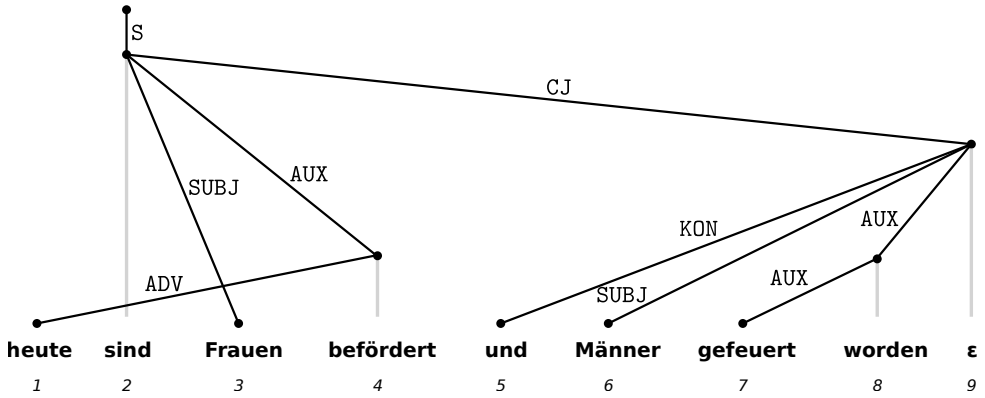


Abbildung 5.31: Keine leeren Knoten für infinite Hilfsverben.

werden in dieser Arbeit allgemein keine leeren Knoten für infinite Hilfsverben angenommen. Daher wird hier das Partizip 'befördert' direkt dem finiten Verb untergeordnet. Zudem stehen die leeren Knoten stets rechtsseitig.

5.7 Performanz der Grammatik

In diesem Abschnitt wird die Performanz der Grammatik untersucht, die im Rahmen dieser Arbeit entwickelten wurde. Dies geschieht anhand von tausend Sätzen der Dependenzversion des NEGRA-Korpus, die über das Werkzeug *DEPSY* (Dependency Synthesizer) erzeugt wurde (Daum et al., 2004) und auf den Annotationsrichtlinien in Foth (2006a) basiert. Für das Testset werden die Sätze von 18602 bis 19601 herangezogen. Diese werden auch in Foth (2006b) für die Evaluation der WCDG-Grammatik für das Deutsche verwendet. Da dort keine datengetriebenen Erweiterungen einbezogen sind, ist ein direkter Vergleich möglich. Im Folgenden wird das System, das in dieser Arbeit entwickelt wurde, als *SCDG* (Semiring-based CDG) bezeichnet.

Die hier entwickelte Grammatik weicht leicht von den WCDG-Annotationsrichtlinien ab (siehe Abschnitt 5.6). Dies betrifft beispielsweise die Behandlung der Koordination, die Hinzunahme von leeren Elementen oder die

stärkere Einbeziehung von Nichtprojektivität. Daher werden die Analyseergebnisse einer nachgeschalteten Transformation unterzogen, die in vielen Fällen eine Informationsreduktion bedeutet, bei der Kanten im Baum höher angehängt werden. Allgemein ist eine Rückführung auf die WCDG-Annotationsrichtlinien immer möglich. Das Transformationsskript wurde auf den ersten tausend Sätzen des NEGRA-Korpus entwickelt und getestet.

Um die Performanz der Grammatik zu evaluieren, wird als Evaluationsmaß der *Labeled Attachment Score* (LAS) und der *Unlabeled Attachment Score* (UAS) gewählt (siehe Kübler et al., 2009, Abschnitt 6.1). Über den UAS wird untersucht, ob das Parsingergebnis die korrekten Köpfe und Dependenzkanten enthält. Beim LAS wird zusätzlich die Akkuratheit der Dependenz-Label berücksichtigt. Des Weiteren wird für die Evaluation angenommen, dass einem Wort von vornherein die korrekte Wortart zugeordnet ist (Gold Part-of-Speech Tags) und dass wie in Foth (2006b) alle Token in die Berechnung mit einbezogen werden, also auch die Satzzeichen. So umfasst das Testset 16.687 Token.[1]

Die hier entwickelte Grammatik soll mindestens die Qualität der WCDG-Grammatik für das Deutsche erreichen. So können in den folgenden Abschnitten die Lösungsstrategien, die in dieser Arbeit entwickelt wurden, unter realistischen Bedingungen untersucht werden. Reine Laborbedingungen mit einer Restriktion auf bestimmte sprachliche Phänomene sollen damit vermieden werden. Zudem soll gezeigt werden, dass der Grammatikformalismus für die Erstellung einer qualitativ hochwertigen, abdeckungsreichen Grammatik geeignet ist. Daher wird im Folgenden erst auf den Vergleich zwischen der Analysequalität der SCDG und der WCDG eingegangen. Danach werden die Dependenz-Label in Bezug zur SCDG im Detail betrachtet.

5.7.1 Vergleich mit der WCDG

In Tabelle 5.5 sind die Evaluationsergebnisse der SCDG im Vergleich zu den der WCDG aus Foth (2006b, Abschnitt 3.6.2) auf Basis des UAS und LAS gegeben. Die Ergebnisse sind für verschiedene Satzlängen aufgeführt. Dies ist gerade deswegen interessant, weil die Satzlänge erfahrungsgemäß einen großen Einfluss auf die Analysequalität hat. Bei dem Vergleich ist anzumer-

[1] In Foth (2006b) werden 16690 Token für den gleichen Abschnitt des Korpus angegeben. Grund hierfür könnten nachträgliche Korrekturen an dem Korpus sein. Dieser marginale Unterschied wird hier vernachlässigt.

ken, dass in Foth (2006b) nicht die tatsächliche Performanz der Grammatik evaluiert wurde, da ein korrektes Ausrechnen der Analysen aus praktischen Gründen nicht möglich war. Es wurde jedoch bestrebt, über einen transformationsbasierten Lösungsansatz und einem ausreichend gesetzten Lösungszeitraum die Suchfehler zu minimieren.

Länge	Sätze	WCDG		SCDG	
		UAS	LAS	UAS	LAS
1-10	340	95,0%	93,4%	96,0%	94,6%
11-20	323	91,7%	90,0%	94,5%	93,0%
21-30	229	90,7%	89,3%	93,3%	92,0%
31-40	76	87,3%	85,8%	92,2%	90,7%
>40	32	84,7%	83,4%	91,6%	90,8%
insgesamt	1000	90.4%	88,8%	93.6%	92.3%

Tabelle 5.5: Vergleich der WCDG-Evaluationsergebnisse aus Foth (2006b) mit denen der SCDG auf Basis des UAS un LAS.

Der Tabelle ist zu entnehmen, dass die WCDG-Grammatik bezüglich des UAS um 3,2 und bezüglich des LAS um 3,5 Prozentpunkte übertroffen werden konnte. Interessant ist hierbei, dass die SCDG nicht so sensibel auf die Satzlänge reagiert. So fällt die Analysequalität bei der WCDG bei beiden Evaluationsmaßen bei Sätzen mit 31-40 Token stark ab. Bei über 40 Token ist dann nochmals ein erheblicher Abfall zu beobachten. Bei der SCDG liegt solch ein Qualitätseinbruch nicht vor. Bei Sätzen mit über 31 Token sind in Bezug zum LAS sogar nur marginale Einbußen zu erkennen. So hat die SCDG insbesondere bei längeren Sätzen eine bessere Performanz. Bei kürzeren mit 1-10 Token ist hingegen nur ein leichter Vorsprung zu beobachten.

Das bessere Abschneiden der hier entwickelten Grammatik ist ein Indiz dafür, dass die Strukturierungsmöglichkeiten wie beispielsweise die dynamischen Merkmale oder die mehrstelligen Hilfsconstraints ein adäquateres formulieren von Beschränkungen ermöglichen. Als weiteren Grund sei das Grammatik-Design genannt, das z. B. über die intensive Einbeziehung nichtprojektiver Strukturen eine präzisere Regelformulierung erlaubt.

Über eine weiterführende Grammatikentwicklung und durch das Tuning auf verschiedenen Textkorpora ließe sich die Analysequalität des bestehenden SCDG-Systems sicherlich verbessern. Jedoch wäre hierbei bald eine Grenze erreicht, bei der die zur Verfügung stehenden Informationen nicht mehr aus-

reichen würden, um zwischen Alternativen korrekt entscheiden zu können. Beispiele hierfür sind im folgenden Abschnitt aufgeführt.

Wie in Foth (2006b) können datengetriebene Methoden einbezogen werden, um die Parsingqualität weiter voranzutreiben. In Khmylko et al. (2009) wird das WCDG-System mit einem MSTParser-Prediktor und fünf anderen Prediktoren wie beispielsweise einem Supertagger kombiniert. Dort wird für dasselbe Testset ein UAS von 93.3 und ein LAS von 92.0 angegeben. In der direkten Gegenüberstellung sind hier die Werte der SCDG etwas besser. Natürlich ist der MSTParser (McDonald, 2006), der dort verwendet wurde, heute nicht mehr State of the Art. Aber der Vergleich zeigt, dass es sich lohnt zu hinterfragen, ob die Möglichkeiten auf regelbasierter Seite bereits ausgeschöpft sind.

5.7.2 Betrachtung der einzelnen Dependenz-Label

In der Tabelle 5.6 sind die Evaluationsergebnisse der SCDG für die einzelnen Dependenz-Label auf Basis des LAS gegeben. Da die Label `NP2` und `VOK` nicht

LAS	Label
-	NP2
-	VOK
100%	
100%	AVZ
100%	KONJ
100%	OBJA2
99,8%	AUX
99,1%	DET
98,8%	PART
98,7%	PN
97,7%	ATTR
96,9%	GMOD

LAS	Label
96,4%	SUBJ
93,4%	CJ
91,7%	ZEIT
90,3%	OBJA
90,0%	S
89,6%	APP
88,5%	KON
88,1%	OBJI
87,5%	GRAD
85,7%	PAR
85,5%	NEB
84,9%	KOM

LAS	Label
84,6%	REL
81,7%	OBJD
79,7%	PRED
77,8%	PP
77,3%	ADV
75,0%	EXPL
71,2%	OBJC
67,5%	SUBJC
47,8%	ETH
37,4%	OBJP
33,3%	OBJG

Tabelle 5.6: Ergebnisse der Grammatikevaluation pro Label.

in den Testdaten vorkamen, konnte für diese kein Wert erhoben werden. Den Ergebnissen ist zu entnehmen, dass einige Label sehr schlecht abschneiden und einen LAS mit unter 80% besitzen. Diese Fälle werden im Folgenden kurz diskutiert.

Für das Label OBJG wird lediglich ein LAS von 33,3% erreicht. Das ist damit zu erklären, dass dieses Label nur sehr selten auftritt und es schwer ist, das Genitivobjekt von einem attributiven Genitiv zu unterscheiden. Auch die Unterscheidung zwischen den Labeln OBJC und SUBJC ist über die Grammatik schwer zu modellieren. Bei den Labeln SUBJ und OBJA gelingt dies hingegen besser.

Bei dem Label OBJP muss berücksichtigt werden, dass im Lexikon Valenzrahmen für Verben mit einem Präpositionalobjekt eingetragen sind. Für diese Verben sind die Präpositionalgruppen in den Korpusdaten jedoch nicht unbedingt als OBJP annotiert, obwohl eine entsprechende Konstruktion vorliegt und andersherum.

Die Label PP und ADV sind generell hochfrequent und schwer über eine Grammatik zu modellieren. Für die Anbindung der Präpositionalgruppen können Valenzinformationen von Verben, Nomen und Adjektiven herangezogen werden. Dies wird für Verben und Nomen auch getan. Jedoch reicht dies bei Weitem nicht aus. Es fehlen für eine adäquate Grammatikmodellierung schlichtweg grammatische Markierungen oder Anhaltspunkte. So ist es in manchen Fällen nicht möglich, anhand der lexikalischen Informationen bei mehreren möglichen Lesarten die richtige zu präferieren. Hier wären z. B. Kookkurrenzinformationen hilfreich.

Bei dem Label ADV tritt zusätzlich das Problem auf, dass in vielen Fällen eine globale Ambiguität in der Anbindung des Adverbs besteht. Bei der manuellen Korpus-Annotation muss sich aber auf eine Lesart festgelegt werden. So können Analysen, die eigentlich korrekt sind, über das annotierte Korpus als falsch eingestuft werden. Das wird dadurch verschärft, dass die Auswahl bei globalen Ambiguitäten bei der Annotation nicht immer nach dem gleichen Prinzip geschieht. Daher sollte die Korrektheit oder Aussagekraft der annotierten Korpora stets kritisch betrachtet werden (siehe hierzu z. B. Volokh, 2013).

Kapitel 6

Experimente

In Kapitel 4.5 wurden verschiedene Strategien vorgestellt, um die Zwischenergebnisse während des Problemlösens handhabbar zu halten. Hier wird über konkrete Experimente untersucht, welchen Effekt diese tatsächlich haben und ob sie einen echten Mehrwert ergeben.

Für die Experimente werden die Grammatik und die Implementierung herangezogen, wie sie in Kapitel 5 beschrieben sind. Des Weiteren wird als Grundlage für das Testkorpus die Dependenzversion des *NEGRA-Korpus* verwendet, die über das Werkzeug *DEPSY* (Dependency Synthesizer) erzeugt wurde (Daum et al., 2004). Es sind Satzlängen in Fünferschritten in das Korpus aufgenommen. Es besteht demnach aus 5, 10, 15 usw. Token langen Sätzen. Da für höhere Satzlängen zu wenig Material für aussagekräftige Ergebnisse zur Verfügung steht, ist eine Satzlänge von 60 als Maximum gewählt. Insgesamt umfasst das Korpus 1.844 Sätze. Für die folgenden Experimente wird angenommen, dass die Wortarten von vornherein bekannt sind (Gold Part-of-Speech Tags).

Es wird untersucht, wie sich die Zustands- und Übergangsanzahl beim Problemlösen verhält. Wenn das Lösen ohne Memoisation betrachtet wird, kann der Algorithmus 4 aus Abschnitt 4.5.6 herangezogen werden. Dann werden bei einer Satzlänge von fünf Zwischenergebnisse mit einer Größe von 1.211.074 Zuständen erreicht. Bei einer Länge von zehn können einzelne Analysen nicht mehr berechnet werden, da die Zustandsanzahl relativ früh auf über 10 Millionen steigt. Der Algorithmus führt demnach schon bei sehr kleinen Satzlängen zu unhandhabbarer FSMs, wodurch keine ausgiebigen Experimente möglich sind. Daher wird im Folgenden ausschließlich das Problem-

lösen mit Memoisation betrachtet.

Eine memoisierte Kaskade besteht aus einer Sequenz von Paaren von FSMs. Wenn $(\widehat{\mathcal{K}}_i, \overline{\mathcal{K}}_i)$ gegeben ist, dann meint die Größe der Zwischenergebnisse zu der Ebene i die Zustandsanzahl des FSMs $\widehat{\mathcal{K}}_i \circ Support(\overline{\mathcal{K}}_i)$, also der Komposition der ersten mit dem Support der zweiten Komponente.

Im Folgenden wird erst die initiale Problemzerlegung und daraufhin der Effekt der verzögerten Anwendung von harten und von weichen Constraints untersucht. Für die Optimalzerlegung wird der Algorithmus 8 aus Abschnitt 4.5.8 herangezogen und für die affixorientierten Zerlegungen der Algorithmus 11 aus Abschnitt 4.5.10. Die abstrakte Lösung für die Ebenen einer memoisierten Kaskade innerhalb einer Optimalzerlegung wird über den Algorithmus 9 aus Abschnitt 4.5.9 und innerhalb einer affixorientierte Zerlegung über den Algorithmus 12 aus Abschnitt 4.5.10 berechnet. Das verzögerte Anwenden von Constraints ist über den Algorithmus 10 aus Abschnitt 4.5.9 realisiert.

6.1 Problemzerlegung

In diesem Abschnitt wird die Problemzerlegung auf Basis von harten Constraints betrachtet. Es wird untersucht, welchen Effekt die einzelnen Lösungsstrategien aus Abschnitt 4.5 haben. Das Augenmerk liegt hierbei auf der Zerlegungstiefe und auf dem Anwachsen der Zwischenergebnisse während der Zerlegung in Bezug zur Satzlänge und zur Ebenenanzahl. Hierbei werden sowohl die Optimalzerlegung als auch die affixorientierten Zerlegungen betrachtet. Für die Letzteren wird eine sukzessive Erweiterung des Affixes um fünf Token angenommen, wobei einer präfixorientierten und einer suffixorientierten Zerlegung nachgegangen wird.

6.1.1 Satzlänge und Zerlegungstiefe

Innerhalb dieses Abschnitts wird für das Testkorpus ermittelt, wie viele Zerlegungsebenen bei den unterschiedlichen Zerlegungsstrategien für die einzelnen Satzlängen durchschnittlich benötigt werden. Die Ergebnisse sind in dem Diagramm in Abbildung 6.1 dargestellt. Dieses ordnet den Satzlängen die durchschnittliche Zerlegungstiefe für die jeweiligen Zerlegungsstrategien zu. Dem Diagramm ist zu entnehmen, dass die Ebenenanzahl bei allen drei Strategien linear mit der Länge des Satzes anwächst. Hierbei sinkt bei der Optimalzerle-

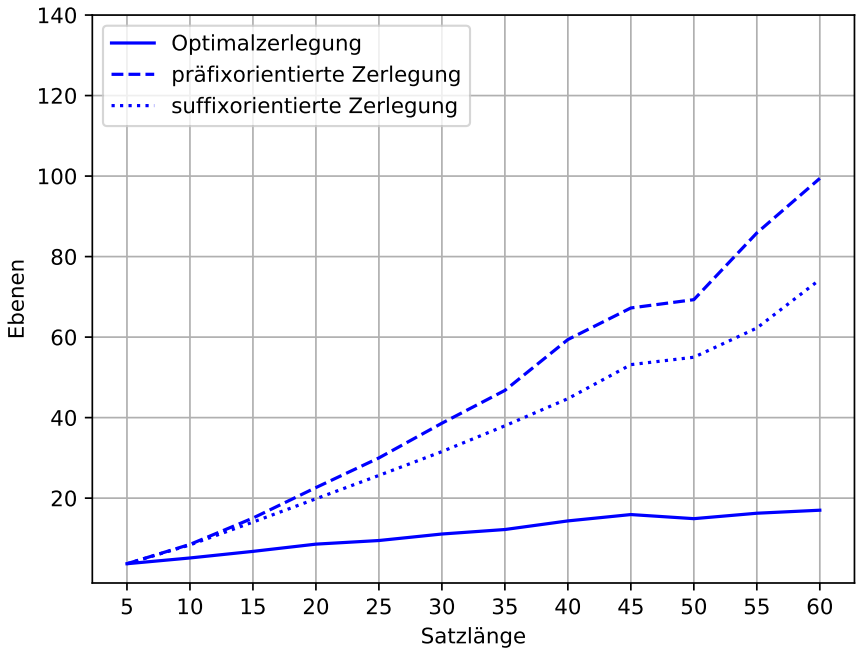


Abbildung 6.1: Zusammenhang zwischen der Satzlänge und der Anzahl an Problemzerlegungsebenen in Bezug zur Zerlegungsstrategie.

gung das Verhältnis zwischen Zerlegungstiefe und Satzlänge von 0,7 auf 0,28. Bei der präfixorientierten Zerlegung steigt dieses hingegen von 0,7 auf 1,66 und bei der suffixorientierten von 0,7 auf 1,24. Wie zu erwarten, führt eine affixorientierte Zerlegungsstrategie zu einer deutlich höheren Ebenenanzahl. Hierbei hat die präfixorientierte mehr Ebenen zur Folge als die suffixorientierte.

6.1.2 Satzlänge und Zustandsanzahl

In diesem Abschnitt wird untersucht, wie sich die Zwischenergebnisse in Bezug zur Satzlänge verhalten. Hierbei wird erst die Berechnung der Problemzerlegung auf Grundlage aller harten Constraints und danach die der abstrakten Lösung zu den einzelnen Zerlegungsebenen betrachtet.

Anhand des Testkorpus wird für jede Zerlegung eines Satzes die höchste Zustandsanzahl ermittelt, die innerhalb einer Ebene auftritt. Hierdurch soll verdeutlicht werden, wie groß die Zwischenergebnisse zu einer Zerlegungsebene werden können. Die Ergebnisse sind für jede Satzlänge gemittelt und in

dem Diagramm in Abbildung 6.2 dargestellt. Dieses zeigt, dass die Zustands-

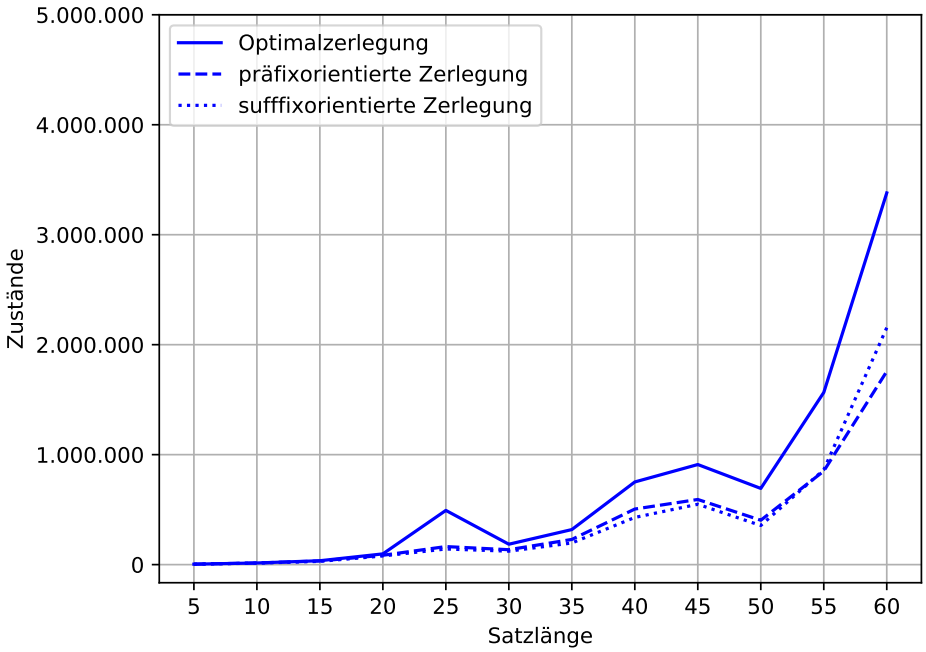


Abbildung 6.2: Zusammenhang zwischen der Satzlänge und der Größe der Zwischenergebnisse nach der Berechnung der Problemzerlegung in Bezug zur Zerlegungsstrategie.

anzahl initial leicht mit der Satzlänge ansteigt. Bei längeren Sätzen, also bei einer Länge von 50 bis 60, ist ein stärkeres Anwachsen der Zwischenergebnisse zu beobachten. Die Zustandsanzahl steigt bei der Optimalzerlegung letztlich auf 3,38, bei der präfixorientierten auf 1,76 und bei der suffixorientierten auf 2,15 Millionen Zustände. Die drei Kurven verlaufen tendenziell recht ähnlich. Die der affixorientierten Zerlegungen liegen hierbei nahe beieinander und zeigen insgesamt ein moderateres Anwachsen als bei der Optimalzerlegung.

Der Vorteil einer affixorientierten Zerlegung wird deutlich, wenn die abstrakte Lösung der einzelnen Ebenen der memoisierten Kaskaden betrachtet wird. Auch hier wird für jede Problemzerlegung eines Satzes die höchste Zustandsanzahl ermittelt, die innerhalb einer Zerlegungsebene auftritt. Die Ergebnisse sind für die einzelnen Satzlängen gemittelt und in dem Diagramm in Abbildung 6.3 dargestellt. Bei der Optimalzerlegung wird bei einer Satzlänge von 55 bereits eine durchschnittliche Zustandsanzahl von 12.580.312 Zuständen erreicht. Für die Länge 60 kann wegen unhandhabbarer Zwischen-

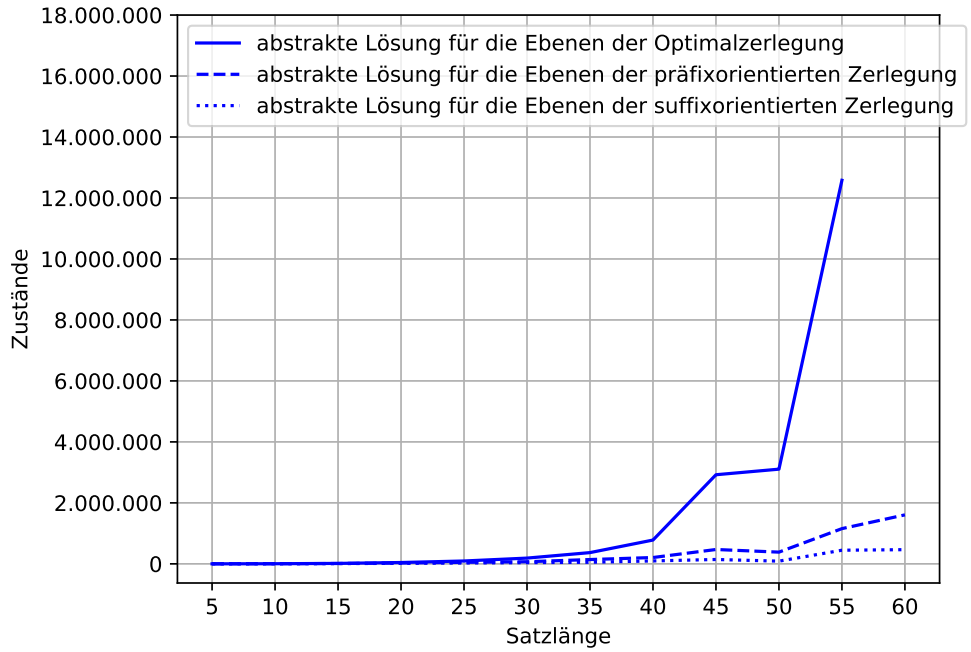


Abbildung 6.3: Zusammenhang zwischen der Satzlänge und der Größe der Zwischenergebnisse nach der Berechnung der abstrakten Lösung für die einzelnen Ebenen der Problemzerlegung in Bezug zur Zerlegungsstrategie.

ergebnisse kein Wert ermittelt werden. Bei den affixorientierten Zerlegungen ist hingegen solch ein extremes Anwachsen der Zustandsanzahl nicht zu beobachten. Die präfixorientierte Zerlegung hat ihren höchsten Wert bei der Satzlänge 60 mit 1.606.650 Zuständen, die suffixorientierte bei der gleichen Länge mit 465.225 Zuständen.

6.1.3 Zerlegungsebenen und Zustandsanzahl

Innerhalb dieses Abschnitts wird untersucht, wie sich die Zwischenergebnisse über die Ebenen hinweg verhalten. Es wird der Vorteil einer kontrollierten Zerlegung verdeutlicht und es werden die Auswirkungen der Berechnung der abstrakten Lösung auf die einzelnen Zerlegungsebenen betrachtet. Hierzu werden ausschließlich Sätze mit einer Satzlänge von 30 Token berücksichtigt, da die Länge nicht zu klein gewählt ist und dabei genügend Daten für die Experimente zur Verfügung stehen. Es wird sowohl die Berechnung der Problemzerlegung auf Grundlage aller harten Constraints als auch die der

abstrakten Lösung zu den einzelnen Zerlegungsebenen untersucht. Für die Experimente werden die Sätze herangezogen, die eine Zerlegungstiefe von 11 besitzen. Dies ist die Durchschnittstiefe für diese Satzlänge im Korpus.

Als Erstes wird die Optimalzerlegung, die abstrakte Lösung für die Ebenen der Optimalzerlegung und zusätzlich die unkontrollierte Zerlegung betrachtet, die den Vorteil einer kontrollierten Zerlegung verdeutlichen soll. Für den letzteren Fall wird eine leicht abgeänderte Version des Algorithmus 8 aus Abschnitt 4.5.8 hinzugezogen, bei der eine simultane lokale Konsistenzregel verwendet wird, die keine konkrete Zerlegungsstrategie unterstützt (siehe hierzu Abschnitt 4.5.1). Damit der Algorithmus terminiert, wird über eine zusätzliche Bedingung eine feste maximale Zerlegungstiefe von 11 gefordert (vgl. Algorithmus 2 aus Abschnitt 4.5.5). Hierdurch wird eine unkontrollierte Zerlegung erreicht. Dies ist analog zu dem Vorgehen in Yli-Jyrä (2012). Die Zustandsgrößen zu den einzelnen Zerlegungsebenen sind für die verschiedenen Strategien gemittelt und in dem Diagramm in Abbildung 6.4 dargestellt.

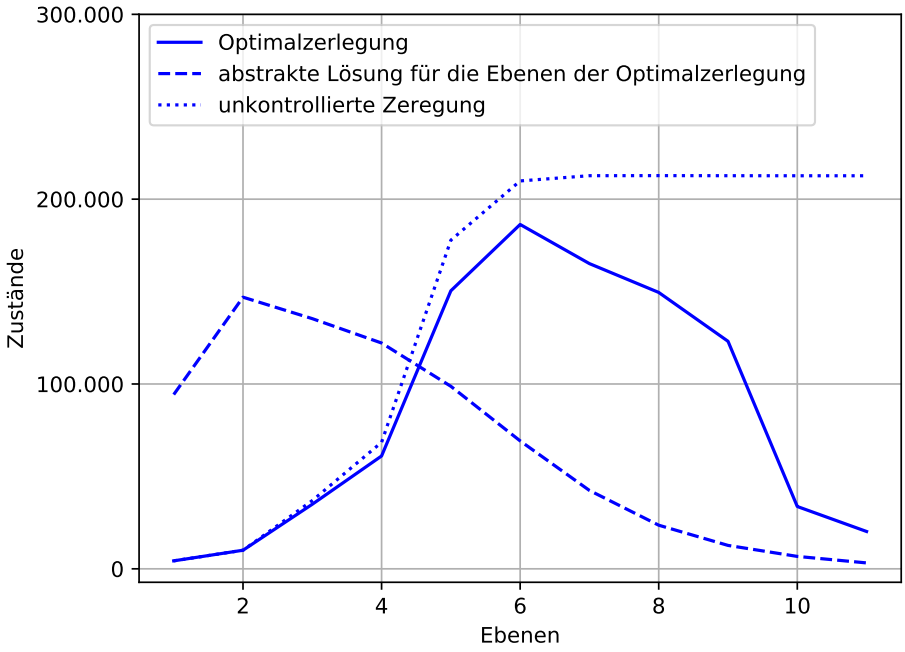


Abbildung 6.4: Zusammenhang zwischen der Anzahl an Problemzerlegungsebenen und der Größe der Zwischenergebnisse nach der Berechnung der Optimalzerlegung, der abstrakten Lösung für die Ebenen der Optimalzerlegung und der unkontrollierten Zerlegung.

Bei der Optimalzerlegung steigt und fällt die Zustandsanzahl in Form einer Glocke, die den höchsten Punkt in der 6. Ebene besitzt. Dort werden 186.308 Zustände erreicht. Die Zustandsanzahl steigt initial, da sowohl Constraints angewendet als auch verschiedene Zerlegungen generiert werden. Das Sinken ist damit zu erklären, dass final weniger Variablen in Teilprobleme involviert werden. So muss über die FSMs weniger Information über Position und Projektion memoisiert werden.

Bei der unkontrollierten Zerlegung erhöht sich die Zustandsanzahl bis zur 6. Ebene auf 209.863 Zustände, wobei der Anstieg etwas höher ist als bei der Optimalzerlegung. Ab der 6. steigt die Anzahl nur noch leicht an und danach stagniert sie von der 10. bis zur 11. Letztlich werden 212.661 Zustände erreicht. Da bei der unkontrollierten Zerlegung in jeder Zerlegungsebene stets auch die Teilprobleme gelöst werden, die bereits in einer vorhergehenden gelöst wurden, sinkt die Zustandsanzahl nach der 6. Ebene nicht mehr. Dass sie nach der 6. aufhört, stark zu steigen, ist damit zu erklären, dass die Information über Position und Projektion ab der 6. über (fast) identische FSMs repräsentiert werden kann.

Die Kurve für die abstrakte Lösung der einzelnen Zerlegungsebenen hat ihren höchsten Punkt in der zweiten Ebene mit 146.961 Zuständen, danach flacht sie langsam auf 3.191 Zustände ab. Es werden sukzessive Wertetupel ausgeschlossen, was in den ersten Zerlegungsebenen ein starkes Anwachsen der Zustandsanzahl mit sich zieht, da dort viele unterschiedliche Variablenbelegungen über die FSMs repräsentiert werden müssen. In den höheren Ebenen sind hingegen weniger Variablen in den Teilproblemen einbezogen, sodass der Wegfall von möglichen Analysen eher zu einem Sinken führt.

Nun wird die präfixorientierte Zerlegung betrachtet. Es sind die Sätze herangezogen, die eine Zerlegungstiefe von 39 besitzen. In diesem Fall ist dies die Durchschnittstiefe für die Satzlänge von 30 im Korpus. Die Ergebnisse sind in dem Diagramm in Abbildung 6.5 dargestellt, wobei die Zustandsgrößen zu den einzelnen Zerlegungsebenen gemittelt sind. Diesem ist zu entnehmen, dass die Zwischenergebnisse wellenartig größer werden und ihren höchsten Punkt in der 37. Ebene mit 103.911 Zuständen besitzen. Das wellenartige Verhalten ist dadurch zu erklären, dass das Präfix sukzessive Stück für Stück erweitert wird. Wenn hingegen die abstrakte Lösung der einzelnen Zerlegungsebenen betrachtet wird, hat die Kurve einen stabileren Verlauf mit zwei Hügeln. Der höchste Punkt ist hier in der 28. Ebene mit 41.254 Zuständen

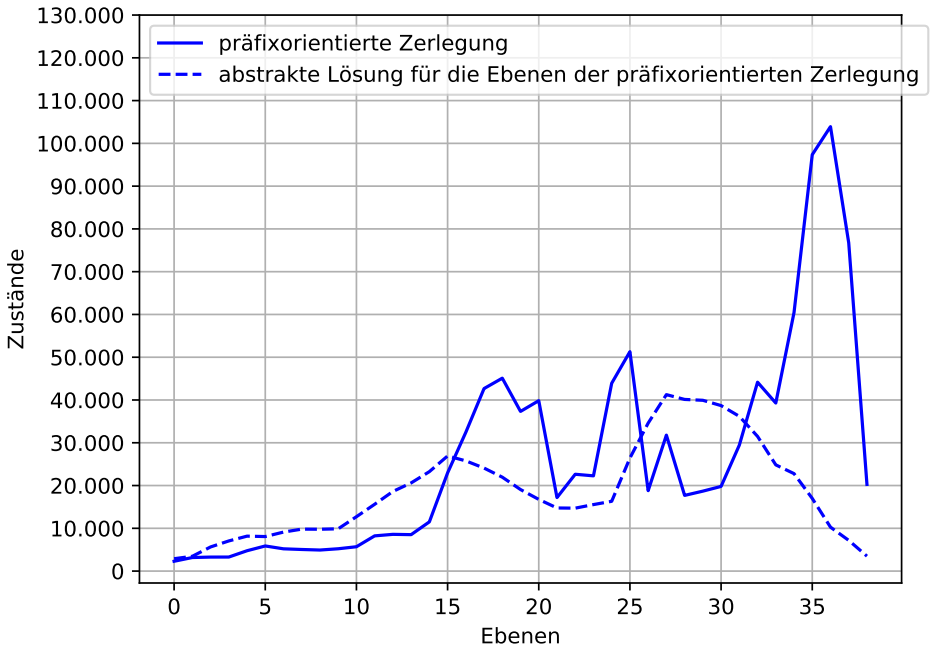


Abbildung 6.5: Zusammenhang zwischen der Anzahl an Problemzerlegungsebenen und der Größe der Zwischenergebnisse nach der Berechnung der präfixorientierten Zerlegung und nach der Berechnung der abstrakten Lösung für die Ebenen der präfixorientierten Zerlegung.

und deutlich niedriger als vor der Berechnung der abstrakten Lösung.

Ergänzend hierzu wird der Fall der suffixorientierten Zerlegung betrachtet. Für diesen sind die Sätze herangezogen, die eine Zerlegungstiefe von 32 besitzen. Dies ist hier die Durchschnittstiefe für die Satzlänge von 30 im Korpus. Die Ergebnisse sind in dem Diagramm in Abbildung 6.6 dargestellt, wobei die Zustandsgrößen zu den einzelnen Zerlegungsebenen gemittelt sind. Es ist ein spitzerer wellenartiger Verlauf zu beobachten als bei der präfixorientierten Zerlegung. Die sechs Spitzen korrespondieren hier mit den sechs Suffixerweiterungen. Der höchste Punkt liegt in der 30. Ebene bei 45.325 Zuständen. Die Kurve für die abstrakte Lösung der Zerlegungsebenen hat einen stabileren Verlauf und ist leicht wellenartig. Der höchste Punkt ist in der 13. Ebene mit 17.333 Zuständen und auch hier deutlich niedriger als vor der Berechnung der abstrakten Lösung.

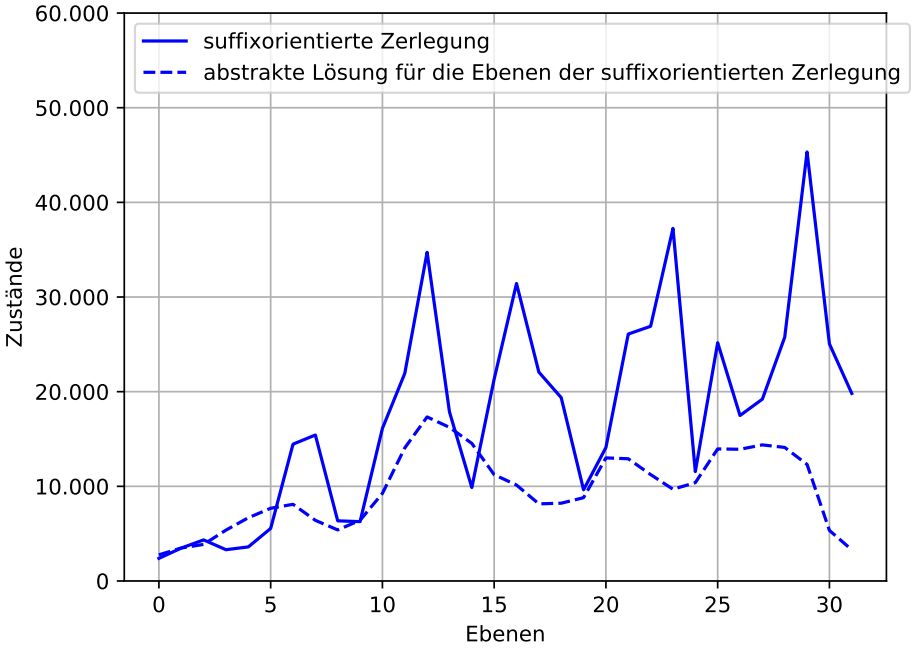


Abbildung 6.6: Zusammenhang zwischen der Anzahl an Problemzerlegungsebenen und der Größe der Zwischenergebnisse nach der Berechnung der suffixorientierten Zerlegung und nach der Berechnung der abstrakten Lösung für die Ebenen der suffixorientierten Zerlegung.

6.2 Verzögerte Anwendung von harten Constraints

Innerhalb dieses Abschnitts wird untersucht, welchen Effekt die verzögerte Anwendung von harten Constraints hat. Es existieren insgesamt 706 harte Constraints in Form von Regelbestandteilen einer simultanen lokalen Konsistenzregel (siehe Abschnitt 5.4). Diese sind hier in drei Mengen aufgeteilt.

- **erste harte Constraint-Menge** mit 567 harten Constraints für zerlegungsrelevante Beschränkungen
- **zweite harte Constraint-Menge** mit 26 Constraints für Is-Beschränkungen
- **dritte harte Constraint-Menge** mit 113 harten Constraints für abtrennbare Präfixe, das Genus Verbi, die Eindeutigkeit von Satzfunktionen und Has-Beschränkungen

Die erste harte Constraint-Menge wird bei der Berechnung der initialen Problem-Zerlegung herangezogen. Die Constraints der zweiten und dritten Menge sind so gewählt, dass sie für die Berechnung der Zerlegung nur eine geringe Relevanz besitzen. Sie werden entsprechend der oben aufgeführten Reihenfolge verzögert angewendet.

Im Folgenden wird ausschließlich die suffixorientierte Zerlegung betrachtet, da sich diese in dem vorherigen Abschnitt bezüglich der Größe der Zwischenergebnisse als am geeignetsten erwiesen hat.

6.2.1 Satzlänge und Zerlegungstiefe

Bei der verzögerten Anwendung von harten Constraints sind nicht alle harten Constraints in die Berechnung der initialen Problemzerlegung einbezogen. Daher kann sich auch die Zerlegungstiefe ändern. Das Diagramm in Abbildung 6.7 ordnet den einzelnen Satzlängen die durchschnittliche Ebenenanzahl zu, die benötigt wird, wenn die Problemzerlegung anhand der ersten Constraint-Menge berechnet wird. Als Vergleich hierzu ist der Fall hinzugezeichnet, bei dem alle harten Constraints in die Berechnung eingebunden sind (rote Linie, vgl. Abbildung 6.1 aus Abschnitt 6.1.1). Dem Diagramm ist zu entnehmen, dass die Zerlegungstiefe leicht höher ist, wenn die Zerlegung anhand der ersten Constraint-Menge berechnet wird. Der größte Unterschied bei den beiden Kurven ist bei den Satzlängen 45 bis 50 zu beobachten. Hier beträgt er bis zu fünf Ebenen.

6.2.2 Satzlänge und Zustandsanzahl

Es wird untersucht, welchen Einfluss die verzögerte Anwendung von harten Constraints auf die Zwischenergebnisse in Bezug zur Satzlänge haben. Für jeden Satz aus dem Testkorpus ist die höchste Zustandsanzahl ermittelt, die sowohl nach der initialen Problemzerlegung als auch nach den verzögerten Constraint-Anwendungen innerhalb einer Ebene auftritt. Die Ergebnisse sind für jede Satzlänge gemittelt.

Als Erstes wird die initiale Problemzerlegung betrachtet. In dem Diagramm in Abbildung 6.8 sind die Zustandsgrößen für die suffixorientierte Zerlegung anhand der ersten harten Constraint-Menge und anhand aller harten Constraints (rote Linie, vgl. Abbildung 6.2 aus Abschnitt 6.1.2) gegenübergestellt. Es zeigt sich, dass die Zustandsanzahl deutlich geringer ist, wenn

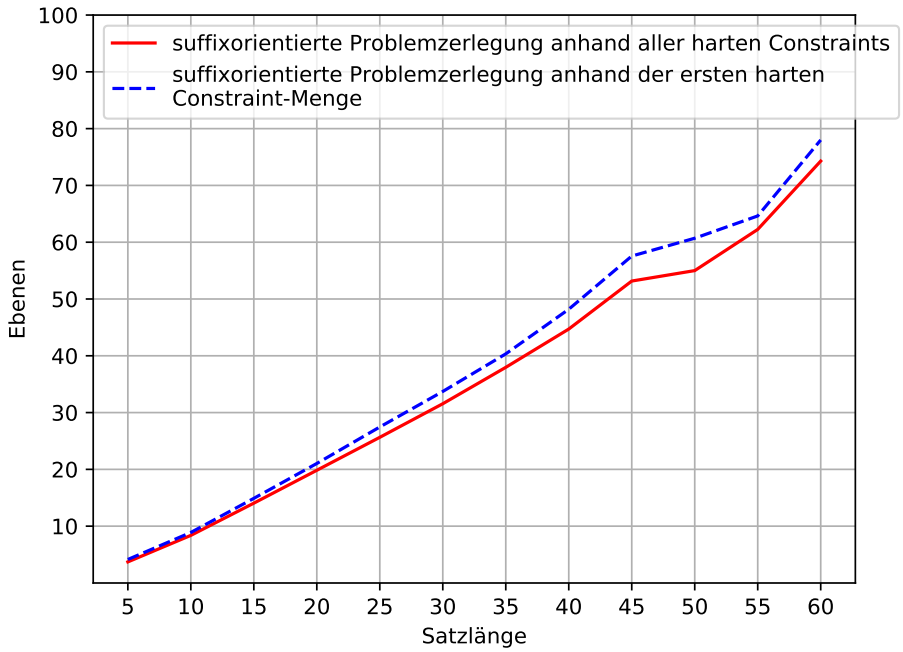


Abbildung 6.7: Zusammenhang zwischen der Satzlänge und der Anzahl an Problemzerlegungsebenen bei einer suffixorientierten Problemzerlegung auf Basis aller harten Constraints und auf Basis der ersten harten Constraint-Menge.

nur die erste harte Constraint-Menge für die Problemzerlegung herangezogen wird. Die Kurve steigt hier auf einen Wert von 106.309 Zuständen. Wenn alle harten Constraints in die Problemzerlegung einbezogen werden, werden hingegen 2.154.484 Zustände erreicht. Die Zustandsanzahl ist dann 20 Mal so groß.

Nun soll die abstrakte Lösung der Zerlegungsebenen nach der verzögerten Anwendung der zweiten und dritten harten Constraint-Menge betrachtet werden. Die Zustandsgrößen sind in dem Diagramm in Abbildung 6.9 gegeben. Für eine bessere Vergleichbarkeit ist zusätzlich die abstrakte Lösung der Ebenen für den Fall berücksichtigt, dass alle harten Constraints in die Berechnung der initialen Problemzerlegung einbezogen sind (rote Linie, vgl. Abbildung 6.3 aus Abschnitt 6.1.2). Dem Diagramm ist zu entnehmen, dass die Kurven sehr ähnlich verlaufen und nahe beieinander liegen. Nur bei den Satzlängen 55 und 60 zeigen sich leichte Unterschiede.

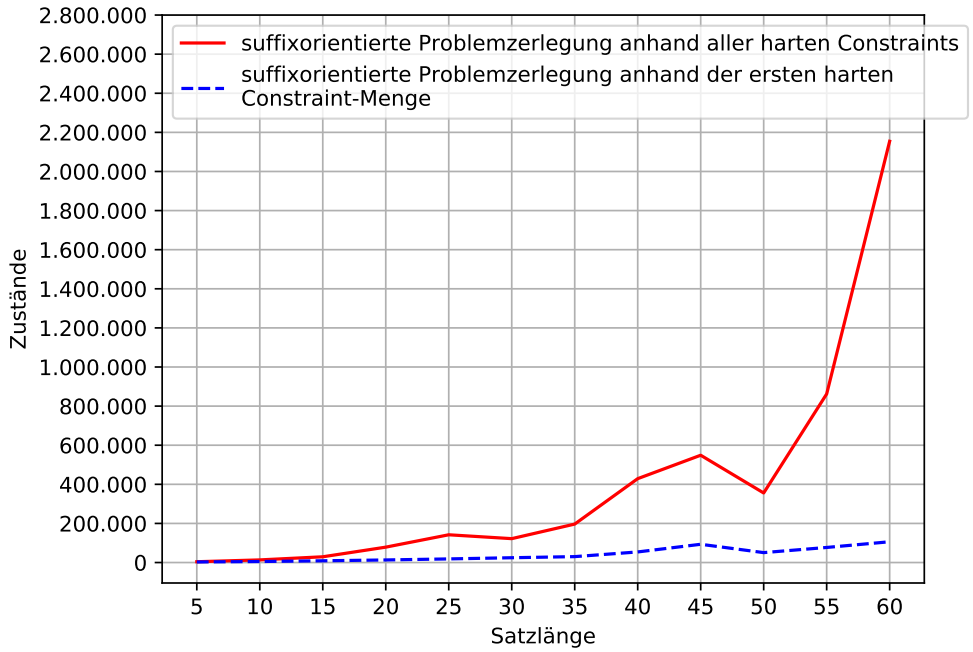


Abbildung 6.8: Zusammenhang zwischen der Satzlänge und der Größe der Zwischenergebnisse bei einer suffixorientierten Problemzerlegung auf Basis aller harten Constraints und auf Basis der ersten harten Constraint-Menge.

6.2.3 Zerlegungsebenen und Zustandsanzahl

Es wird das Anwachsen der Zwischenergebnisse über die Ebenen hinweg im Fall der verzögerten Anwendung von harten Constraints untersucht. Hierzu werden die Sätze für die Experimente herangezogen, die eine Zerlegungstiefe von 34 besitzen. Dies ist die Durchschnittstiefe bei der suffixorientierten Zerlegung mit verzögerter Anwendung von harten Constraints für diese Satzlänge im Korpus. Es wird die abstrakte Lösung der Zerlegungsebenen nach der initialen Zerlegung über die erste harte Constraint-Menge als auch nach der verzögerten Anwendung der zweiten und dritten harten Constraint Menge betrachtet.

Die Zustandsgrößen zu den einzelnen Ebenen sind für die verschiedenen Fälle gemittelt und in dem Diagramm in Abbildung 6.10 aufgetragen. Als Vergleich ist auch die Kurve hinzugezeichnet, die sich auf die abstrakte Lösung der Ebenen der initialen Zerlegung auf Basis aller harten Constraints bezieht (rote Linie, vgl. Abbildung 6.6 aus Abschnitt 6.1.3). Ein Vergleich

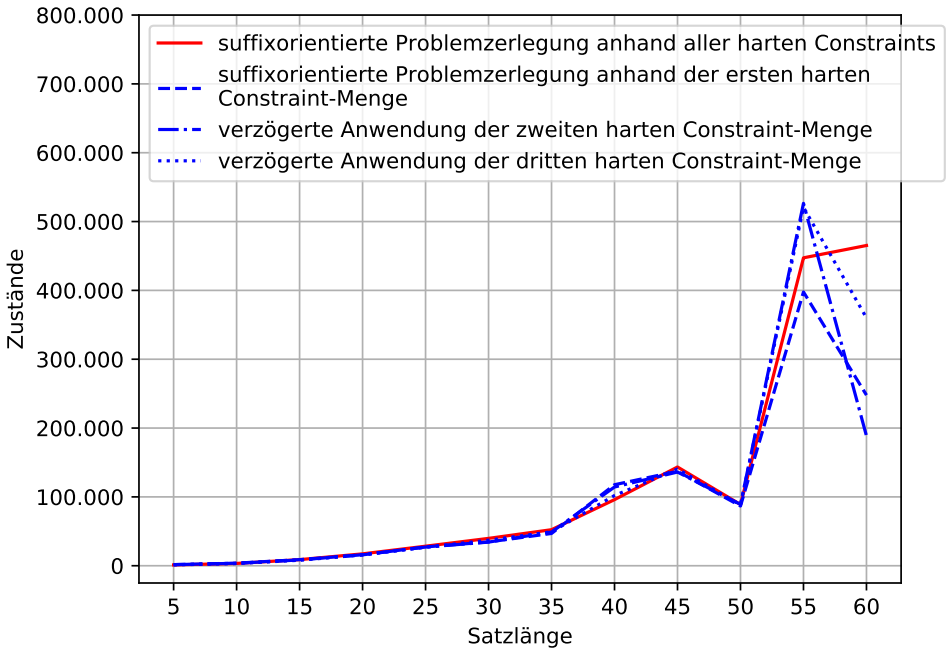


Abbildung 6.9: Zusammenhang zwischen der Satzlänge und der Größe der Zwischenergebnisse bei der abstrakten Lösung der einzelnen Zerlegungsebenen einer suffixorientierten Problemzerlegung im Kontext der verzögerten Anwendung von harten Constraints.

mit der (roten) Kurve aus Abschnitt 6.1.3 ist hier jedoch nur bedingt möglich, da sie auf leicht unterschiedlichen Daten beruht. Die Kurve für die erste Constraint-Menge steigt wellenartig an und hat ihren höchsten Punkt in der 31. Ebene mit 24.965 Zuständen. Danach fällt sie auf 16.062 Zustände ab. Bei der zweiten und dritten Menge steigen die Kurven hingegen wellenartig bis zur 30. Ebene auf 20.675 bzw. 2.2191 Zustände an und fallen dann wieder auf 3.010 bzw. 4.356 Zustände. Wenn alle harten Constraints in die initiale Problemzerlegung einbezogen sind, wird in der Spitze eine etwas niedrigere Zustandsanzahl erreicht. Dass heißt, dass sich die verzögerte Anwendung harter Constraints nur vor der Berechnung der abstrakten Lösungen positiv auf die Größe der Zwischenergebnisse auswirkt.

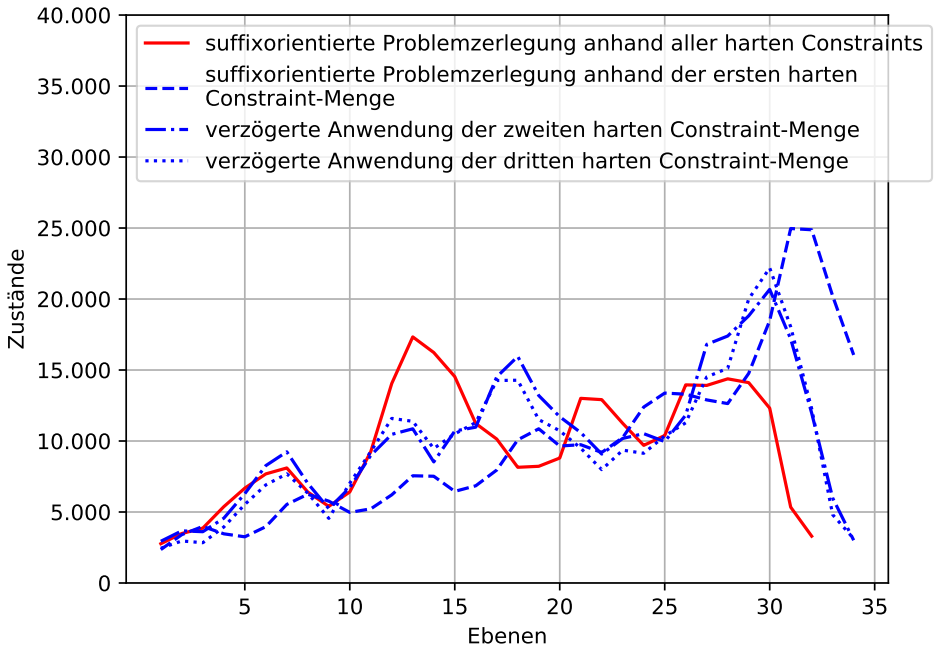


Abbildung 6.10: Zusammenhang zwischen der Anzahl an Problemzerlegungsebenen und der Größe der Zwischenergebnisse bei der Berechnung der abstrakten Lösung der einzelnen Zerlegungsebenen einer suffixorientierten Problemzerlegung im Kontext der verzögerten Anwendung von harten Constraints.

6.3 Verzögerte Anwendung von weichen Constraints

Innerhalb dieses Abschnitts wird die verzögerte Anwendung von weichen Constraints untersucht. Hierbei wird wie in Abschnitt 6.2 ausschließlich die suffixorientierte Zerlegung betrachtet. Des Weiteren wird davon ausgegangen, dass eine Problemzerlegung vorliegt, wie sie in Abschnitt 6.2 beschrieben ist und im Rahmen der verzögerten Anwendung von harten Constraints berechnet wurde.

Es existieren insgesamt 935 weiche Constraints in Form von Regelbestandteilen einer simultanen lokalen Konsistenzregel (siehe Abschnitt 5.4). Diese Constraints sind anhand der Tabelle 5.2 in Abschnitt 5.3.2 in vier Mengen aufgeteilt:

- **erste weiche Constraint-Menge** mit 109 weichen Constraints für

schwere Fehler

- **zweite weiche Constraint-Menge** mit 141 weichen Constraints für minderschwere Fehler
- **dritte weiche Constraint-Menge** mit 167 weichen Constraints für heuristische Regeln
- **vierte weiche Constraint-Menge** mit 518 weichen Constraints für Präferenzen

Diese werden nacheinander entsprechend der hier aufgeführten Reihenfolge angewendet.

6.3.1 Satzlänge und Zustandsanzahl

Hier wird untersucht, wie sich die Zwischenergebnisse bei der verzögerten Anwendung von weichen Constraints in Bezug zur Satzlänge verhalten. Für jeden Satz aus dem Testkorpus ist für die vier weichen Constraint-Mengen jeweils die höchste Zustandsanzahl ermittelt, die innerhalb einer Zerlegungsebene auftritt, nachdem die abstrakte Lösung zu den Zerlegungsebenen berechnet wurde. Die Ergebnisse sind bezüglich jeder Satzlänge für die einzelnen Mengen gemittelt und in dem Diagramm in Abbildung 6.11 dargestellt. Diesem ist zu entnehmen, dass die Zustandsanzahl zu einer Zerlegungsebene von verzögerter Constraint-Anwendung zu verzögerter Constraint-Anwendung sinkt. Von der verzögerten Anwendung der ersten Constraint-Menge zur zweiten Constraint-Menge ist dies am stärksten zu beobachten. Nach der ersten liegt der höchste Punkt bei 147.160 Zuständen, bei der nächsten nur noch bei 18.292, bei der vorletzten bei 6.172 und bei der letzten bei 3.345.

6.3.2 Zerlegungsebenen und Zustandsanzahl

Hier soll das genauere Anwachsen der Zustandsanzahl über die Ebenen hinweg untersucht werden, wenn weiche Constraints verzögert angewendet werden. Wie in Abschnitt 6.2.3 beschrieben, besitzen die zugrundeliegenden Problemzerlegungen eine Zerlegungstiefe von 34. In dem Diagramm in Abbildung 6.12 ist die durchschnittliche Zustandsanzahl nach der abstrakten Lösung der Zerlegungsebenen für die einzelnen verzögerten Constraint-Anwendungen dargestellt. Die Kurven zeigen, dass die Zustandsanzahl zu einer

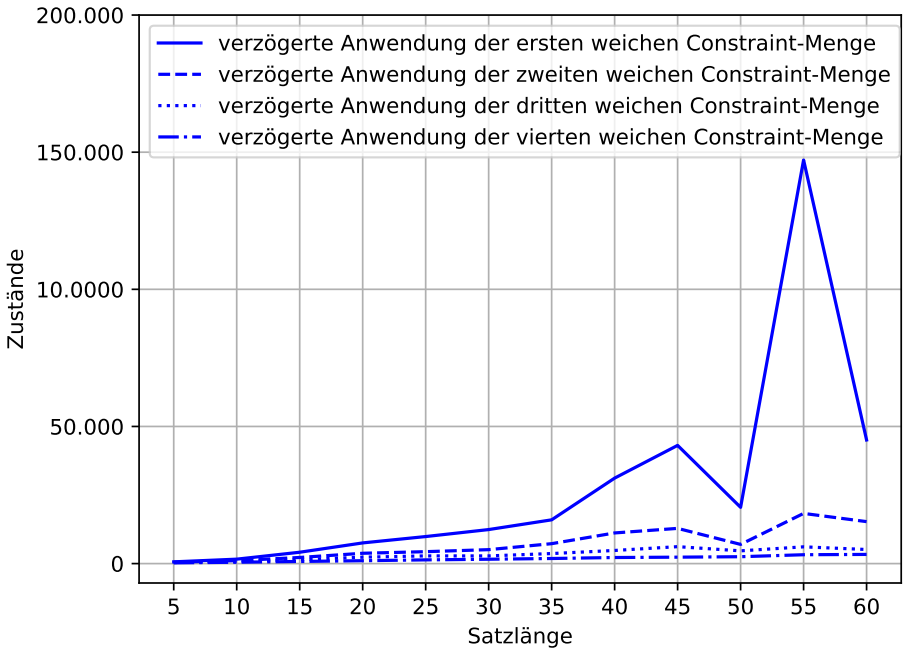


Abbildung 6.11: Zusammenhang zwischen der Satzlänge und der Größe der Zwischenergebnisse nach der Berechnung der abstrakten Lösung für die einzelnen Zerlegungsebenen einer suffixorientierten Zerlegung im Kontext der verzögerten Anwendung von weichen Constraints.

Ebene von verzögerter Constraint-Anwendung zu verzögerter Constraint-Anwendung sinkt. Nach der ersten weichen Constraint-Menge liegt der höchste Punkt durchschnittlich bei 23.532 Zuständen, bei der nachfolgenden nur noch bei 9.883, bei der vorletzten bei 3.538 und bei der letzten nur noch bei 1.705.

6.4 Praktische Laufzeit

Die Experimente zeigen bisher, dass es auf Grundlage einer umfangreichen und abdeckungsreichen Grammatik auch für längere Sätze in der Praxis möglich ist, die abstrakte Lösung zu berechnen. Dies wird über die Problemlösungsstrategien, die in dieser Arbeit erarbeitet wurden, ermöglicht. Im Folgenden wird die praktische Laufzeit in Bezug zur Satzlänge betrachtet.

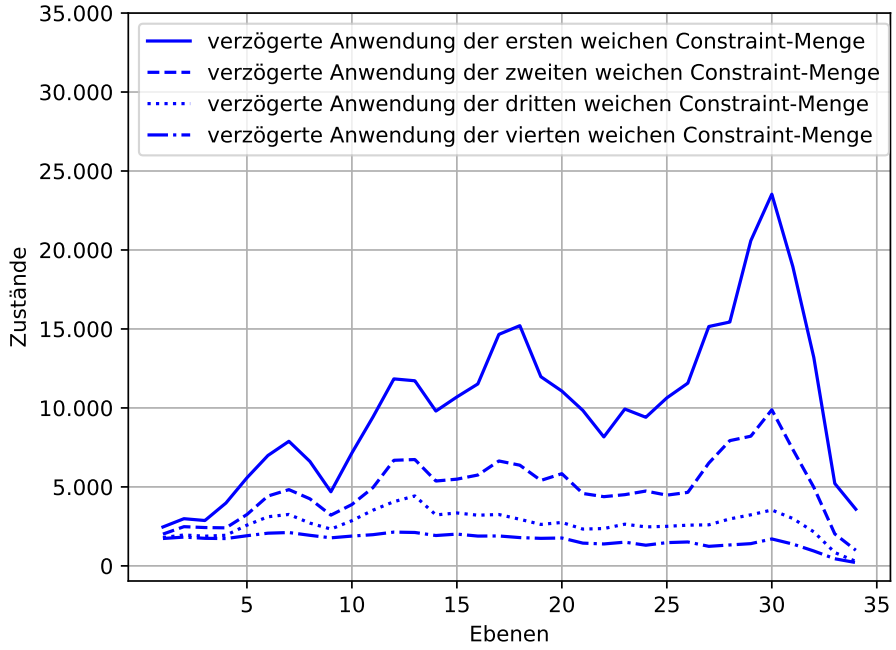


Abbildung 6.12: Zusammenhang zwischen der Anzahl an Zerlegungsebenen und der Zustandsanzahl nach der Berechnung der abstrakten Lösung für die einzelnen Zerlegungsebenen einer suffixorientierten Zerlegung im Kontext der verzögerten Anwendung von weichen Constraints.

6.4.1 Laufzeit und Satzlänge

Die praktische Laufzeit wird für den Fall der verzögerten Anwendung von harten und weichen Constraints im Kontext einer suffixorientierten Zerlegung (siehe Abschnitt 6.2 und 6.3) betrachtet. Die Reihenfolge der Regelbestandteile ist hierbei zufällig gewählt. Als Rechenmaschine lag ein Intel Core i7 Prozessor mit 64-bit und ein Arbeitsspeicher von 16 GB vor. Die Laufzeiten sind für die einzelnen Satzlängen aus dem Testkorpus gemittelt und in Tabelle 6.1 aufgeführt. Dort sind zudem die prozentualen Anteile an der Analysezeit für die drei folgenden Schritte gegeben: Berechnung der Problemzerlegung; verzögerte Anwendung der harten Constraints; verzögerte Anwendung der weichen Constraints. Der Tabelle ist zu entnehmen, dass die Problemlösungszeit grob mit der Satzlänge steigt. Interessant ist hierbei der Anteil der einzelnen Lösungsschritte an der Lösungszeit. So ist zu beobachten, dass die Analysezeit bei den Längen 45 und 55 am höchsten ist und dass dort der größte Zeitbedarf bei der verzögerten Anwendung harter

Satzlänge (Token)	Lösungszeit (Sekunden)	Problem-zerlegung (%)	verzögert hart (%)	verzögert weich (%)
5	4,87	65,09	7,81	27,09
10	11,10	65,89	9,59	24,52
15	23,36	65,64	11,75	22,61
20	41,30	63,54	15,13	21,32
25	65,98	64,48	16,64	18,88
30	97,85	63,39	20,71	15,90
35	156,14	56,83	30,25	12,92
40	390,28	47,87	42,53	9,59
45	1055,65	33,83	61,57	4,59
50	262,81	63,03	23,11	13,86
55	1705,15	25,39	64,03	10,58
60	678,36	57,43	29,76	12,80

Tabelle 6.1: Praktische Laufzeit für die verschiedenen Satzlängen und Anteil der einzelnen Analyseschritte an der Laufzeit.

Constraints liegt. Bei allen anderen Satzlängen liegt der größte Zeitbedarf bei der Berechnung der initialen Problemzerlegung. Bei den Satzlängen 50 und 60 ist ein kurzer Einbruch in der Analysezeit zu verzeichnen. Dies ist analog zu dem Verhalten des Anwachsens der Zustandsanzahl bei der verzögerten Anwendung von harten Constraints (Abbildung 6.9 aus Abschnitt 6.2.2) und lässt sich an dem entsprechenden prozentualen Anteil an der Lösungszeit ablesen. Der Zeitaufwand für die verzögerte Anwendung der weichen Constraints hat hingegen einen moderaten Anteil an der Analysezeit. Für sehr kurze Sätze ist dieser sogar größer als für längere. Insgesamt verhält sich die Problemlösungszeit ähnlich zu dem Anwachsen der Zwischenergebnisse.

6.5 Zusammenfassende Ergebnisse

In Abschnitt 4.5.3 wurde gezeigt, dass beim Parsing über simultane perfekte Relaxierung ohne Zerlegungstiefenbeschränkung der Zeit- und Platzbedarf im schlechtesten Fall exponentiell mit der Eingabelänge wächst. Wenn keine Memoisation einbezogen wird, ist solch ein Verhalten auch in der Praxis zu beobachten. So können wie Anfang des Kapitels angemerkt, keine weiterführenden Experimente für den Algorithmus 4 aus Abschnitt 4.5.6 durchgeführt

werden. Dieser setzt eine perfekte Relaxierung mit Bottom-Up-Lösen, Optimalzerlegung und ungebundener Tiefe um. Ein naiver Parsingansatz führt demnach zu den erwarteten Problemen.

Über die Einbeziehung von Memoisierung wird eine Problemzerlegung im Rahmen der Optimalzerlegung in der Praxis möglich (siehe Abbildung 6.2 aus Abschnitt 6.1.2). Hier ist ein polynomielles Anwachsen der Zustandsanzahl zu erkennen. So führt die Vergessen-Komposition auch mit Einbeziehung von Nichtprojektivität zu kompakten Zwischenergebnissen. In Yli-Jyrä (2012, Abschnitt 5.4) wird eine praktische Auswertung für die Vergessen-Komposition im Fall von projektiven Strukturen durchgeführt. Auch dort wird über Experimente eine polynomielle Zustandskomplexität beobachtet. Zudem wird gezeigt, dass ohne die Vergessen-Komposition ein exponentielles Anwachsen der Zustandsanzahl erreicht wird.

Problematisch ist bei der Optimalzerlegung die Berechnung der abstrakten Lösungen für die einzelnen Zerlegungsebenen (siehe Abbildung 6.3 aus Abschnitt 6.1.2). Änderungen in den Wertezuweisungsmöglichkeiten der Constraint-Variablen, die top-down propagiert werden, führen zu unhandhabbaren Zwischenergebnissen. Über die FSMs muss mehr Information über Projektion, Position und Variablenbelegung für verschiedene Constraint-Systeme repräsentiert werden. Die Kombinatorik führt gerade in unteren Schichten zu unhandhabbaren Zwischenergebnissen, da dort viele Variablen involviert sind (vgl. Abbildung 6.4 aus Abschnitt 6.1.3). So ist eine Analyse für längere Sätze nicht mehr möglich. Das bedeutet, dass die Memoisierung alleine nicht ausreicht, die Zwischenergebnisse klein zu halten.

Die affixorientierten Zerlegungen stellen einen weiteren effektiven Mechanismus dar, die unhandhabbaren Zwischenergebnisse anzugehen. So können hier die FSMs bei der initialen Problemzerlegungsberechnung etwas kompakter gehalten werden (siehe Abbildung 6.2 aus Abschnitt 6.1.2). Bei der Berechnung der abstrakten Lösung für die Ebenen sind die durchschnittlichen Zustandsanzahlen im Vergleich zur initialen Zerlegung sogar geringer (siehe Abbildung 6.3 aus Abschnitt 6.1.2). Über die affixorientierte Strategie wird die Zerlegungstiefe künstlich erhöht (siehe Abbildung 6.1 aus Abschnitt 6.1.1). Hierdurch sind in den einzelnen Zerlegungsebenen weniger Variablen involviert und es können Änderungen in den Wertezuweisungsmöglichkeiten der Constraint-Variablen, die top-down propagiert werden, kompakter dargestellt werden. Die suffixorientierte Zerlegung führt hierbei zu etwas klei-

neren Zwischenergebnissen als bei der präfixorientierten. Dies kann damit zusammenhängen, dass die Dependenzanbindungen mit großer Distanz, die zu Nichtprojektivität führen, stets nach links angebunden werden.

Ein weiteres erfolgreiches Mittel, die Zwischenergebnisse klein zu halten, ist die verzögerte Anwendung von harten Constraints. Über diese kann das Anwachsen der FSMs innerhalb der initialen Problemzerlegung erheblich abgemildert werden (siehe Abbildung 6.9 aus Abschnitt 6.2.2). Hier ist ein lineares Anwachsen der Zustandsanzahl zu erkennen, was im starken Kontrast zu der exponentiellen Komplexitätsabschätzung in Abschnitt 4.5.3 steht. In dieser Betrachtung muss aber berücksichtigt werden, dass gleichzeitig auch ein lineares Ansteigen der Zerlegungsebenen zu beobachten ist und dass die FSMs zu den einzelnen Ebenen gespeichert werden müssen (siehe Abbildung 6.7 in Abschnitt 6.2.1). Zudem werden hier gemittelte Daten für die Kurven herangezogen. Dieser initiale Gewinn hat im weiteren Lösungsverlauf keinen negativen Einfluss auf das Kompakthalten der Zwischenergebnisse (siehe Abbildung 6.10 aus Abschnitt 6.2.3). Das bedeutet, dass es sich lohnt, zu überlegen, welche harten Constraints für eine Zerlegungsberechnung überhaupt relevant sind.

Bisher ist die Einbeziehung von Gewichtungen in das Parsing mit FSMs recht unerforscht. In Yli-Jyrä (2012) werden Gewichte zwar in der Theorie in das Parsing mit Memoisation einbezogen, aber die praktische Auswertung bezieht sich dort ausschließlich auf den ungewichteten Fall. Es wird darauf hingewiesen, dass Gewichte zu einem größeren Anwachsen von Zwischenergebnissen führen können. Hier wurde dahingegen gezeigt, dass über gezieltes Anwenden von weichen Constraints die FSMs auch im gewichteten Fall klein gehalten werden können und dass deren Größe völlig unkritisch ist (siehe Abbildung 6.11 aus Abschnitt 6.3.1). Dies wird insbesondere dadurch ermöglicht, dass weiche Constraint-Mengen auf Grundlage der getrennten Bewertungsbereiche nach und nach angewendet werden. Es werden Änderungen in den Wertezuweisungsmöglichkeiten der Constraint-Variablen nacheinander vollzogen und propagiert. So sind hier die getrennten Bewertungsbereiche einerseits gewünscht, aber andererseits für die effiziente Anwendung der weichen Constraints auch notwendig.

Zusammenfassend können über die präfixorientierte Zerlegung, über die Memoisierung und über die verzögerte Anwendung von harten und weichen Constraints die Zwischenergebnisse klein und handhabbar gehalten werden.

Ein enormes Anwachsen der FSMs ist nicht zu beobachten. Im Gegenteil, die Größen sind gut handhabbar und innerhalb von FSM-basierten Ansätzen nicht ungewöhnlich.

Für die Satzlänge 5 werden ca. 5 Sekunden für die Lösung benötigt. Bei einer Länge von 55 steigt die Lösungszeit auf 28 Minuten (siehe Tabelle 6.1 aus Abschnitt 6.4.1). Dass heißt, dass die Effizienz des Verfahrens an die Satzlänge gekoppelt ist. Der Anstieg scheint nicht ganz quadratisch zu sein und liegt eher darunter und spiegelt grob das beobachtete Anwachsen der Zwischenergebnisse wider. Hierbei muss aber berücksichtigt werden, dass die zugrunde liegenden Daten für die Tabelle gemittelt wurden. Zudem ist anzumerken, dass bei der Grammatik nicht versucht wurde, Einschränkungen vorzunehmen, um die Analyse zu beschleunigen. Im Gegenteil, es wurde vermehrt Nichtprojektivität einbezogen, und es wurden leere Elemente hinzugezogen. Aber letztlich ist hier kein exponentielles Laufzeitverhalten zu beobachten, was sicherlich auch mit der natürlichen Sprache und somit der Grammatik zusammenhängt; diskontinuierliche Strukturen und Zentraleinbettungen sind beispielsweise stark eingeschränkt (siehe z. B. Karlsson, 2010 und Yli-Jyrä, 2003).

6.5.1 Konsequenzen und Möglichkeiten

In diesem Abschnitt wird diskutiert, welche Möglichkeiten eine exakte Lösungsberechnung mit sich bringt und wie diese beschleunigt werden kann. Es wird aber auch auf die lange Laufzeit eingegangen und darauf, dass in zeitkritischen Situationen zusätzlich Pruning-Strategien herangezogen werden können, um schnelles Parsing zu ermöglichen.

Exakte Lösungsberechnung

Die Experimente haben gezeigt, dass die Lösung des Parsingproblems in der Praxis exakt ausgerechnet werden kann. Im Folgenden werden Situationen aufgeführt, in denen diese Möglichkeit einen echten Mehrwert ergibt.

Wenn sich ein Parsing-System in einer produktiven Anwendung befindet, unterliegt es in der Regel einer kontinuierlichen Qualitätskontrolle und einem stetigen Optimierungsprozess. Über die exakte Lösungsberechnung ist es möglich, die Performanz einer Grammatik exakt zu berechnen. Hierdurch können verschiedene Grammatikversionen verglichen und der Effekt

bestimmter geänderter, gelöschter oder hinzugefügter Regeln geprüft werden. So sind exakte Qualitäts- und Regressionstests möglich, die sich ausschließlich auf die Grammatikarbeit beziehen und nicht durch Änderungen im Lösungsalgorithmus beeinflusst werden, wie z. B. den Austausch der unterliegenden Endliche-Maschine-Bibliothek oder die Umsetzung von Laufzeitoptimierungen. Über die völlige Nachvollziehbarkeit der syntaktischen Analysen können zudem Schwachstellen oder konkrete Fehler schnell erkannt, gefunden und ausgebessert werden.

Wenn die Parsingqualität und nicht die Laufzeit im Vordergrund steht, sollte eine exakte Berechnung ohne Approximation bevorzugt werden. Im Folgenden sind Anwendungsszenarien gegeben, die nicht zeitkritisch sind und die Qualität der Analyse im Fokus haben.

Ein solches Szenario ist die automatische syntaktische Annotation von Korpusdaten. Bei dieser werden Textdaten mit linguistischen Informationen angereichert. Diese können Lexikographen bei der Recherche von Stichwörtern unterstützen (siehe z. B. Grefenstette, 1998). Der Annotationsvorgang wird in der Regel weit im Voraus geplant und ist daher nicht zeitkritisch. Analog hierzu verhält sich die Extraktion von syntaktischen Kookkurrenzen für die Erstellung von Kookkurrenzstatistiken, die ebenso die Arbeit der Lexikographen unterstützen kann (siehe Geyken et al., 2009). Bei beiden Szenarien sollte qualitativ alles aus dem Parsing-System herausgeholt werden, was möglich ist. Dies kann über die exakte Lösungsberechnung gewährleistet werden. Wenn die Grammatik es zulässt, können hierbei syntaktische Ambiguitäten unaufgelöst bleiben. Diese können dazu genutzt werden, die Qualität der Kookkurrenzstatistiken zu verbessern. Wenn nach dem Parsing syntaktische Ambiguitäten bestehen, können sowohl mehrere Lesarten berücksichtigt als auch gezielt konkurrierende Lesarten ausgeschlossen werden. Hierdurch ist es möglich, zusätzliche Kookkurrenzen einzubeziehen, aber auch falsche zu vermeiden.

Ein weiteres Anwendungsszenario ist die Grammatikfehlerdiagnose im schulischen Kontext. Hier ist es im Fall von Lernstandserhebungen nicht nötig, sofort ein Feedback zu geben, da die Auswertung erst später dem Lehrenden zur Verfügung stehen muss. Es können grammatische Fehler erkannt werden, weil es eine vorgegebene Norm gibt, die besagt, was richtig ist. In diesem Szenarium stellt die Grammatik die Norm zusammen mit verletzbaren Regularien dar. Wenn eine optimale Analyse gefunden werden kann

und ein wichtiger weicher Constraint verletzt ist, bedeutet dies, dass der Eingabesatz fehlerhaft ist. So können fehlerbehaftete Stellen kenntlich und erklärbar gemacht werden (vgl. Menzel und Schröder, 1999). Hierbei sollte es möglich sein, genau zu bestimmen, ob Regularien verletzt sind. Über eine exakte Lösungsberechnung kann dies gewährleistet werden. Des Weiteren können syntaktische Ambiguitäten, falls die Grammatik diese zulässt, innerhalb einer nachgeschalteten semantischen Komponente genutzt werden, um domänenspezifisches Wissen in einen weiteren Disambiguierungsprozess einzubeziehen (vgl. Köhn und Menzel, 2015).

Innerhalb des WCDG-Ansatzes ist eine vollständige Suche und somit das exakte Ausrechnen der besten Analyse für größere Probleme in der Praxis heikel. So lässt sich die tatsächliche Performanz einer abdeckungsreichen Grammatik in der Praxis nicht ermitteln. Für das Parsing hat sich ein transformationsbasierter Ansatz etabliert, der das Grammatikmodell approximiert (Foth et al., 2000b). Hier kann es vorkommen, dass ein Analysefehler nicht aus der Grammatik, sondern aus dem heuristischen Lösungsansatz resultiert. Bei Qualitäts- und Regressionstests muss daher eine algorithmische Komponente mitgedacht werden, auch wenn ausschließlich die Grammatikentwicklung im Fokus ist. Des Weiteren muss mit nicht völlig nachvollziehbaren Resultaten umgegangen werden. Zudem werden über den transformationsbasierten Lösungsansatz keine syntaktischen Ambiguitäten vorgehalten, die beispielsweise in einer nachgeschalteten semantischen Komponente verarbeiten werden können. Dafür können aber innerhalb der WCDG Syntax und Semantik über verschiedene Rollen einbezogen und verknüpft werden (siehe Foth et al., 2005).

Laufzeitverbesserungen

Hier sollen praktische Laufzeitverbesserungen für den Fall betrachten werden, bei dem die Lösung weiterhin exakt ausgerechnet wird.

Innerhalb des Ansatzes werden FSMs als Regelbestandteile wiederholt angewendet. Würden diese offline zusammengefasst werden, wäre eine deutlich schnellere Lösungsberechnung möglich. So müssten beispielsweise im Fall der initialen Problemzerlegung nicht mehr 567 harte Constraints nacheinander ausgewertet werden. Das Zusammenfassen ist wegen der Zustandskomplexität aber heikel. Jedoch würde bereits ein selektives Zusammenlegen eine erhebliche Laufzeitverbesserung bedeuten. Hier müsste geprüft werden, wel-

che Regelbestandteile sich dafür eignen. Zudem müsste zusätzlich die Abfolgesequenz der entsprechenden FSMs optimiert werden, um eine schnellere Berechnung zu ermöglichen (vgl. Koskenniemi et al., 1992, Abschnitt 4.2 oder Tapanainen, 1997, Abschnitt 10.3.3). Zudem könnte die Anwendung der 935 weichen Constraints beschleunigt werden, indem diese im Rahmen der einzelnen Constraint-Mengen parallel angewendet werden. Gewichtungen müssten dann unter Zuhilfenahme eines angepassten c-Semirings mithilfe der Äquivalenztransformationen über FSMs verrechnet werden.

Insbesondere die verzögerte Anwendung harter Constraints scheint zu längeren Lösungszeiten zu führen. Das grundlegende Problem liegt darin, dass Kompositionsoperationen und Äquivalenztransformationen wiederholt angewendet werden müssen, die bei größeren Maschinen sehr aufwendig sein können (siehe Abschnitt 2.2.1). Hier wäre es denkbar, eine Propagierungstechnik ähnlich zu der des Regular-Membership-Constraints (Pesant, 2004) einzubeziehen, um Inkonsistenzen auszuschließen und initial Zustandsgrößen zu reduzieren.

Die derzeitige Koordinationsumsetzung innerhalb der Grammatik wirkt sich stark auf die Größe der Regelautomaten aus. Um die Automaten kleiner zu halten, könnte die Koordination alternativ über Sub-Dependenzknoten-Teilprobleme umgesetzt werden. Hierdurch würde die Koordinationslogik ausgelagert werden und es gäbe mehr Spielraum, FSMs offline zusammenzufassen. Zudem könnten auf diese Weise die Zwischenergebnisse während des Problemlösens kleiner gehalten werden, da die Anzahl der Zerlegungsebenen erhöht und dadurch eine kompaktere Repräsentation ermöglicht wird.

Die Behandlung von virtuellen Knoten könnte optimiert werden, indem die entsprechenden Variablen erst während des Parsings dynamisch eingefügt werden. Dies dürfte sich positiv auf die Größe der Zwischenergebnisse auswirken, da die FSMs der initialen Zerlegungsebenen kleiner gehalten werden könnten. Es könnte aber auch innerhalb der Dependenzknoten-Teilprobleme nativ mehr Kontext einbezogen werden, sodass die Verwendung der virtuellen Knoten obsolet wird. So könnte beispielsweise der Beschränkungsbereich um adjazente Variablen zur Rechten und Linken erweitert werden. Dann müssten jedoch Teilproblemüberlappungen beachtet werden, was die Kompilierung erheblich verkomplizieren würde.

Des Weiteren kann im Rahmen der FSMs von neuen Forschungsergebnissen Gebrauch gemacht werden, die sich auf das effiziente Berechnen von

rationalen Operationen oder Äquivalenztransformationen beziehen. So wird beispielsweise für die Kompositionsoperation in Argueta und Chiang (2018) eine Implementierung für Graphics Processing Units und in Elghadyry et al. (2021) eine Parallelisierung in einem Parallel Programming Model vorgestellt. Diese Arbeiten zeigen beispielsweise, wie die praktische Laufzeit der Komposition erheblich verbessert werden kann.

Pruning

Über die Experimente wurde gezeigt, dass ein Parsing-Ergebnis in der Praxis vollständig ausgerechnet werden kann. Jedoch ist die Parsing-Zeit sehr lang und stark an die Satzlänge gekoppelt. Bei Aufgaben, die eine zügige Rückmeldung benötigen, ist die Analysegeschwindigkeit aber ein kritischer Faktor.

Der transformationsbasierte Lösungsansatz der WCDG ist aus praktischer Sicht deutlich schneller als eine vollständige Suche (siehe Foth et al., 2000b). Jedoch ist die Laufzeit im schlechtesten Fall immer noch exponentiell. Der Ansatz besitzt aber die Anytime-Eigenschaft. So kann jederzeit die Analyse gestoppt und ein komplettes Ergebnis zurückgegeben werden. Über den hier vorgestellten Ansatz ist dies nur bedingt möglich. Es könnte die Berechnung der initialen Problemzerlegung frühzeitig gestoppt werden. Danach müsste aber noch die memoisierte Kaskade abgearbeitet werden. Zudem würde bei der affixorientierten Analyse lediglich ein analysiertes Affix zurückgegeben werden. Bei der Optimalzerlegung könnten hingegen partielle Ergebnisse für den gesamten Satz ermittelt werden. Diese Zerlegungsform hat sich jedoch als berechnungskritischer erwiesen.

Alternativ könnten Pruning-Strategien einbezogen werden, die es ermöglichen, den Suchraum zu beschneiden. Die Idee dahinter ist, die Größe der FSMs zu den Zerlegungsebenen zu begrenzen und somit eine komplexitätsreduzierende Approximation umzusetzen. Je nach Zeitdruck kann der Lösungsraum stärker oder schwächer beschnitten werden. Der hybride Parser ParZu (Sennrich et al., 2009) zeigt beispielsweise, dass auch bei aggressivem Pruning eine gute Parsingqualität erreicht werden kann und dass auf diese Weise schnelles Parsing ermöglicht wird (siehe z. B. Ortmann et al., 2019). Pruning-Strategien können sowohl für die Optimalzerlegung als auch für die affixorientierten Zerlegungen herangezogen werden und Bezug auf die Schichtteilprobleme und die Dependenzknoten-Teilprobleme nehmen. Die Möglichkeit, im

Parsingergebnis syntaktische Ambiguitäten vorzuhalten, wird durch das Beschneiden des Lösungsraums nicht ausgeschlossen. Des Weiteren ist es möglich, den Einfluss des Prunings auf die Parsingqualität auszuwerten, da die Parsing-Ergebnisse für einen Vergleich exakt ausgerechnet werden können.

Es existieren verschiedene Vorgehensweisen, das Beschneiden des Suchraums zu implementieren (im Kontext der Stot Grammar vgl. McCord et al., 2012, Abschnitt „SG syntactic analysis“). Es kann regelbasiertes Pruning anhand von Constraints umgesetzt werden, die z. B. über reguläre prädikatenlogische Formeln erstellt werden. Bei diesem Vorgehen würden die Parsingergebnisse halbwegs nachvollziehbar bleiben, insbesondere, wenn das Beschneiden des Suchraums nachverfolgt werden würde. Eine andere Möglichkeit, das Parsing zu beschleunigen, ist die Einbeziehung von datengetriebenem Pruning. Bei diesem würden die Analysen ausgeschlossen werden, die aus datengetriebener Sicht nicht erfolgversprechend sind. Eine interessante Variante ist hierbei die Verwendung von Constraints, die aus den eigenen Parsingergebnissen und somit aus den memoisierten Kaskaden abgeleitet werden. Bei dieser Herangehensweise wäre es möglich, sowohl Hilfsconstraint-Informationen als auch dynamische Merkmale einzubeziehen.

Kapitel 7

Schlussbetrachtung

Der Bereich der Constraint-Verarbeitung und der Bereich der endlichen Techniken haben ein großes Potenzial, sich gegenseitig mit neuen Perspektiven und Möglichkeiten zu bereichern und deren Kombination schafft einen echten Mehrwert.

Ausgehend von dieser These wurde in den letzten sechs Kapiteln den einzelnen Zielen dieser Arbeit nachgegangen. Dies betraf einerseits die Repräsentation, Lösung und Erweiterung des SCSPs mithilfe von FSMs und andererseits das robuste Dependenzparsing als konkreten Anwendungsfall. Hierbei bestand der Anspruch, kein kleines Problem unter Laborbedingungen, sondern ein komplexes, realitätsnahes Problem zu betrachten.

Im Folgenden wird eine kurze Zusammenfassung der Arbeit gegeben, um das erfolgreiche Bestreiten der Arbeitsziele kompakt wiederzugeben. Danach wird zusammenfassend auf die gegenseitige Bereicherung des Bereichs der Constraint-Verarbeitung und des Bereichs der endlichen Techniken eingegangen. Daraufhin wird eine wissenschaftliche Einordnung der Arbeit in den Kontext der aktuellen Forschung vorgenommen. Hierüber wird deutlich, welchen Mehrwert die Kombination aus Constraint-Verarbeitung und endlichen Techniken mit sich bringt. Abschließend wird ein Ausblick für zukünftige Forschungen gegeben, die sich aus der vorliegenden Arbeit ergeben können.

7.1 Zusammenfassung

Initial wurde ein Ansatz vorgestellt, der es ermöglicht, das SCSP (Bistarelli et al., 1995) und somit auch indirekt das VCSP (Schiex et al., 1995) auf natürliche Weise mithilfe von FSMs zu repräsentieren und zu lösen. Hierzu wurde ein Vorgehen erarbeitet, über das ein beliebiges SCSP in die Repräsentationsform der FSMs überführt werden kann und es wurde gezeigt, wie die Constraint-Operationen über rationale Operationen umgesetzt werden können. So können, eine Endliche-Maschine-Bibliothek und einen Compiler für reguläre Ausdrücke zur Hand, neben dem klassischen beispielsweise auch das gewichtete und probabilistische CSP und das Fuzzy-CSP implementiert und Constraint-Propagierungstechniken einbezogen werden.

Hierauf aufbauend wurde das SCSP um drei wesentliche Aspekte erweitert: um die Möglichkeit, einen unendlichen Wertebereich einzubeziehen; um eine Repräsentationsform, über die Probleme simultan gelöst werden können; um Constraints, die für jeden Skopus, aber auch für verschiedene Constraint-Systeme wohldefiniert sind. Bei dem Ansatz wurde davon Gebrauch gemacht, dass FSMs unendliche Sprachen erkennen können und eine kompakte Repräsentation von Mengen von Zeichenketten erlauben.

Ausgehend hiervon wurde das Dependenzparsing im Stil der WCDG (Foth et al., 2004) mithilfe von FSMs angegangen. Es wurde auf die Problematiken einer naiven Herangehensweise in der Umsetzung des WCDG-Ansatzes eingegangen und es wurden Zwischenergebnisse von unhandhabbarer Größe und fehlende Lokalität in der Grammatik als grundlegendes Problem erkannt. Um diesen Schwierigkeiten zu begegnen, wurde für das Parsingproblem eine geeignete Problemzerlegung und ein zugeschnittener Beschränkungsbereich erarbeitet.

Darauf aufbauend wurde ein Problemzerlegungsansatz zur perfekten Relaxierung vorgestellt, der es erlaubt, anhand einer Grammatik alle optimalen Lösungen zu berechnen. Es wurde gezeigt, dass auch lokale Konsistenzregeln über FSMs umgesetzt werden können. Über diese wurden verschiedene Zerlegungsstrategien und Anwendungsstrategien von harten und weichen Constraints erarbeitet. Der Ansatz wurde mit einer Memoisationstechnik kombiniert, um Zwischenergebnisse während des Problemlösens klein zu halten. Zudem wurde gezeigt, wie überlokale Constraints über Hilfsconstraints und dynamische Merkmale umgesetzt und sowohl Verletzungen als auch Präfe-

renzen nachverfolgt werden können.

Für die praktische Evaluierung des neuen Ansatzes wurde sowohl ein Parsing-System implementiert als auch ein Grammatikformalismus entwickelt, der auf regulären prädikatenlogischen Formeln (Hulden, 2008) beruht. Hierauf aufbauend wurde eine umfangreiche Grammatik für das Deutsche entwickelt, die im Vergleich zur WCDG im größeren Maße nichtprojektive Strukturen einbezieht, mehrstellige überlokale Constraints und dynamische Merkmale erlaubt, leere Köpfe unterstützt, um sprachliche Ellipsen zu behandeln, und neben Allquantifizierungen auch existenzielle Quantifizierungen ermöglicht. Sie wurde auf einem Teil der Dependenzversion des NEGRA-Korpus (Daum et al., 2004) evaluiert und erreicht auf dem Testkorpus eine bessere Parsingqualität als das WCDG-System (Foth, 2006a). Hierdurch wurde gezeigt, dass über die Ausdrucksmöglichkeiten innerhalb der Grammatik eine präzise und strukturierte Formulierung von Regeln möglich ist.

Abschließend wurde über Experimente gezeigt, dass über die perfekte Relaxierung und die Memoisation von Zwischenergebnissen in Kombination mit einer suffixorientierten Problemzerlegung und der verzögerten Anwendung von harten und weichen Constraints die FSMs während des Problemlösens handhabbar gehalten werden können. So erlaubt der vorgestellte Ansatz die Berechnung aller nach der Grammatik optimalen Analysen auch in der Praxis. Jedoch wurde ebenfalls gezeigt, dass die Laufzeit für zeitkritische Anwendung nicht ausreicht. Abschließend wurde diskutiert, welche Möglichkeiten die exakte Lösungsberechnung eröffnet und welche Konsequenzen die lange Lösungszeit mit sich bringt.

7.2 Gegenseitige Bereicherung

Im Folgenden wird darauf eingegangen, inwieweit der Bereich Constraint-Verarbeitung im Rahmen dieser Arbeit von dem Bereich der der endlichen Techniken profitiert und andersherum.

7.2.1 Constraint-Verarbeitung

Das MUSE-CSP (Helzerman und Harper, 1996) erlaubt es, ähnliche CSP-Instanzen kompakt zu repräsentieren. Dies sind Probleme, die schwer oder umständlich über ein einzelnes CSP dargestellt werden können. Über FSMs ist

es möglich, SCSP-Instanzen und somit ein weiteres Spektrum solcher Probleme simultan und kompakt zu repräsentieren. Dies eröffnet in der Constraint-Verarbeitung neue Möglichkeiten, realitätsnahe Probleme abzubilden. Innerhalb der WCDG werden verschiedene lexikalische Lesarten eines Satzes über ein einzelnes COP abgebildet (siehe Schröder, 2002, Abschnitt 2.4). Über ein simultanes SCSP können die lexikalischen Lesarten hingegen über unterschiedliche simultan repräsentierte Probleme umgesetzt werden, die im Vergleich weniger komplex sind.

Über FSMs können Variablen, Wertetupel und Constraints gemeinsam innerhalb einer Repräsentationsform behandelt werden. Dies macht es möglich, unvollständige CSPs über FSMs zu kompilieren und später durch Hinzufügen von Constraints konkrete Instanzen zu bilden (siehe Hadzic et al., 2008). Über die Erweiterung auf das SCSP ist das Kompilieren über FSMs auch für COPs umsetzbar. Hierbei fällt die Kompilierung mit der FSM-Repräsentationsform des SCSPs zusammen: Sie ist nichts anderes als ein Kombinieren von Constraints. In dieser Arbeit wurde gezeigt, dass das SCSP mithilfe von zyklischen FSMs um einen unendlichen Wertebereich erweitert werden kann. Zyklische FSMs können aber auch dafür verwendet werden, eine unendliche Anzahl an SCSP-Instanzen simultan zu repräsentieren und globale Constraints umzusetzen, die für jeden Skopus wohldefiniert sind. Hierdurch können partielle Lösungen noch unabhängiger von konkreten Instanzen vorberechnet werden. Über die Umsetzung und Erweiterung des SCSPs mithilfe von endlichen Maschinen entsteht somit eine Perspektive, in der die kompakte Repräsentation mehrerer Probleminstanzen, das Kompilieren von Problemen und die Verwendung globaler Constraints gesamtheitlich gesehen werden können. Hierdurch entstehen neue Möglichkeiten, Probleme abzubilden und partielle Lösungen für eine bessere Performanz beim Online-Problemlösen vorzuberechnen.

Innerhalb der CDG werden bei der Analyse initial Knoten und Kantenkonsistenz hergestellt (siehe Maruyama, 1990a). Dieses Vorgehen ist für weiche Constraints jedoch nicht geeignet. Innerhalb der WCDG wurden daher initial heuristische Suche und Pruning-Techniken einbezogen (Menzel und Schröder, 1998). Durchgesetzt hat sich jedoch ein transformationsbasierter Lösungsansatz (siehe Foth et al., 2000b). Im Rahmen des SCSPs ist nun mithilfe der FSMs ein Ansatz zur perfekten Relaxierung (siehe Ugo Montanari, 1991) möglich. In Oflazer (2003) wird Bottom-Up-Parsing mit der

Anwendung von Constraints kombiniert. In der vorliegenden Arbeit wurde gezeigt, dass sich dieses Vorgehen im Rahmen des SCSPs auch auf die perfekte Relaxierung übertragen lässt. So kann einer Lösungsform nachgegangen werden, bei der die Berechnung einer Problemzerlegung und das Anwenden von Constraints gleichzeitig geschehen. Hierbei kann der Suchraum kompakt dargestellt werden, da die Einbeziehung einer Memoisierte-Kaskade eine kompakte Repräsentation von Zwischenergebnissen ermöglicht (siehe Yli-Jyrä, 2012). So kann aufseiten der Constraint-Verarbeitung eine Perspektive eingenommen werden, bei der Zerlegung und Constraint-Anwendung nicht völlig getrennt voneinander sind und bei der auch bei komplexeren Problemen ein Ergebnis exakt ausgerechnet werden kann. Zudem werden über den Zerlegungsansatz Pruning-Strategien interessant, um die Analyse zu beschleunigen, da sie sich gut mit der perfekten Relaxierung kombinieren lassen.

In einem binären COP müssen bis zu $O(n^2)$ Wertepaare geprüft werden, in einem ternären COP bereits bis zu $O(n^3)$ und so weiter. Aus diesem Grund sind innerhalb der WCDG nur binäre Constraints erlaubt und daher nativ keine existenziell quantifizierten Beschränkungen möglich (siehe Schröder, 2002, Abschnitt 2.6). Innerhalb der Umsetzung des SCSPs mithilfe von FSMs ist die Constraint-Stelligkeit hingegen kein relevanter Faktor. Die quantifizierten regulären Constraints unterliegen beispielsweise keiner Stelligkeitseinschränkung und ermöglichen die Realisierung von existenziell quantifizierten Beschränkungen. Aus diesem Grund sind auch mehrstellige Hilfsconstraints möglich. Über die kompakte Repräsentationsform der FSMs können zudem dynamische Merkmale umgesetzt werden, die mit einem Erweitern des Wertebereichs korrespondieren. Über die FSMs stehen im Rahmen des Parsings als COP somit neue Möglichkeiten zur Verfügung, Beschränkungen zu formulieren, den Sichtbereich einer Regel zu erweitern und eine Grammatik zu strukturieren und somit präzise Regeln umzusetzen.

7.2.2 Endliche Techniken

In dieser Arbeit wurde gezeigt, dass das SCSP (siehe Bistarelli et al., 1995) mithilfe von FSMs umgesetzt werden kann. Hierdurch steht aufseiten der endlichen Techniken ein einheitlicher Formalismus zur Verfügung, Situationen zu beschreiben und Lösungen zu berechnen. Dies verhält sich ähnlich zu den regulären prädikatenlogischen Formeln, die eine einheitliche Notation

ermöglichen, um komplexe FSMs zu kompilieren. Klassische CSP-Probleme wie das N-Queens-Problem sind in der Lehre bereits Teil der Einführung in den praktischen Umgang mit FSMs. Über das SCSP existiert nun ein Zugang zu den gewichteten Maschinen, der gerade wegen seiner Eleganz und Einfachheit sehr geeignet ist.

Im Rahmen des SCSPs werden realitätsnahe Situationen mithilfe der c-Semiring-Struktur (Bistarelli et al., 1995) abgebildet. Diese, aber auch verwandte positive oder bipolare Präferenzstrukturen (siehe Bistarelli et al., 2005) bieten neue Möglichkeiten, FSM-Lösungen zu implementieren, die unabhängig von konkreten Gewichtsstrukturinstanziierungen sind, aber bestimmte Eigenschaften fordern. Es kann zudem auf Arbeiten zur c-Semiring-Abstraktion, also dem Mapping von einem c-Semiring in einen anderen c-Semiring, zurückgegriffen werden, um Approximationen umzusetzen oder um sich bei einer Berechnung auf ein optimales Ergebnis zu beschränken (siehe Bistarelli et al., 2000).

Über den WCDG-Ansatz wird robustes Dependenz-Parsing durch graduell verletzbare Regeln umgesetzt (siehe Heinecke et al., 1998). Beim Parsing mit FSMs wird Robustheit hingegen durch partielle Analysen, durch das Auslassen von Regeln und durch das Belassen von strukturellen Ambiguitäten ermöglicht (siehe z. B. Koskenniemi, 1990 und Oflazer, 2003). In dieser Arbeit wurde gezeigt, dass auch im Rahmen der endlichen Techniken eine robuste Analyse über graduell verletzbare Regeln möglich ist. Aufseiten der endlichen Techniken kann demnach stark von den Forschungsergebnissen und Herangehensweisen der WCDG in dem Bereich robuster Systeme profitiert werden.

Die Zustandskomplexität ist ein fundamentales Problem bei der Analyse durch FSMs. Ein Beispiel hierfür ist die FSIG (siehe Tapanainen, 1997). Das SCSP eröffnet neue Möglichkeiten, dieses Problem anzugehen. So könnte der FSIG-Ansatz über ein simultanes SCSP umgesetzt werden, um Constraint-Propagierungstechniken wie die des Regular-Membership-Constraints (Pesant, 2004) einzubeziehen. Inwieweit dies zu einem erfolgreichen Ansatz führt, ist unklar. Dies soll aber zeigen, dass das SCSP insbesondere über die Propagierungstechniken neue Perspektiven eröffnet, unhandhabbaren Zwischenergebnissen zu begegnen.

Innerhalb der WCDG lassen sich Regeln über logische Formeln unabhängig von der darunterliegenden Repräsentationsform formulieren. Bei FSM-

Ansätzen muss bei der Regelformulierung über reguläre Ausdrücke hingegen eine bestimmte Zeichenkettencodierung der Eingabe mitgedacht werden. Über reguläre prädikatenlogische Formeln ist zwar eine gewünschte Abstraktion auch im Rahmen der FSMs möglich (siehe Hulden, 2008); dies wurde durch den in dieser Arbeit entwickelten Parsing-Ansatz verdeutlicht. Jedoch haben sich diese Formeln bisher noch nicht in der FSM-Community durchsetzen können. Der WCDG-Ansatz zeigt aber, welchen immensen Vorteil deren Verwendung mit sich bringt. Zudem kann aufseiten der endlichen Techniken stark von den Forschungsergebnissen im Bereich der Beschränkungsformulierung profitiert werden.

7.3 Einordnung

Schon seit längerer Zeit wird im Bereich des Parsings mit FSMs nicht mehr geforscht. Die letzten Arbeiten liegen bereits lange zurück. Zwar zeigt Yli-Jyrä (2012) über seinen Memoisierungsansatz neue Perspektiven auf, aber zu stark ist die Erfahrung, dass Parsing mit FSMs in der Praxis nicht praktikabel ist und nicht mit anderen Ansätzen mithalten kann. WCDG war Anfang der 2000er Jahre eines der besten Parsing-Systeme für das Deutsche (siehe Versley, 2005) und konnte sich später im Rahmen eines hybriden Systems behaupten (siehe Khmylko et al., 2009). Jedoch scheinen rein datengetriebene Methoden nun eine höhere Parsingqualität bei einer weit besseren Parsinggeschwindigkeit zu erreichen (vgl. Adelmann et al., 2018). Zudem fehlen dem WCDG-System die Vorteile eines klassischen regelbasierten Parsers: Es wird kein exaktes Ergebnis ermittelt und es wird keine Ambiguitätsinformation in Form von verschiedenen syntaktischen Lesarten vorgehalten.

Hinzu kommt, dass künstliche neuronale Netze und Deep Learning bisherige Methoden in der Sprachverarbeitung verdrängen. Dies betrifft auch den Bereich des Dependenzparsings (für einen Überblick siehe Otter et al., 2021). Über die neuronalen Technologien ist es beispielsweise möglich, die Vorteile des übergangsbasierten (transition-based) und des graphbasierten (graph-based) Dependenzparsings zusammenzubringen (siehe Kulmizev et al., 2019). Die symbolischen Verfahren und insbesondere die regelbasierten Methoden verlieren in der Sprachverarbeitung hierdurch sukzessive an Bedeutung. Dies trifft auch die endlichen Techniken und die klassische Constraint-Verarbeitung. Zudem kommt im Kontext der neuronalen End-to-End-Systeme die Frage

auf, ob die syntaktische Analyse überhaupt einen Mehrwert besitzt (im Kontext von Language Understanding siehe Glavaš und Vulić, 2021).

Im starken Kontrast hierzu wächst ein immer prominenter werdender Trend in der KI-Forschung, bei dem versucht wird, subsymbolische und symbolische Methoden zusammenzubringen (für einen Überblick siehe z. B. d'Avila Garcez und Lamb, 2020). Das Ziel der neurosymbolischen Verfahren ist es, die Schwächen der symbolischen und der subsymbolischen Ansätze zu beseitigen und gleichzeitig ihre Stärken zu bewahren. Zentrale Motivationen für diesen Trend ist unter anderem die Interpretierbarkeit und die Out-of-Distribution-Generalisierung. Systementscheidungen sollen für einen menschlichen Benutzer transparent und erklärbarer sein. Zudem sollte ein System in der Lage sein, Szenarien zu bewältigen, die sich deutlich von den Trainingsdaten unterscheiden. Es existieren aber auch andere Motivationspunkte wie das Lernen von kleinen Datenmengen und ein geringerer Ressourcenverbrauch.

Diese neurosymbolische Strömung ist auch im Bereich der endlichen Techniken deutlich spürbar. In Rastogi et al. (2016) wird im Rahmen eines hybriden Ansatzes die Neuronalisierung von FSTs im Kontext von Zeichenkettentransduktionen vorgeschlagen. Es werden die Kanten eines FSTs kontextabhängig mithilfe der neuronalen Features eines neuronalen Modells gewichtet. In Lin et al. (2019) wird eine Erweiterung dieses Ansatzes vorgeschlagen, bei dem es möglich ist, die Pfade anstatt der Kanten über ein neuronales Modell abhängig von beliebigen Pfadmerkmalen zu gewichten. Dann hängt das Kantengewicht von dem Kontext ab, in dem sich die Kante befindet. Hierdurch löst sich der Ansatz von der Markov-Eigenschaft. Neuronalisierte FSMs sind dadurch motiviert, dass sie interpretierbar sind und es erlauben, Domänenwissen einzubeziehen. Eine anderer Neuronalisierungsansatz wird in Stahlberg et al. (2019) im Rahmen eines Rankingansatzes im Kontext der automatischen Grammatikprüfung vorgestellt. Dort wird über FSMs ein gewichteter Hypothesenraum erstellt, der anschließend mithilfe von neuronalen Modellen bewertet und umgewichtet wird. Hierbei werden die Hypothesen dazu genutzt, die neuronalen Modelle einzuschränken. Ähnlich hierzu werden in Zhang et al. (2019) FSTs herangezogen, um im Kontext der Textnormalisierung neuronale Modelle restringieren und somit Normalisierungen auszuschließen, die angesichts der Eingabe völlig irreführend sind.

Im Rahmen der natürlichen Sprachverarbeitung haben FSMs ein großes neurosymbolisches Potential. Über die Neuronalisierung können neuronale

Modelle sowohl mit regelbasierten Modellen, die z. B. über reguläre Ausdrücke oder reguläre prädikatenlogische Formeln erstellt wurden, als auch mit klassischen statistischen Modellen wie z. B. N-Gram-Modellen (siehe Nasr und Volanschi, 2005), Entscheidungsbäumen (siehe Sproat und Riley, 1996) oder lineare Modellen (siehe Wu et al., 2014) verknüpft werden. Auf diese Weise ist es möglich, modulare Systeme zu entwerfen, die symbolische und subsymbolische Aspekte zusammenbringen. Diese Modularität steht im Kontrast zu dem derzeitigen Trend zu monolithischen neuronalen Netzen und ermöglicht eine bessere Interpretierbarkeit und Erklärbarkeit. Es können aber auch Lernverfahren für FSMs einbezogen werden, die es erlauben, eine FSM-Kaskade auf End-to-End-Daten zu trainieren oder zu optimieren (siehe Chiang et al., 2010). Hierüber können die Vorteile eines monolithischen Systems eingebracht werden. Da in dieser Arbeit gezeigt wurde, dass das SCSP mithilfe von FSMs umgesetzt werden kann, kann das SCSP sowohl von den Möglichkeiten der neuralisierten FSMs als auch von neuen Forschungsergebnissen in diesem Bereich profitieren. So sind beispielsweise neuronalisierte Constraints als eine spezielle Art der weichen Constraints naheliegend.

In Kilgarriff et al. (2014) wird verdeutlicht, dass das Parsing natürlicher Sprache vor großen Herausforderungen steht. Es wird im Rahmen der *Sketch Engine* (`https://www.sketchengine.eu`, Kilgarriff et al., 2004) das Szenario der automatischen Erstellung von *Word Sketches* betrachtet. Das sind einseitige Zusammenfassungen des kollokationellen Verhaltens eines Wortes in Form von grammatischen Verbindungen. Diese werden auf Grundlage enormer Textmengen erstellt und beruhen auf der automatischen Extraktion von syntaktischen Kookkurrenzen. Letztere können mithilfe einer sogenannten *Sketch Grammar* extrahiert werden, die auf Extraktionsregeln basiert, die über reguläre Ausdrücke definiert sind. Eine andere Möglichkeit der Extraktion ist das syntaktische Parsing. Die Extraktionsqualität kann hierbei indirekt über die Qualität der Word Sketches abgeschätzt werden. Für das Tschechische wurde auf Basis eines Korpus mit 368 Millionen Wörtern gezeigt, dass mithilfe von statistischen und regelbasierten Parsern keine bessere Qualität der Word Sketches erreicht werden konnte als bei der Verwendung der Sketch Grammar, die im Vergleich ein sehr einfaches Werkzeug darstellt.

Das *DWDS-Wortprofil* ist eine Plattform des *Digitalen Wörterbuchs der deutschen Sprache* (`http://www.dwds.de`, Geyken et al., 2017), die analog zu den Word Sketches sogenannte *Wortprofile* für das Deutsche liefert und

unter anderem im Sprachlernen eingesetzt wird (siehe Geyken et al., 2009). Den Lernenden werden sowohl die Wortprofile als auch die Texttreffer zu den einzelnen grammatischen Verbindungen angeboten. Es besteht demnach ein besonderer Anspruch an die Texttrefferqualität. Aus diesem Grund wurde bei der Erstellung von Wortprofil ein regelbasierter partieller Parser (SynCoP, Didakowski, 2008a) eingesetzt, der auf einer abdeckungsreichen Morphologie (TAGH, Geyken und Hanneforth, 2005) basiert und für die Kookkurrenzextraktion optimiert wurde und eher auf Präzision als auf Ausbeute setzt. Da die zugrunde liegenden Textmengen viele Fragmente und ungrammatische Sätze enthalten und eine hohe Textsortenstreuung besitzen, ist hier die Verwendung vollständiger Parser für die Extraktion der syntaktischen Kookkurrenzen heikel (vgl. Didakowski und Geyken, 2013). Dies wird über die Ergebnisse der Label-spezifischen Auswertungen von verschiedenen Dependenzparsern im Kontext von Out-of-Domain-Texten und der Einbeziehung von automatischem Wortarten-Tagging in Adelmann et al. (2018) verdeutlicht. Das heißt, dass die Anwendbarkeit des vollständigen Dependenzparsings stark vom jeweiligen Anwendungsfall abhängt.

Die vorliegende Arbeit entstand parallel zum Aufbau des DWDS-Wortprofils und ist stark von dem oben beschriebenen Szenario geprägt. So formte sich früh die Vision eines hybriden, leistungsstarken Parsing-Systems, dass auf FSMs beruht und die bestehenden Herausforderungen angeht. Wie schon Anfang der Arbeit angemerkt, ist die Entwicklung solch eines Systems ein sehr langer Weg. Daher wurde sich in dieser Arbeit zunächst auf einen regelbasierten Ansatz, auf das exakte Ausrechnen eines komplexen Problems, das Handling von Zwischenergebnissen und die Formulierung von Beschränkungen konzentriert. Über die Verbindung mit dem SCSP wurde der Grundstein für ein leistungsstarkes hybrides Parsing-System gelegt. Über den regelbasierten Teil kann ein erklärbarer Rahmen gegeben werden und es kann gewährleistet werden, dass bestimmte grammatikalische Regularitäten eingehalten sind. So besitzt beispielsweise der hybride Dependenzparser ParZu (Sennrich et al., 2009) die Eigenschaft, dass die Analyse in Teilbäume zerfällt, wenn gegebene Regularien verletzt sind. Zudem kann domänenunabhängiges linguistisches Wissen einbezogen werden, um die Robustheit bei Out-of-Domain-Texten zu erhöhen (siehe auch Plank und van Noord, 2010). Statistische Verfahren können hingegen dabei helfen, mit der Variabilität der Sprache umzugehen und den Suchraum über fundierte Vorhersagen einzuschränken.

Über die neuronalen Methoden steht zudem eine gebrauchsorientierte Semantik zur Verfügung, die es ermöglicht, Ambiguitäten akkurat aufzulösen. So könnten beispielsweise *kontextualisierte Word Embeddings* (siehe z. B. Miaschi und Dell'Orletta, 2020) über neuronalisierte FSMs einbezogen werden, um über implizit repräsentiertes linguistisches Wissen fundierte Entscheidungen zu treffen.

Innerhalb des WCDG-Ansatzes können datengetriebene Modelle in Form eines Prädiktors einbezogen werden (siehe Foth und Menzel, 2006 und Khmylko et al., 2009). Hierbei wird die Ausgabe eines Modells mithilfe von weichen Constraints in den Parser integriert, wodurch ein externer Vorschlag innerhalb einer Beschränkung berücksichtigt werden kann. Wenn ein neuronales Modell als Prädiktor einbezogen werden würde, ergäbe dies einen neurosymbolischen Ansatz. Die datengetriebenen Aspekte werden hierbei sehr locker mit der WCDG verbunden. Über den in dieser Arbeit entwickelten Ansatz ist hingegen eine engere Verzahnung der regelbasierten und datengetriebenen Methoden möglich und es kann eine höhere Ebene der Hybridisierung erreicht werden. So können im Rahmen des SCSPs statistische und neuronalisierte Modelle direkt als Constraints implementieren werden und es ist nicht mehr notwendig, sie als externe Ressourcen einzubinden. Innerhalb des Problemzerlegungsansatzes ist es dann möglich, die Modelle als Constraints gezielt auf Schichtteilprobleme oder Dependenzknoten-Teilprobleme anzuwenden. Es kann deren Relevanz gelernt werden (im Kontext statistischer Modelle vgl. Silfverberg und Lindén, 2010, Abschnitt 4.3) oder sie können selbst im Kontext der Problemzerlegung trainiert oder optimiert werden (vgl. Chiang et al., 2010). Hierdurch können sie gegenseitig Einfluss aufeinander nehmen. So verschmelzen die regelbasierten, statistischen und neuronalisierten Modelle innerhalb des Ansatzes. Die Verbindung von SCSP und FSMs bietet demnach ein ideales Framework, um im Bereich des Dependenzparsings dem aktuellen Trend zur Hybridisierung zu begegnen.

7.4 Ausblick

Im Rahmen dieser Arbeit wurde mithilfe von endlichen Techniken eine SCSP-Umgebung für ein spezielles realitätsnahes Problem erarbeitet. Interessant wäre die Implementierung einer allgemein gehaltenen generischen Umgebung für (weiche) Constraints, die Propagierungstechniken wie die Herstellung von

Knoten-, Kanten- oder generalisierter Kantenkonsistenz oder die perfekte Relaxierung unterstützt und auf FSMs beruht. Hier könnte von den SCSP-Erweiterungen des unendlichen Wertebereichs, der simultanen Repräsentation und der globalen Constraints Gebrauch gemacht werden. Zudem könnte eine Constraint-Programmiersprache einbezogen werden, über die der neue SCSP-Ansatz leicht auf verschiedenen Problemen erprobt und mit anderen Implementierungen wie der in Bistarelli et al. (2004) oder der in Delgado et al. (2005) verglichen werden könnte.

Innerhalb des WCDG-Ansatzes sind zusätzlich zur syntaktischen auch andere sprachliche Ebenen wie z. B. die der anaphorischen Beziehungen oder die der semantischen Rollen möglich. Diese können über verschiedene WCDG-Rollen realisiert werden. Der hier vorgestellte Ansatz wurde noch nicht auf diese Möglichkeit erweitert. Alternativ könnten aber auch Merkmalsstrukturen herangezogen werden, um zusätzliche sprachliche Ebenen umzusetzen. Diese Strukturen finden unter anderem innerhalb von unifikationsbasierten Parsingansätzen Anwendung wie z. B. in der *Lexical Functional Grammar* (LFG) oder in *PATR-II* (für einen Überblick siehe Shieber, 1986). Innerhalb der LFG wird beispielsweise eine separate Ebene der funktionalen Struktur angenommen, die grammatische Funktionen und Prädikat-Argument-Strukturen über Merkmalsstrukturen umsetzt. In Amtrup (2003) wird gezeigt, wie typisierte Merkmalsstrukturen als Gewichte über einen *Unifikationssemiring* (Unification Semiring) innerhalb von FSMs verwendet werden können. So wäre zu erforschen, inwieweit solch eine Struktur als c-Semiring in den neuen Ansatz integriert werden kann und welche Perspektiven und Möglichkeiten hieraus entstehen. Es wäre zu klären, ob über Merkmalsstrukturen verschiedene sprachliche Ebenen, aber auch überlokale Constraints umgesetzt werden können. Des Weiteren wäre zu untersuchen, ob es möglich ist, lexikalische Informationen und Beschränkungen wie z. B. die Kongruenz von Numerus, Kasus und Genus oder die Umsetzung von Subkategorisierungsrahmen auf die Semiring-Struktur auszugliedern. Hierbei läge die Hoffnung darin, Constraints einzusparen, FSMs kleiner zu halten und die Kompilationszeiten zu verkürzen und letztlich das simultane SCSP zu vereinfachen, das Schreiben von Regeln zu erleichtern und ein reichhaltigeres Parsingergebnis zurückzugeben.

Ein zentraler Punkt für weitere Forschungsarbeiten ist die Erweiterung des hier entwickelten Parsing-Ansatzes auf ein hybrides System und betrifft

das Beschleunigen der Analyse und die Verbesserung der Analysequalität (vgl. Abschnitt 7.3). Bisher ist die Laufzeit des Ansatzes für eine praktische Anwendung nicht ausreichend. Nichtsdestotrotz stellt er einen vielversprechenden Ausgangspunkt für effizientes Constraint-basiertes Parsing mit FSMs dar. So wäre zu erforschen wie Pruning-Strategien mithilfe von regelbasierten und datengetriebenen Methoden umgesetzt werden können. Hierbei sollte angestrebt werden, das Parsing bei (nahezu) gleichbleibender Parsingqualität zu beschleunigen und die Erklärbarkeit beizubehalten (siehe auch Abschnitt 6.5.1). Bei der Evaluation der Grammatik und des neuen Parsing-Ansatzes wurde der Fall betrachtet, dass die Wortart zu jedem Wort vorweg bekannt ist (Gold Part-of-Speech Tags). Auf Grundlage einer gut arbeitenden Pruning-Strategie wäre die Einbeziehung der Wortartendisambiguierung in der Praxis möglich und äußerst interessant für den Vergleich mit anderen Ansätzen. Gleiches gilt für die Disambiguierung von morphologischen oder lexikalischen Merkmalen. Die Informationen innerhalb der Grammatik reichen nicht immer aus, um zwischen alternativen Analysen korrekt zu entscheiden und somit Ambiguitäten akkurat aufzulösen zu können (siehe Abschnitt 5.7). So wäre auch zu erforschen, wie datengetriebene Methoden über weiche Constraints implementiert, einbezogen und mit dem Zerlegungsansatz in Einklang gebracht werden können, um die Parsingqualität zu verbessern. Hierbei sollte der Fokus auf die Einbeziehung neuronaler Aspekte liegen, um von den immensen Möglichkeiten des Deep Learnings profitieren zu können (vgl. Abschnitt 7.3).

Eine grundlegende Frage bei der Hybridisierung ist, in welchem konkreten Szenario sie erfolgt. Dass heißt, dass das Dependenzparsing nicht zum reinen Selbstzweck betrachtet werden sollte. Gerade im Hintergrund der neuronalen End-to-End-Systeme muss gezeigt werden, dass das Parsing im Kontext eines komplexen Szenarios einen echten Mehrwert bietet. Ein mögliches Szenario wäre die Erstellung von Wortprofilen auf Basis der Extraktion von syntaktischen Kookkurrenzen (siehe Abschnitt 7.3). Um jedoch einen größeren Fokus auf die Herausforderungen für ein hybrides neurosymbolisches System zu legen, sollte ein anderes anspruchsvolles Szenario gewählt werden und zwar die Grammatikprüfung und Grammatikfehlerdiagnose. Die Grammatikprüfung ist im Rahmen von kommerziellen Produkten eine besondere Herausforderung. Dort wird sie im Wesentlichen in dem Bereich der automatischen Qualitätssicherung von Texten angewendet. Beispielprodukte für

das Deutsche sind *Duden-Mentor* (`https://mentor.duden.de`) und *Language-Tool* (`https://languagetool.org`, Naber, 2003) aber auch *congree* (`https://www.congree.com`) im Kontext von Corporate Language. Wenn das System zu viele Fehlalarme verursacht oder eindeutige Fehler nicht erkennt oder Alarme ausgibt, die bezüglich der Eingabe völlig irreführend sind, oder im Rahmen der Interaktion mit dem User zyklische Alarme auslöst, führt dies schnell zu Unmut bei den Nutzern und schadet dem Produkt oder sogar der Marke. Im Fall der Grammatikfehlerdiagnose im schulischen Kontext, z. B. in Lernstandserhebungen, besteht zudem eine besondere Verantwortung. Das bedeutet, dass eine Lösung hier nur genau dann eingesetzt werden kann, wenn eine bestimmte Akkuratheit der Fehlererkennung tatsächlich gewährleistet ist. Es besteht also ein starker Bedarf an Erklärbarkeit, Nachvollziehbarkeit und Präzision. Der in dieser Arbeit entwickelte Parsingansatz kann analog zum WCDG-System grammatikalische Fehler tolerieren, wenn weiche Constraints verletzt sind und über die Constraint-Verletzungen eine direkte Fehlerdiagnose zurückgeben. So können Grammatikfehler kenntlich und erklärbar gemacht werden (vgl. Foth et al., 2005 und Menzel und Schröder, 1999 und Menzel, 1988). So ist der neue Ansatz ein idealer Ausgangspunkt, sich diesem anspruchsvollen Szenario in einem hybriden neurosymbolischen Setting zu stellen. Es wäre zu untersuchen, wie der neue Ansatz konkret in die Grammatikprüfung und Grammatikfehlerdiagnose eingebracht werden kann, um zusammen mit den Deep-Learning-Technologien und anderen datengetriebenen Methoden als hybrides System verlässliche und erklärbare Ergebnisse zu liefern und dem starken Anspruch an Akkuratheit zu genügen.

Literaturverzeichnis

Steven Abney. Partial parsing via finite-state cascades. *Journal of Natural Language Engineering*, 2(4):337–344, 1996.

Benedikt Adelmann, Melanie Andresen, Wolfgang Menzel und Heike Zinsmeister. Evaluation of out-of-domain dependency parsing for its application in a digital humanities project. In: Adrien Barbaresi, Hanno Biber, Friedrich Neubarth und Rainer Osswald (Hgg.): *Proceedings of the 14th Conference on Natural Language Processing (KONVENS 2018)*, S. 121–135, Wien, Österreich, 19. – 21. September 2018.

Salah Aït-Mokhtar und Jean-Pierre Chanod. Incremental finite-state parsing. In: *Proceedings of the 5th Conference on Applied Natural Language Processing (ANLP 1997)*, Washington, D.C., USA, 31. März – 3. April 1997.

Cyril Allauzen, Michael Riley, Johan Schalkwyk, Wojciech Skut und Mehryar Mohri. OpenFst: A general and efficient weighted finite-state transducer library. In: Jan Holub und Jan Žďárek (Hgg.): *Implementation and Application of Automata – 12th International Conference, CIAA 2007, Praque, Czech Republic, July 16–18, 2007, Revised Selected Papers*, Band 4783 von *Lecture Notes in Computer Science (LNCS)*, S. 11–23. Springer, Berlin/Heidelberg, 2007.

Cyril Allauzen, Michael Riley und Johan Schalkwyk. A generalized composition algorithm for weighted finite-state transducers. In: *Proceedings of the 10th Annual Conference of the International Speech Communication Association (INTERSPEECH 2009)*, S. 1203–1206, Brighton, Vereinigtes Königreich, 6.–10. September 2009.

Marco Almeida, Nelma Moreira und Rogério Reis. On the performance of automata minimization algorithms. In: Arnold Beckmann, Costas Dimitracopoulos und Benedikt Löwe (Hgg.): *Local Proceedings of the 4th Conference on Computation in Europe: Logic and Theory of Algorithms (CiE 2008)*, S. 3–14, Athen, Griechenland, 2008.

Jérôme Amilhastre, Philippe Janssen und Marie-Catherine Vilarem. FA minimization heuristics for a class of finite languages. In: Oliver Boldt und Helmut Jürgensen (Hgg.): *Automata Implementation, 4th International Workshop on Implementing Automata, WIA'99, Potsdam, Germany, July 17–19, 1999 Revised Papers*, Band 2214 von *Lecture Notes in Computer Science (LNCS)*, S. 1–12. Springer, Berlin/Heidelberg, 2001.

Jérôme Amilhastre, Hélène Fargier und Pierre Marquis. Consistency restoration and explanations in dynamic CSPs – application to configuration. *Artificial Intelligence*, 135(1-2):199–234, 2002.

Jan W. Amtrup. Feature structures as weights in finite state morphology. In: *Proceedings of the EACL 2003 Workshop on Finite-State Methods in Natural Language Processing (FSMNLP 2003)*, Budapest, Ungarn, 12.–17. April 2003.

Arturo Argueta und David Chiang. Composing finite state transducers on GPUs. In: *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (ACL 2018)*, S. 2697–2705, Melbourne, Australien, Juli 2018.

Roman Barták. Modelling soft constraints: A survey. *Neural Network World*, 12(5):421–431, 2002.

Roman Barták, Miguel A. Salido und Francesca Rossi. New trends in constraint satisfaction, planning, and scheduling: A survey. *The Knowledge Engineering Review*, 25(3):249–279, 2004.

Kenneth R. Beesley und Lauri Karttunen. *Finite State Morphology*. CSLI Publications, 2003.

Jean Berstel. *Transductions and context-free languages*. Teubner, Stuttgart, 1979.

Niels Beuck und Wolfgang Menzel. Structural prediction in incremental dependency parsing. In: Alexander Gelbukh (Hg.): *Computational Linguistics and Intelligent Text Processing – 14th International Conference, CICLing 2013, Samos, Greece, March 24–30, 2013, Proceedings, Part I*, Band 7816 von *Lecture Notes in Computer Science (LNCS)*, S. 245–257. Springer, Berlin/Heidelberg, 2013.

Stefano Bistarelli und Fabio Gaducci. Enhancing constraints manipulation in semiring-based formalisms. In: Gerhard Brewka, Silvia Coradeschi, Anna Perini und Paolo Traverso (Hgg.): *Proceedings of the 17th European Conference on Artificial Intelligence (ECAI 2006)*, Band 141 von *Frontiers in Artificial Intelligence and Applications*, S. 63–67, Riva del Garda, Italien, 2006. IOS Press.

Stefano Bistarelli und Francesca Rossi. About arc-consistency in semiring-based constraint problems. In: *Proceedings of the 5th International Symposium on Artificial Intelligence and Mathematics (ISAIM 1998)*, Fort Lauderdale, Florida, USA, 4.–6. Januar 1998.

Stefano Bistarelli, Ugo Montanari und Francesca Rossi. Constraint solving over semirings. In: *Proceedings of the 14th International Joint Conference on Artificial Intelligence (IJCAI 1995)*, Band 1, S. 624–630, Montreal, Québec, Kanada, 20.–25. August 1995. Morgan Kaufman.

Stefano Bistarelli, Ugo Montanari und Francesca Rossi. Semiring-based constraint satisfaction and optimization. *Journal of the ACM (JACM)*, 44(2): 201–236, 1997.

Stefano Bistarelli, Ugo Montanari, Francesca Rossi, Thomas Schiex, Gerard Verfaillie und Hélène Fargier. Semiring-based CSPs and valued CSPs: Frameworks, properties and comparison. *Constraints*, 4(3):199–240, 1999.

Stefano Bistarelli, Philippe Codognet, Yan Georget und Francesca Rossi. Abstracting soft constraints. In: Krzysztof R. Apt, Eric Monfroy, Antonis C. Kakas und Francesca Rossi (Hgg.): *New Trends in Constraints: Joint ERCIM/Compulog Net Workshop, Paphos, Cyprus, October 25–27, 1999, Selected Papers*, Band 1865 von *Lecture Notes in Computer Science (LNCS)*, S. 108–133. Springer, Berlin/Heidelberg, 2000.

Stefano Bistarelli, Thom Frühwirth, Michael Marte und Francesca Rossi. Soft constraint propagation and solving in constraint handling rules. *Computational Intelligence: Special Issue on Preferences in AI and CP*, 20(2): 287–307, 2004.

Stefano Bistarelli, Maria Silvia Pini, Francesca Rossi und Kristen Brent Venable. Positive and negative preferences. In: *Proceedings of 7th International CP 2005 Workshop on Preferences and Soft constraints (Soft 2005)*, Sitges, Spanien, 1. Oktober 2005.

Beate Bollig und Ingo Wegener. Improving the variable ordering of OBDDs is NP-complete. *IEEE Transactions on Computers*, 45(9):993–1002, 1996.

Guillaume Bonfante und Joseph Le Roux. Intersection optimization is NP-complete. In: Thomas Hanneforth und Kay Michael Würzner (Hgg.): *Finite-State Methods and Natural language Processing: revised papers / 6th International Workshop on Finite-State Methods and Natural language Processing (FSMNLP 2007)*, S. 74–85. Potsdam University Press, 2008.

Craig Boutilier, Ronen I. Brafman, Carmel Domshlak, Holger H. Hoos und David Poole. A tool for representing and reasoning with conditional ceteris paribus preference statements. *Journal of Artificial Intelligence Research*, 21:135–191, 2004.

Ferdinand Börner, Andrei Bulatov, Peter Jeavons und Andrei Krokhin. Quantified constraints: Algorithms and complexity. In: Matthias Baaz und Johann M. Makowsky (Hgg.): *Computer Science Logic – 17th International Workshop, CSL 2003, 12th Annual Conference of the EACSL, and 8th Kurt Gödel Colloquium, KGC 2003, Vienna, Austria, August 25–30, 2003, Proceedings*, Band 2803 von *Lecture Notes in Computer Science (LNCS)*, S. 58–70. Springer, Berlin/Heidelberg, 2003.

Kenil Cheng, Wei Xia und Roland Yap. Space-time tradeoffs for the regular constraint. In: Michela Milano (Hg.): *Principles and Practice of Constraint Programming – CP 2012, 18th International Conference, CP 2012, Québec City, QC, Canada, October 8–12, 2012, Proceedings*, Band 7514 von *Lecture Notes in Computer Science (LNCS)*, S. 223–237. Springer, Berlin/Heidelberg, 2012.

Kenil C. K. Cheng und Roland H. C. Yap. An MDD-based generalized arc consistency algorithm for positive and negative table constraints and some global constraints. *Constraints*, 15(2):265–304, 2010.

David Chiang, Jonathan Graehl, Kevin Knight, Adam Pauls und Sujith Ravi. Bayesian inference for finite-state transducers. In: *Proceedings of the 2010 Conference of the North American Chapter of the Association for Computational Linguistics – Human Language Technologies (NAACL-HLT 2010)*, S. 447–455, Los Angeles, Kalifornien, USA, 2.–4. Juni 2010.

Jan Daciuk, Richard E. Watson und Bruce W. Watson. Acyclic finite-state automata and transducers. In: Lauri Karttunen und Kemal Oflazer (Hgg.): *Proceedings of the International Workshop on Finite-State Methods in Natural Language Processing (FSMNLP 1998)*, Ankara, Türkei, 30. Juni – 1. Juli 2005.

Michael Daum, Kilian A. Foth und Wolfgang Menzel. Automatic transformation of phrase treebanks to dependency trees. In: *Proceedings of the 4th international conference on Language Resources and Evaluation (LREC 2004)*, S. 1149–1152, Lissabon, Portugal, 26.–28. Mai 2004. European Language Resources Association.

Artur d'Avila Garcez und Luís C. Lamb. Neurosymbolic AI: the 3rd wave. *CoRR*, abs/2012.05876, 2020.

Ralph Debusmann. Scrambling as the intersection of relaxed context-free grammars in a model-theoretic grammar formalism. In: *Proceedings of the ESSLLI 2007 Workshop on Model-Theoretic Syntax at 10*, Dublin, Irland, 6.–17. August 2007a.

Ralph Debusmann. The complexity of first-order extensible dependency grammar. Technical report, Programming Systems Lab, Universität des Saarlandes, 2007b.

Ralph Debusmann, Denys Duchier, Alexander Koller, Marco Kuhlmann, Gert Smolka und Stefan Thater. A relational syntax-semantics interface based on dependency grammar. In: *Proceedings of the 20th International Conference on Computational Linguistics (COLING 2004)*, S. 176–182, Geneva, Schweiz, 23.–27. August 2004.

Rina Dechter. *Constraint Processing*. Morgan Kaufmann Publishers, Inc., 2003.

Alberto Delgado, Carlos Alberto Olarte, Jorge Andrés Pérez und Camilo Rueda. Implementing semiring-based constraints using Mozart. In: Peter Van Roy (Hg.): *Multiparadigm Programming in Mozart/Oz, Second International Conference, MOZ 2004, Charleroi, Belgium, October 7–8, 2004, Revised Selected and Invited Papers*, Band 3389 von *Lecture Notes in Computer Science (LNCS)*, S. 224–236. Springer, Berlin/Heidelberg, 2005.

Sophie Demassey, Gilles Pesant und Louis-Martin Roussea. A cost-regular based hybrid column generation approach. *Constraints*, 11(4):315–333, 2006.

Jörg Didakowski. SynCoP – combining syntactic tagging with chunking using weighted finite state transducers. In: Thomas Hanneforth und Kay Michael Würzner (Hgg.): *Finite-State Methods and Natural language Processing, 6th International Workshop, FSMNLP 2007, Potsdam, Germany, September 14–16, revised papers*, S. 107–118. Potsdam University Press, Potsdam, 2008a.

Jörg Didakowski. Local syntactic tagging of large corpora using weighted finite state transducers. In: *Text Resources and Lexical Knowledge – Selected Papers from the 9th Conference on Natural Language Processing KONVENS 2008*, Text, Translation, Computational Processing, S. 65–78. Mouton de Gruyter, Berlin/New York, 2008b.

Jörg Didakowski und Marko Drotschmann. Proper noun recognition and classification using weighted finite state transducers. In: Jakub Piskorski, Bruce Watson und Anssi Mikael Yli-Jyrä (Hgg.): *Finite-State Methods and Natural Language Processing, Post-Proceedings of the 7th International Workshop FSMNLP 2008*, Band 191 von *Frontiers in Artificial Intelligence and Applications*, S. 34–45. IOS Press, 2008.

Jörg Didakowski, Alexander Geyken und Thomas Hanneforth. Eigennamenerkennung mit großen lexikalischen Ressourcen. In: Miriam Butt (Hg.):

Proceedings of the 8th Conference on Natural Language Processing (KONVENS 2006), S. 9–14, Konstanz, Deutschland, 4.–7. Oktober 2006. Universität Konstanz.

Jörg Didakowski, Alexander Geyken und Thomas Hanneforth. Eigennamenerkennung zwischen morphologischer Analyse und Part-of-Speech Tagging: ein automatentheoriebasierter Ansatz. *Zeitschrift für Sprachwissenschaft*, 26(4):157–186, 2007.

Jörg Didakowski. Robust parsing as a constraint optimization problem within a finite-state approach. In: Thomas Hanneforth und Gisbert Fanselow (Hgg.): *Language and Logos: Studies in Theoretical and Computational Linguistics. Festschrift for Peter Staudacher on his 70th birthday*, Band 72 von *studia grammatica*, S. 269–296. Akademie Verlag, Berlin, 2010.

Jörg Didakowski und Alexander Geyken. From DWDS corpora to a german word profile – methodological problems and solutions. In: Andrea Abel und Lothar Lemnitzer (Hgg.): *Vernetzungsstrategien, Zugriffsstrukturen und automatisch ermittelte Angaben in Internetwörterbüchern*, Band 2/2014 von *OPAL - Online publizierte Arbeiten zur Linguistik*, S. 39–47. IDS Mannheim, 2013.

Jörg Didakowski und Nadja Radtke. Nutzung des DWDS-Wortprofils beim Aufbau eines lexikalischen Informationssystems zu deutschen Stützverbgefügen. In: Andrea Abel, Chiara Vettori und Natascia Ralli (Hgg.): *Proceedings of the 16th EURALEX International Congress (EURALEX 2014)*, S. 345–353, Bozen, Italien, 15.–19. Juli 2014. EURAC research.

Jörg Didakowski und Nadja Radtke. Deutsche Stützverbgefüge in Referenz- und Spezialkorpora: Vergleichsstudien mit dem DWDS-Wortprofil. In: Eric Fuß, Marek Konopka, Beata Trawiński und Ulrich H. Waßner (Hgg.): *Grammar and Corpora 2016*, S. 121–144, Heidelberg, 2018. Heidelberg University Publishing.

Jörg Didakowski und Nadja Radtke. Verwendung der deutschen Stützverbgefüge mit Adjektiven und ihre Ermittlung mithilfe des DWDS-Wortprofils. In: Sabine De Knop und Manon Hermann (Hgg.): *Funktionsverbgefüge im Fokus – Theoretische, didaktische und kontrastive Perspektiven*, S. 65–86. de Gruyter, Berlin, 2020.

Jörg Didakowski, Lothar Lemnitzer und Alexander Geyken. Automatic example sentence extraction for a contemporary german dictionary. In: Ruth Vatvedt Fjeld und Julie Matilde Torjusen (Hgg.): *Proceedings of the 15th EURALEX International Congress (EURALEX 2012)*, S. 343–349, Oslo, Norwegen, 7.-11. August 2012. Department of Linguistics and Scandinavian Studies, University of Oslo.

Carmel Domshlak, Francesca Rossi, Kristen Brent Venable und Toby Walsh. Reasoning about soft constraints and conditional preferences: complexity results and approximation techniques. In: *Proceedings of the 18th International Joint Conference on Artificial Intelligence (IJCAI 2003)*, S. 215–220, Acapulco, Mexico, 2003.

Denys Duchier. Axiomatizing dependency parsing using set constraints. In: *Proceedings of the Sixth Meeting on Mathematics of Language*, S. 115–126, Orlando, Florida, USA, 23.–25. Juli 1999.

Denys Duchier und Ralph Debusmann. Topological dependency trees: A constraint-based account of linear precedence. In: *Proceedings of the 39th Annual Meeting of the Association for Computational Linguistics (ACL 2001)*, S. 180–187, Toulouse, Frankreich, 6.–11. Juli 2001.

Jason Eisner. Simpler and more general minimization for weighted finite-state automata. In: *Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics – Human Language Technologies (NAACL-HLT 2003)*, S. 64–71, Edmonton, Alberta, Canada, 27. Mai – 1. Juni 2003.

Bilal Elghadyry, Faissal Ouardi und Sébastien Vérel. Composition of weighted finite transducers in MapReduce. *Journal of Big Data*, 8(1):1–15, Januar 2021.

Gisbert Fanselow. „Fakten, Fakten, Fakten!“. *Linguistische Berichte*, 200: 481–493, 2004.

Kilian Foth, Wolfgang Menzel und Ingo Schröder. Robust parsing with weighted constraints. *Natural Language Engineering*, 11(1):1–25, 2005.

Kilian A. Foth. *Eine umfassende Constraint-Dependenz-Grammatik des Deutschen*, 2006a. Verfügbar unter `http://edoc.sub.uni-hamburg.de/`

informatik/volltexte/2014/204/pdf/foth_eine_umfassende_.pdf, Abrufdatum: 03.03.2022.

Kilian A. Foth. *Hybrid Methods of Natural Language Analysis*. Dissertation, University of Hamburg, Department of Computer Science, 2006b.

Kilian A. Foth und Wolfgang Menzel. Hybrid parsing: Using probabilistic models as predictors for a symbolic parser. In: *Proceedings of the 21st International Conference on Computational Linguistics and the 44th annual meeting of the Association for Computational Linguistics (COLING-ACL 2006)*, Sydney, Australien, 17.—21. Juli 2006.

Kilian A. Foth, Wolfgang Menzel, Horia F. Pop und Ingo Schröder. An experiment on incremental analysis using robust parsing techniques. In: *Proceedings of the 18th International Conference on Computational Linguistics (COLING 2000)*, Band 2, S. 1026–1030, Saarbrücken, Deutschland, 31. Juli – 4. August 2000a.

Kilian A. Foth, Ingo Schröder und Wolfgang Menzel. A transformation-based parsing technique with anytime properties. In: *Proceedings of the 6th International Workshop on Parsing Technologies (IWPT 2000)*, S. 89–100, Trento, Italien, 3.–25. Februar 2000b.

Kilian A. Foth, Michael Daum und Wolfgang Menzel. Subtree parsing to speed up deep analysis. In: *Proceedings of the 8th International Workshop of Parsing Technologies (IWPT 2003)*, S. 91–103, Nancy, Frankreich, 23.–25. April 2003.

Kilian A. Foth, Michael Daum und Wolfgang Menzel. A broad-coverage parser for german based on defeasible constraints. In: *Proceedings of the 7th Conference on Natural Language Processing (KONVENS 2004)*, S. 45–52, Wien, Österreich, 14.–17. September 2004.

Haim Gaifman. Dependency systems and phrase-structure systems. *Information and Control*, 8:304–337, 1965.

Alexander Geyken und Thomas Hanneforth. TAGH: a complete morphology for german based on weighted finite state automaton. In: *Proceedings of the 5th International Workshop on Finite-State Methods and Natural Language Processing (FSMNLP 2005)*, Helsinki, Finland, 1.–2. September 2005.

Alexander Geyken, Jörg Didakowski und Alexander Siebert. Generation of word profiles on the basis of a large and balanced german corpus. In: Elisenda Bernal und Janet DeCesaris (Hgg.): *Proceedings of the 13th EURALEX International Congress (EURALEX 2008)*, Band 20 von *Sèrie Activitats*, S. 158–159, Barcelona, Spanien, 15.–19. Juli 2008. Documenta Universitaria.

Alexander Geyken, Jörg Didakowski und Alexander Siebert. Generation of word profiles for large german corpora. In: *Corpus Analysis and Variation in Linguistics*, Band 1 von *Tokyo University of Foreign Studies, Studies in Linguistics*, S. 141–157. John Benjamins, Amsterdam/Philadelphia, 2009.

Alexander Geyken, Adrien Barbaresi, Jörg Didakowski, Bryan Jurish, Frank Wiegand und Lothar Lemnitzer. Die Korpusplattform des „Digitalen Wörterbuchs der deutschen Sprache“ (DWDS). *Zeitschrift für germanistische Linguistik*, 45(2):327–344, 2017.

Filip Ginter, Sampo Pyysalo, Jorma Boberg und Tapio Salakoski. Regular approximation of link grammar. In: Tapio Salakoski, Filip Ginter, Sampo Pyysalo und Tapio Pahikkala (Hgg.): *5th International Conference on NLP, FinTAL 2006 Turku, Finland, August 23–25, 2006 Proceedings*, Band 4139 von *Lecture Notes in Computer Science (LNCS)*, S. 564–575. Springer, Berlin/Heidelberg, 2006.

Goran Glavaš und Ivan Vulić. Is supervised syntactic parsing beneficial for language understanding tasks? An empirical investigation. In: *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics (EACL 2021)*, S. 3090–3104, Online, 19. – 23. April 2021.

Keith Golden und Wanlin Pang. Constraint reasoning over strings. In: Francesca Rossi (Hg.): *Principles and Practice of Constraint Programming – CP 2003, 9th International Conference, CP 2003, Kinsale, Ireland, September 29 – October 3, 2003. Proceedings*, Band 2833 von *Lecture Notes in Computer Science (LNCS)*, S. 377–391. Springer, Berlin/Heidelberg, 2006.

Gregory Grefenstette. The future of linguistics and lexicographers: Will there be lexicographers in the year 3000? In: Thierry Fontenelle, Philippe Hiligsmann, Archibald Michiels, André Moulin und Siegfried Theissen

(Hgg.): *Proceedings of the 8th EURALEX International Congress (EURALEX 1998)*, S. 25–41, Liège, Belgien, August 1998. Euralex.

Gregory Grefenstette. Light parsing as finite state filtering. In: András Kornai (Hg.): *Extended Finite State Models of Language*, Studies in Natural Language Processing, S. 86–94. Cambridge University Press, 1999.

Tarik Hadzic, Esben Rune Hansen und Barry O'Sullivan. On automata, MDDs and BDDs in constraint satisfaction. In: *Proceedings of the ECAI 2008 Workshop on Inference Methods based on Graphical Structures of Knowledge*, Patras, Griechenland, 22. Juli 2008.

Thomas Hanneforth. fsm2 – a scripting language interpreter for manipulating weighted finite-state automata. In: Anssi Yli-Jyrä, András Kornai, Jacques Sakarovitch und Bruce Watson (Hgg.): *Finite-State Methods and Natural Language Processing 8th International Workshop, FSMNLP 2009, Pretoria, South Africa, July 21–24, 2009, Revised Selected Papers*, Band 6062 von *Lecture Notes in Computer Science (LNCS)*, S. 13–30. Springer, Berlin/Heidelberg, 2010.

Thomas Hanneforth und Kay-Michael Würzner. Statistical language models within the algebra of weighted rational languages. *Acta Cybernetica*, 19 (2):313–356, 2009.

Esben Rune Hansen und Henrik Reif Andersen. Interactive configuration with regular string constraints. In: *Proceedings of the 22th Conference on Artificial Intelligence (AAAI 2007)*, S. 217–223, Vancouver, British Columbia, Kanada, 22.–26. Juli 2007. AAAI Press.

Esben Rune Hansen und Peter Tiedemann. Improving the performance of interactive configuration with regular string constraints. In: *Proceedings of the 20th IEEE International Conference on Tools with Artificial Intelligence (ICTAI 2008)*, Band 1, S. 3–10, Dayton, Ohio, USA, 3.–5. November 2008.

Karin Harbusch. The relation between tree-adjoining grammars and constraint dependency grammars. In: *Proceedings of the 5th Meeting on Mathematics of Language (MOL5)*, S. 38–45, Saarbrücken, Deutschland, 1997.

Mary P. Harper und Randall A. Helzerman. Extensions to constraint dependency parsing for spoken language processing. *Computer Speech and Language*, 9:187–234, 1995.

Mary P. Harper, Leah H. Jamieson, Carla B. Zoltowski und Randall A. Helzerman. Semantics and constraint parsing of word graphs. In: *Proceedings of the International Conference on Acoustics, Speech, and Signal Processing (ICASSP 1993)*, Band 2, S. 63–66, Minneapolis, Minnesota, USA, 27.–30. April 1993.

Udo Hebisch und Hanns Joachim Weinert. *Semirings – Algebraic Theory and Applications in Computer Science*, Band 5 von *Series in Algebra*. World Scientific, Singapore, 1998.

Johannes Heinecke, Jürgen Kunze, Wolfgang Menzel und Ingo Schröder. Eliminative parsing with graded constraints. In: *Proceedings of the 17th International Conference on Computational Linguistics and 36th Annual Meeting of the ACL (COLING-ACL 1998)*, S. 526–530, Montreal, Québec, Kanada, 10.–14. August 1998.

Randall A. Helzerman und Mary P. Harper. Log time parsing on the MasPar MP-1. In: *Proceedings of the 6th International Conference on Parallel Processing (ICPP 1992)*, S. 209–217, 1992.

Randall A. Helzerman und Mary P. Harper. MUSE CSP: An extension to the constraint satisfaction problem. *Journal of Artificial Intelligence Research*, 5:5–239, 1996.

Randall A. Helzerman, Mary P. Harper und Carla B. Zoltowski. Parallel parsing of spoken language. In: *Proceedings of the 4th Symposium on the Frontiers of Massively Paralel Computation (Frontiers 1992)*, S. 528–530, 19.–21. Oktober 1992.

Mans Hulden. Regular expressions and predicate logic in finite-state language processing. In: Jakub Piskorski, Bruce Watson und Anssi Mikael Yli-Jyrä (Hgg.): *Finite-State Methods and Natural Language Processing, Post-Proceedings of the 7th International Workshop FSMNLP 2008*, Band 191 von *Frontiers in Artificial Intelligence and Applications*, S. 67–77, Ispra, Italien, 11.–12. September 2008. IOS Press.

Mans Hulden. Foma: a finite-state compiler and library. In: *Proceedings of the 12th Conference of the European Chapter of the Association for Computational Linguistics (EACL 2009)*, S. 29–32, Athen, Griechenland, 30. März – 3. April 2009.

Jeffrey D. Ullman John E. Hopcroft. *Introduction to Automata Theory, Languages, and Computation.* Addison-Wesley, 1979.

Sylvain Kahane. Bubble trees and syntactic representations. In: T. Becker und H.-U. Krieger (Hgg.): *Proceedings of the 5th Meeting on Mathematics of Language (MOL5)*, S. 70–76, Saarbrücken, Deutschland, 1997.

Ronald M. Kaplan und Martin Kay. Regular models of phonological rule systems. *Computational Linguistics*, 20(3):331–378, 1994.

Fred Karlsson. Constraint grammar as a framework for parsing running text. In: Hans Karlgren (Hg.): *Proceedings of the 13th International Conference on Computational Linguistics (COLING 1990)*, Band 3, S. 168–173, Helsinki, Finland, 1990.

Fred Karlsson. Syntactic recursion and iteration. In: Harry van der Hulst (Hg.): *Recursion and Human Language*, S. 43–67. Mouton de Gruyter, Berlin/New York, 2010.

Lauri Karttunen. The replace operator. In: *Proceedings of the 33rd Annual Meeting of the Association for Computational Linguistics (ACL 1995)*, S. 16–23, Cambridge, Massachusetts, USA, 26.–30. Juni 1995.

Lauri Karttunen. Directed replacement. In: *Proceedings of the 34th Annual Meeting of the Association for Computational Linguistics (ACL 1996)*, S. 108–115, Santa Cruz, Kalifornien, USA, 24.–27. Juni 1996.

Lauri Karttunen. The proper treatment of optimality theory in computational phonology. In: Lauri Karttunen und Kemal Oflazer (Hgg.): *Proceedings of the International Workshop on Finite-State Methods in Natural Language Processing (FSMNLP 1998)*, S. 1–12, Ankara, Türkei, 29. Juni – 1. Juli 1998.

Lauri Karttunen, Kimmo Koskenniemi und Ronald M. Kaplan. A compiler for two-level phonological rules. In: Mary Dalrymple, Ronald M. Kaplan,

Lauri Karttunen, Kimmo Koskenniemi, Sami Shaio und Michael Wescoat (Hgg.): *Tools for Morphological Analysis*, Band 87-108 von *CSLI Reports*, S. 1–61. Center for the Study of Language and Information, Stanford University, Palo Alto, California, Palo Alto, Kalifornien, USA, 1987.

Lauri Karttunen, Jean-Pierre Chanod, Gregory Grefenstette und Anne Schiller. Regular expressions for language engineering. *Natural Language Engineering*, 2(4):305–328, 1996.

Lidia Khmylko, Kilian A. Foth und Wolfgang Menzel. Co-parsing with competitive models. In: *Proceedings of the 11th International Workshop on Parsing Technologies (IWPT 2009)*, S. 99–107, Paris, Frankreich, 7.–9. Oktober 2009.

Adam Kilgarriff, Pavel Rychly, Pavel Smrz und David Tugwell. The Sketch Engine. In: *Proceedings of the 11th EURALEX international congress (EURALEX 2004)*, S. 105–116, Lorient, Frankreich, 6.–10. July 2004.

Adam Kilgarriff, Pavel Rychlý, Miloš Jakubíček, Vojtěch Kovář, Vít Baisa und Lucia Kocincová. Extrinsic corpus evaluation with a collocation dictionary task. In: *Proceedings of the 9th International Conference on Language Resources and Evaluation (LREC 2014)*, S. 545–552, Reykjavik, Island, May 2014. European Language Resources Association.

Stephen C. Kleene. Representation of events in nerve nets and finite automata. In: C. E. Shannon und J. McCarthy (Hgg.): *Automata Studies*, S. 3–41. Princeton Univ. Press, Princeton, New Jersey, USA, 1956.

Kimmo Koskenniemi. *Two-level morphology: a general computational model for word-form recognition and production.* Number 11 in Publications. University of Helsinki, Department of General Linguistics, 1983.

Kimmo Koskenniemi. Finite-state parsing and disambiguation. In: Hans Karlgren (Hg.): *Proceedings of the 13th International Conference on Computational Linguistics (COLING 1990)*, Band 2, S. 229–232, Helsinki, Finland, 1990.

Kimmo Koskenniemi. Representations and finite-state components in natural language. In: Emmanuel Roche und Yves Schabes (Hgg.): *Finite-State*

Language Processing, S. 99–116. MIT Press, Cambridge, Massachusetts, USA, 1997.

Kimmo Koskenniemi, Pasi Tapanainen und Atro Voutilainen. Compiling and using finite-state syntactic rules. In: *Proceedings of the 14th conference on Computational linguistics (COLING 1992)*, Band 1, S. 156–162, Nantes, Frankreich, 23.–28. August 1992.

Werner Kuich und Arto Salomaa. *Semirings, Automata and Languages*, Band 5 von *EATCS Monographs on Theoretical Computer Science*. Springer, Berlin/Heidelberg, 1985.

Artur Kulmizev, Miryam de Lhoneux, Johannes Gontrum, Elena Fano und Joakim Nivre. Deep contextualized word embeddings in transition-based and graph-based dependency parsing – a tale of two parsers revisited. In: *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP 2019)*, S. 2755–2768, Hong Kong, China, 3. –7. November 2019.

Franz Kutschera und Alfred Breitkopf. *Einführung in die moderne Logik*. Alber, Freiburg und München, Deutschland, 7. Edition, 2000.

Christine Köhn und Wolfgang Menzel. Towards parsing language learner utterances in context. In: *Proceedings of the 26th International Conference of the German Society for Computational Linguistics and Language Technology (GSCL 2015)*, S. 144–153, Essen, Deutschland, 30. September – 2. Oktober 2015. GSCL e.V.

Sandra Kübler, Ryan McDonald und Joachim Nivre. *Dependency Parsing*, Band 2 von *Synthesis Lectures on Human Language Technologies*. Morgan & Claypool, 2009.

Mikael Z. Lagerkvist. *Techniques for Efficient Constraint Propagation*. Dissertation, Licentiate Thesis, KTH Royal Institute of Technology, Stockholm, Sweden, 2008.

Lothar Lemnitzer, Christian Pölitz, Jörg Didakowski und Alexander Geyken. Combining a rule-based approach and machine learning in a good-example

extraction task for the purpose of lexicographic work on contemporary standard german. In: Iztok Kosem, Miloš Jakubíček, Jelena Kallas und Simon Krek (Hgg.): *Proceedings of the eLex 2015 conference*, S. 21–31, Herstmonceux Castle, Vereinigtes Königreich, 11.-13. August 2015. Trojina, Institute for Applied Slovene Studies/Lexical Computing Ltd, Ljubljana/Brighton.

Chu-Cheng Lin, Hao Zhu, Matthew R. Gormley und Jason Eisner. Neural finite-state transducers: Beyond rational relations. In: *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics – Human Language Technologies (NAACL-HLT 2019)*, Band 1, S. 272–283, Minneapolis, Minnesota, USA, 2.–7. Juni 2019.

Krister Lindén. Guessers for finite-state transducer lexicons. In: Alexander Gelbukh (Hg.): *Computational Linguistics and Intelligent Text Processing – 10th International Conference, CICLing 2009, Mexico City, Mexico, March 1–7, 2009. Proceedings*, Band 5449 von *Lecture Notes in Computer Science (LNCS)*, S. 158–169. Springer, Berlin/Heidelberg, 2009.

Alan K. Mackworth. On reading sketch maps. In: *Proceedings of the 7th International Joint Conference on Artificial Intelligence (IJCAI 1977)*, Band 2, S. 598–606, Cambridge, Massachusetts, USA, 1977.

Hiroshi Maruyama. Structural disambiguation with constraint propagation. In: *Proceedings of the 28th annual meeting on Association for Computational Linguistics (ACL 1990)*, S. 31–38, Pittsburgh, Pennsylvania, USA, 6.–9. Juni 1990a.

Hiroshi Maruyama. Constraint dependency grammar. Technical report, RT0044, IBM Research, Tokyo Research Laboratory, 1990b.

Hiroshi Maruyama. Constraint dependency grammar and its weak generative capacity. *Computer Software*, 7(3), 1990c.

Michael C. McCord, J. William Murdock und Branimir K. Boguraev. Deep parsing in Watson. *IBM Journal of Research and Development*, 56(3/4): 1–15, Mai 2012.

Patrick McCrae, Kilian Foth und Wolfgang Menzel. Modelling global phenomena with extended local constraints. In: Jørgen Villadsen und Henning Christiansen (Hgg.): *Proceedings of the 5th International Workshop on Constraints and Language Processing (CSLP 2008)*, S. 48–60, Hamburg, Deutschland, 11.–15. August 2008.

Warren S. McCulloch und Walter H. Pitts. A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, 5:115–133, 1943.

Ryan McDonald. *Discriminative Learning and Spanning Tree Algorithms for Dependency Parsing*. Dissertation, University of Pennsylvania, 2006.

Beáta Megyesi und Sara Rydin. Towards a finite-state parser for swedish. In: Torbjørn Nordgård (Hg.): *Proceedings of the 12th Nordic Conference of Computational Linguistics (NODALIDA 1999)*, Throndheim, Norwegen, 9.–10. Dezember 1999.

Igor Mel'čuk. *Dependency Syntax: Theory and Practice*. SUNY Series in Linguistics. State University of New York Press, Albany, 1988.

Igor Mel'čuk. *Dependency in linguistic description – It depends! [the main principle of human science—and of human life]*, 1998. Unveröffentlichtes Manuskript, Verfügbar unter `http://olst.ling.umontreal.ca/pdf/Dependency.pdf`, Abrufdatum: 15.03.2022.

Julien Menana und Sophie Demassey. Sequencing and counting with the multicost-regular constraint. In: Willem-Jan van Hoeve und John N. Hooker (Hgg.): *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems, 6th International Conference, CPAIOR 2009 Pittsburgh, PA, USA, May 27–31, 2009 Proceedings*, Band 5547 von *Lecture Notes in Computer Science (LNCS)*, S. 178–192. Springer, Berlin/Heidelberg, 2009.

Wolfgang Menzel. Error diagnosing and selection in a training system for second language learning. In: *Proceedings 12th International Conference on Computational Linguistics (COLING 1988)*, S. 414–419, Budapest, Ungarn, 22.–27. August 1988.

Wolfgang Menzel. Robust processing of natural language. In: Ipke Wachsmuth, Claus Rollinger und Wilfried Brauer (Hgg.): *KI-95: Advances in Artificial Intelligence – 19th Annual German Conference on Artificial Intelligence Bielefeld, Germany, September 11–13, 1995 Proceedings*, Band 981 von *Lecture Notes in Computer Science (LNCS)*, S. 19–34. Springer, Berlin/Heidelberg, 1995.

Wolfgang Menzel. Constraint satisfaction for robust parsing of spoken language. *Journal of Experimental & Theoretical Artificial Intelligence*, 10 (1):77–89, 1998.

Wolfgang Menzel und Ingo Schröder. Decision procedures for dependency parsing using graded constraints. In: Alain Polguère und Sylvain Kahane (Hgg.): *Proceedings of the COLING-ACL 1998 Workshop on Processing of Dependency-Based Grammars*, S. 78–87, Montreal, Québec, Kanada, 15. August 1998.

Wolfgang Menzel und Ingo Schröder. Error diagnosis for language learning systems. *ReCALL*, 11:20–30, Mai 1999. Natural Language Processing in Computer-Assisted Language Learning.

Alessio Miaschi und Felice Dell'Orletta. Contextual and non-contextual word embeddings: an in-depth linguistic investigation. In: *Proceedings of the 5th Workshop on Representation Learning for NLP (RepL4NLP 2020)*, S. 110–119, Online, 9. July 2020.

Mehryar Mohri. Finite-state transducers in language and speech processing. *Computational Linguistics*, 23(2):269–311, 1997.

Mehryar Mohri. Generic epsilon-removal algorithm for weighted automata. In: Sheng Yu und Andrei Păun (Hgg.): *Implementation and Application of Automata, 5th International Conference, CIAA 2000 London, Ontario, Canada, July 24?25, 2000 Revised Papers*, Band 2088 von *Lecture Notes in Computer Science (LNCS)*, S. 230–242. Springer, Berlin/Heidelberg, 2001.

Mehryar Mohri. Semiring frameworks and algorithms for shortest-distance problems. *Languages and Combinatorics*, 7(3):321–350, 2002.

Mehryar Mohri. Edit-distance of weighted automata: general definitions and algorithms. *International Journal of Foundations of Computer Science*, 14 (6):957–982, 2003.

Mehryar Mohri. Weighted automata algorithms. In: Manfred Droste, Werner Kuich und Heiko Vogler (Hgg.): *Handbook of weighted automata*, Monographs in Theoretical Computer Science, S. 213–254. Springer, Berlin/Heidelberg, 2009.

Mehryar Mohri und Richard Sproat. On a common fallacy in computational linguistics. In: Mickael Suominen, Antti Arppe, Anu Airola, Orvokki Heinämäki, Matti Miestamo, Urho Määttä, Jussi Niemi, Kari K. Pitkänen und Kaius Sinnemäki (Hgg.): *A Man of Measure: Festschrift in Honour of Fred Karlsson on his 60th Birthday*, Band 19 von *Special Supplement to SKY Journal of Linguistics*, S. 432–439. Turku: The Linguistic Association of Finland, 2006.

Mehryar Mohri, Fernando C. N. Pereira und Michael Riley. Weighted finite-state transducers in speech recognition. *Computer Speech and Language*, 16(1):69–88, 2002.

Mehryar Mohri, Fernando C. N. Pereira und Michael Riley. Speech recognition with weighted finite-state transducers. In: *Handbook of Speech Processing*, S. 559–584. Springer, Berlin/Heidelberg, 2008.

Daniel Naber. A rule-based style and grammar checker. Diplomarbeit, Universität Bielefeld, Technische Fakultät, 2003.

Alexis Nasr und Alexandra Volanschi. Integrating a POS tagger and a chunker implemented as weighted finite state machines. In: *Proceedings of the 5th International Workshop on Finite-State Methods and Natural Language Processing (FSMNLP 2005)*, Helsinki, Finland, 1.–2. September 2005.

Kemal Oflazer. Dependency parsing with an extended finite-state approach. *Computational Linguistics*, 29(4):515–544, 2003.

Katrin Ortmann, Adam Roussel und Stefanie Dipper. Evaluating off-the-shelf nlp tools for german. In: *Proceedings of the 15th Conference on Natural*

Language Processing (KONVENS 2019), S. 212–222, Erlangen, Deutschland, 9.–11. Oktober 2019. German Society for Computational Linguistics & Language Technology.

Daniel W. Otter, Julian R. Medina und Jugal K. Kalita. A survey of the usages of deep learning for natural language processing. *IEEE Transactions on Neural Networks and Learning Systems*, 32(2):604–624, 2021.

Gilles Pesant. A regular language membership constraint for finite sequences of variables. In: Mark Wallace (Hg.): *Principles and Practice of Constraint Programming – CP 2004, 10th International Conference, CP 2004, Toronto, Canada, September 27 – October 1, 2004. Proceedings*, Band 3258 von *Lecture Notes in Computer Science (LNCS)*, S. 482–495. Springer, Berlin/Heidelberg, 2004.

Jussi Piitulainen. Locally tree-shaped sentence automata and resolution of ambiguity. In: Kimmo Koskenniemi (Hg.): *Proceedings of the 10th Nordic Conference of Computational Linguistics (NODALIDA 1995)*, S. 50–58, Helsinki, Finland, 29.–30. Mai 1995.

Barbara Plank und Gertjan van Noord. Grammar-driven versus data-driven: Which parsing system is more affected by domain shifts? In: *Proceedings of the 2010 Workshop on NLP and Linguistics: Finding the Common Ground*, S. 25–33, Uppsala, Sweden, July 2010.

Martin Popel, David Mareček, Jan Štěpánek, Daniel Zeman und Zdeněk Žabokrtský. Coordination structures in dependency treebanks. In: *Proceedings of 51th Annual Meeting of the Association for Computational Linguistics (ACL 2013)*, S. 517–527, Sofia, Bulgarien, 4.–9. August 2013.

Martin Prinzhorn und Viola Schmitt. Discontinuous DP-coordination in german. In: Jeroen Van Craenenbroeck und Johan Rooryck (Hgg.): *Linguistic Variation Yearbook 2010*, Band 10, S. 161–200. John Benjamins, Amsterdam, 2011.

Michael Oser Rabin und Dana Stewart Scott. Finite automata and their decision problem. *IBM Journal of Research and Development*, 3:114–125, 1959.

Pushpendre Rastogi, Ryan Cotterell und Jason Eisner. Weighting finite-state transductions with neural context. In: *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics – Human Language Technologies (NAACL-HLT 2016)*, S. 623–633, San Diego, Kalifornien, USA, 12.–17. Juni 2016.

Guillaume Richaud, Hadrien Cambazard, Barry O'Sullivan und Narendra Jussien. Automata for nogood recording in constraint satisfaction problems. In: Youssef Hamadi und Lucas Bordeaux (Hgg.): *Proceedings of the first International CP 2006 Workshop on the Integration of SAT and CP techniques*, S. 113–127, Nantes, Frankreich, 25. September 2006.

Emmanuel Roche. Parsing with finite-state transducers. In: Emmanuel Roche und Yves Schabes (Hgg.): *Finite-State Language Processing*, S. 241–281. MIT Press, Cambridge, Massachusetts, USA, 1997.

Emmanuel Roche und Yves Schabes. Introduction to finite-state devices in natural language processing. In: Emmanuel Roche und Yves Schabes (Hgg.): *Finite State Natural Language Processing*, S. 1–66. MIT Press, Cambridge, Massachusetts, USA, 1997.

Francesca Rossi, Kristen Brent Venable und Toby Walsh. Preferences in constraint satisfaction and optimization. *AI Magazine*, 29(4):58–68, 2008.

Thomas Schiex, Hélène Fargier und Gerard Verfaillie. Valued constraint satisfaction problems: Hard and easy problems. In: *Proceedings of the 14th International Joint Conference on Artificial Intelligence (IJCAI 1995)*, Band 1, S. 631–637, Montreal, Québec, Kanada, 1995. Morgan Kaufman.

Anne Schiller, Simone Teufel und Christine Thielen. *Guidelines für das Tagging deutscher Textkorpora mit STTS*, 1995. Verfügbar unter `http://www.sfs.uni-tuebingen.de/resources/stts-1999.pdf`, Abrufdatum: 03.03.2022.

Ingo Schröder. *Natural Language Parsing with Graded Constraints*. Dissertation, University of Hamburg, Department of Computer Science, 2002.

Marcel-Paul Schützenberger. On the definition of a family of automata. *Information and Control*, 4:245–270, 1961.

Wolfgang Seeker und Jonas Kuhn. Making ellipses explicit in dependency conversion for a german treebank. In: Nicoletta Calzolari, Khalid Choukri, Thierry Declerck, Mehmet Uğur Doğan, Bente Maegaard, Joseph Mariani, Asuncion Moreno, Jan Odijk und Stelios Piperidis (Hgg.): *Proceedings of the 8th International Conference on Language Resources and Evaluation (LREC 2012)*, S. 3132–3139, Istanbul, Türkei, 23.–25. Mai 2012. European Language Resources Association.

Wolfgang Seeker und Daniel Quernheim. Optimality theory and vector semirings. In: Jakub Piskorski, Bruce Watson und Anssi Mikael Yli-Jyrä (Hgg.): *Finite-State Methods and Natural Language Processing, Post-Proceedings of the 7th International Workshop FSMNLP 2008*, Band 191 von *Frontiers in Artificial Intelligence and Applications*, S. 134–77, Ispra, Italien, 11.–12. September 2008. IOS Press.

Rico Sennrich, Gerold Schneider, Martin Volk und Martin Warin. A new hybrid dependency parser for german. In: Richard Eckart de Castilho Christian Chiarcos und Manfred Stede (Hgg.): *Form to Meaning: Processing Texts Automatically. Proceedings of the Biennial GSCL Conference 2009*, S. 115–124. Gunter Narr, Tübingen, 2009.

Kyriakos Sgarbas, Nikos D. Fakotakis und George K. Kokkinakis. Incremental construction of compact acyclic NFAs. In: *Proceedings of the 39th Annual Meeting of the Association for Computational Linguistics (ACL 2001)*, S. 474–481, Toulouse, Frankreich, 6.–11. Juli 2001.

Stuart M. Shieber. *An Introduction to Unification-Based Approaches to Grammar*, Band 4 von *CSLI Lecture Notes*. CSLI Publications, Stanford, CA, 1986.

Miikka Silfverberg und Krister Lindén. Part-of-speech tagging using parallel weighted finite-state transducers. In: Hrafn Loftsson, Eiríkur Rögnvaldsson und Sigrún Helgadóttir (Hgg.): *Advances in Natural Language Processing – 7th International Conference on NLP, IceTAL 2010, Reykjavik, Iceland, August 16–18, 2010*, Band 6233 von *Lecture Notes in Computer Science (LNCS)*, S. 369–380. Springer, Berlin/Heidelberg, 2010.

Richard Sproat und Michael Riley. Compilation of weighted finite-state transducers from decision trees. In: *Proceedings of the 34th Annual Meeting on*

Association for Computational Linguistics (ACL 1996), S. 215–222, Santa Cruz, Kalifornien, USA, 1996.

Richard Sproat, Mahsa Yarmohammadi, Izhak Shafran und Brian Roark. Applications of lexicographic semirings to problems in speech and language processing. *Computational Linguistics*, 40(4):733–761, 2014.

Felix Stahlberg, Christopher Bryant und Bill Byrne. Neural grammatical error correction with finite state transducers. In: *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics – Human Language Technologies (NAACL-HLT 2019)*, Band 1, S. 4033–4039, Minneapolis, Minnesota, USA, 2.–7. Juni 2019.

Pasi Tapanainen. Applying a finite-state intersection grammar. In: Emmanuel Roche und Yves Schabes (Hgg.): *Finite-State language processing*, S. 311–327. MIT Press, Cambridge, Massachusetts, USA, 1997.

Pasi Tapanainen und Timo Järvinen. A non-projective dependency parser. In: *Proceedings of the 5th Conference on Applied Natural Language Processing (ANLC 1997)*, S. 64–71, Washington, D.C., USA, 31. März – 3. April 1997.

Zbigniew Tarapata. Selected multicriteria shortest path problems: An analysis of complexity, models and adaptation of standard algorithms. *International Journal of Applied Mathematics and Computer Science*, 17(2): 269–287, 2007.

Ken Thompson. Programming techniques: Regular expression search algorithm. *Communications of the ACM*, 11(6):419–422, 1968.

Evelyne Tzoukerman und Dragomir R. Radev. Use of weighted finite state transducers in part of speech tagging. In: Andras Kornai (Hg.): *Extended Finite State Models of Language*. Cambridge University Press, 1999.

Francesca Rossi Ugo Montanari. Constraint relaxation may be perfect. *Artificial Intelligence*, 48(2):143–170, 1991.

Willem Jan van Hoeve, Gilles Pesant und Luois-Martin Rousseau. On global warming (softening global constraints). In: *Proceedings of 6th International CP 2004 Workshop on Preferences and Soft constraints (Soft 2004)*, Toronto, Ontario, Kanada, 27. September 2004.

Harry Schultz Vandiver. Note on a simple type of algebra in which the cancellation law of addition does not hold. *Bulletin of the American Mathematical Society*, 40:914–920, 1934.

Nageshwara Rao Vempaty. Solving constraint satisfaction problems using finite state automata. In: *Proceedings of the 10th Conference on Artificial Intelligence (AAAI 1992)*, S. 453–458, San Jose, Kalifornien, USA, 12.–16 July 1992. AAAI Press / The MIT Press.

Yannick Versley. Parser evaluation across text types. In: *Proceedings of the 4th Workshop on Treebanks and Linguistic Theories (TLT 2005)*, Barcelona, Spanien, 9.–10. Dezember 2005.

Alexander Volokh. *Performance-Oriented Dependency Parsing*, Band 37 von *Saarbrücken Dissertations in Language Science and Technology*. Universitätsverlag des Saarlandes, 2013.

Atro Voutilainen und Pasi Tapanainen. Ambiguity resolution in a reductionistic parser. In: *Proceedings of the 6th conference on European chapter of the Association for Computational Linguistics (EACL 1993)*, S. 394–403, Utrecht, Niederlande, 21.–23. April 1993.

Ke Wu, Cyril Allauzen, Keith B. Hall, Michael Riley und Brian Roark. Encoding linear models as weighted finite-state transducers. In: *Proceedings of the 15th Annual Conference of the International Speech Communication Association (INTERSPEECH 2014)*, S. 1258–1262, Singapur, 14.–18. September 2014. ISCA.

Anssi Mikael Yli-Jyrä. Schematic finite-state intersection parsing. In: *Short Papers Presented at the 10th Nordic Conference of Computational Linguistics (NODALIDA 1995)*, Helsinki, Finland, 29.–30. Mai 1995.

Anssi Mikael Yli-Jyrä. Structural correspondence between finite-state intersection grammar and constraint satisfaction problem. In: Lauri Karttunen, Kimmo Koskenniemi und Gertjan van Noord (Hgg.): *Proceedings of the ESSLLI 2001 Workshop on Finite-State Methods and Natural Language Processing (FSMNLP 2001)*, S. 1–4, Helsinki, Finland, 20.–24. August 2001.

Anssi Mikael Yli-Jyrä. Multiplanarity – a model for dependency structures in treebanks. In: *Proceedings of the 2nd Workshop on Treebanks and Linguistic Theories (TLT 2003)*, Band 9 von *Mathematical Modelling in Physics, Engineering and Cognitive Sciences*, S. 189–200, Växjö, Sweden, 14.–15. November 2003. University Press.

Anssi Mikael Yli-Jyrä. Axiomatization of restricted non-projective dependency trees through finite-state constraints that analyse crossing bracketings. In: Geert-Jan M. Kruijff und Denys Duchier (Hgg.): *Proceedings of the COLING 2004 Workshop of Recent Advances in Dependency Grammar*, S. 33–40, Geneva, Schweiz, 28. August 2004.

Anssi Mikael Yli-Jyrä. Linguistic grammars with very low complexity. In: Antti Arppe, Lauri Carlson, Krister Lindén, Jussi Piitulainen, Mickael Suominen, Martti Vainio, Hanna Westerlund und Anssi Yli-Jyrä (Hgg.): *Inquiries into Words, Constraints and Contexts. Festschrift for Kimmo Koskenniemi on his 60th Birthday*, CSLI Studies in Computational Linguistics ONLINE, S. 172–183. CSLI Publications, Stanford, 2005a.

Anssi Mikael Yli-Jyrä. Approximating dependency grammars through intersection of regular languages. In: Michael Domaratzki, Alexander Okhotin, Kai Salomaa und Sheng Yu (Hgg.): *Implementation and Application of Automata, 9th International Conference, CIAA 2004, Kingston, Canada, July 22–24, 2004, Revised Selected Papers*, Band 3317 von *Lecture Notes in Computer Science (LNCS)*, S. 281–292. Springer, Berlin/Heidelberg, 2005b.

Anssi Mikael Yli-Jyrä. Applications of diamonded double negation. In: Thomas Hanneforth und Kay Michael Würzner (Hgg.): *Finite-State Methods and Natural language Processing: revised papers / 6th International Workshop on Finite-State Methods and Natural language Processing (FSMNLP 2007)*, S. 6–30, Potsdam, Deutschland, 2008. Potsdam University Press.

Anssi Mikael Yli-Jyrä. On dependency analysis via contractions and weighted FSTs. In: Diana Santos, Krister Lindén und Wanjiku Ng'ang'a (Hgg.): *Shall We Play the Festschrift Game? Essays on the Occasion of Lauri Carlson's 60th Birthday*, S. 133–158. Springer, Berlin/Heidelberg, 2012.

Anssi Yli-Jyrä, Jussi Piitulainen und Atro Voutilainen. Refining the design of a contracting finite-state dependency parser. In: *Proceedings of the 10th International Workshop on Finite-State Methods and Natural Language Processing (FSMNLP 2012)*, S. 108–115, Donostia-San Sebastián, Spanien, 23.–25. Juli 2012.

Hao Zhang, Richard Sproat, Axel H. Ng, Felix Stahlberg, Xiaochang Peng, Kyle Gorman und Brian Roark. Neural models of text normalization for speech applications. *Computational Linguistics*, 45(2):293–337, Juni 2019.

Gisela Zifonun, Ludger Hoffmann und Bruno Strecker. *Grammatik der deutschen Sprache*, Band 1 von *Schriften des Instituts für Deutsche Sprache*. de Gruyter, Berlin/New York, 1997.