Bachelorarbeit zum Thema:

Operatorpräzedenzsprachen und ihre Automaten

vorgelegt am: 1. März 2019

Westfälische Wilhelmsuniversität Münster

Institut für Informatik

Name:	Jonas Kremer
Matrikelnummer:	430666
Email:	j_krem04@uni-muenster.de
Studiengang:	Bsc. Informatik
Fachsemester:	7 (WS. 18/19)
Arbeitsgruppe:	Softwareentwicklung und Verifikation
Erstgutachter:	Prof. Dr. Markus Müller-Olm
Zweitgutachter:	Jens Gutsfeld

Inhaltsverzeichnis

1 Motivation

In dieser Arbeit geht es um die Operatorpräzedenzsprachen (OPL) und um die Eigenschaften ihrer Grammatiken und Automaten. Diese Sprachenklasse wurde 1963 von R.W.Floyd eingeführt, weswegen sie auch häufig als Floydsprachen oder Floydgrammatiken betitelt wird. Floyd interessierte die Struktur von Ausdrücken (sowhl einfache arithmetische als auch programmiertechnische) und die Präzedenz von manchen Operatoren über andere. Besonders interessant war dabei die Präzedenz der Multiplikation über die Addition, die per Konvention gilt und somit nicht explizit durch Klammern klargemacht werden muss. Er definiert dafür drei Präzedenzrelationen $(\lessdot, \gtrdot, \doteq)$, die zwischen den Terminalsymbolen gelten und in einer Operatorpräzedenzmatrix (OPM) festgehalten werden.

s Als motivierendes Beispiel, was auch in den Definitionen weiterhin genutzt wird dient ein einfacher arithmetischer Ausdruck mit $\Sigma = \{n, +, \times, (,)\}$:
$n \times (n + n \times n)$, bei dem die implizite Präzedenz deutlich wird.

Diese Grammatiken verloren allerdings etwas an Interesse mit der Einführung der LR(k) Parser, was sich erst mit der Entwicklung eines geeigneten äquivalenten Automaten durch Violetta Lonati, Dino Mandrioli und Matteo Pradella im Jahre 2010 änderte. Diese Automaten erkennen exakt die Sprachen, die von den Grammatiken generiert werden und umgekehrt. Die Äquivalenz wird in beide Richtungen gezeigt. Generell bieten die OPLs viele interessante Eigenschaften: Sie sind eine echte Teilmenge der kontextfreien Sprachen, genießen aber trotzdem alle typischen Abschlusseigenschaften von regulären Sprachen in Bezug auf Boolische Operatoren, Konkatenation und Kleene-*. Weiterhin kann gezeigt werden, dass die Klasse der Visibly-Pushdown Sprachen und Automaten ebenfalls von OPGs und OPAs erkannt werden. Im Gegensatz zu anderen Parsern wie zB. dem LR(1)- Parser muss hier nicht zwangsläufig von links nach rechts gearbeitet werden, was eine Möglichkeit zur Parallelisierung bietet. Ferner bieten OPLs auch eine Erweiterung zur $\omega - OPL$ und eine *Monadic Second Order Logic Characterization*, was aber den Rahmen dieser Arbeit sprengen würde. Als praktischer letzter Teil folgt dann eine Implementierung der OPAs in geeigneter Form.

2 Grundlagen und Definitionen

In diesem Abschnitt geht es um die Definitionen und Namenskonventionen, die für Operatorpräzedenzsprachen benötigt werden. Zunächst kommt eine kurze Definition von kontextfreien Grammatiken, gefolgt von Definitionen und Einschränkungen für Operatorpräzedenzgrammatiken (OPG). Im Anschluss wird die Klasse der Operatorpräzedenzautomaten (OPA) eingeführt, sowohl deterministisch als auch nichtdeterministisch, sowie deren Äquivalenz zu OPGs bewiesen. Die Definitionen richten sich nach [1] [2] und [3].

2.1 Operatorpräzedenzgrammatik

Eine kontextfreie Grammatik (kfG) ist ein 4-tupel $G = (N, \Sigma, P, S)$, wobei N die Menge der Nichtterminalsymbole, Σ die Menge der Terminalsymbole, P die Menge der Produktionsregeln und S das Startsymbol bezeichnet. Folgende Namenskonventionen werden im weiteren Verlauf verwendet: Kleine lateinische Buchstaben am Anfang des Alphabets $a, b, ...$ bezeichnen einzelne Terminalsymbole; Spätere, kleine lateinische Buchstaben $u, v, ...$ bezeichnen Terminalstrings; Große lateinische Buchstaben $A, B, ...$ stehen für Nichtterminalsymbole und griechische Buchstaben $\alpha, \beta, ...$ für beliebige Strings über $N \cup \Sigma$. Sofern nicht explizit eingeschränkt können Strings auch leer sein.
Weiterhin haben Produktionsregeln die Form $A \to \alpha$ mit der *leeren Regel* $A \to \epsilon$. *Umbenennende* Regeln haben nur ein Nichtterminalsymbol als rechte Seite . Eine *direkte Ableitung* wird mit $\Rightarrow$ beschrieben, eine beliebige Anzahl von Ableitungen mit $\overset{*}{\Rightarrow}$.
Eine Grammatik heisst *reduziert*, wenn jede Regel aus P benutzt werden kann um einen String aus Σ^* zu erzeugen. Sie ist *invertierbar*, wenn keine zwei Regeln identische rechten Seiten haben.
Eine Regel ist in *Operatorform*, wenn ihre rechte Seite keine benachbarten Nichtterminale hat. Entsprechend heisst eine Grammatik, die nur solche Regeln beinhaltet *Operatorgrammatik* (OG). Jede kfG $G = (N, \Sigma, P, S)$ kann in eine äquivalente Operatorgrammatik $G' = (N', \Sigma, P', S)$ umgewandelt werden [9, 10].Die folgenden Definitionen gelten für Operatorpräzedenzgrammatiken (OPGs).

Definition 2.1. *Für eine OG G und ein Nichtterminal A sind die linken und rechten Terminalmengen definiert als*

$$\mathcal{L}_G(A) = \{a \in \Sigma | A \overset{*}{\Rightarrow} Ba\alpha\},\ \mathcal{R}_G(A) = \{a \in \Sigma | A \overset{*}{\Rightarrow} \alpha aB\}$$

mit $B \in N \cup \{\epsilon\}$

Auf den Namen der Grammatik G kann verzichtet werden, wenn der Kontext klar ist. Eines der wichtigsten Merkmale von Operatorpräzedenzgrammatiken ist die Definition von drei binären Operatorpräzedenzrelationen.

Definition 2.2 (Präzedenzrelationen)**.**
Gleiche Präzedenz: $a \doteq b \Leftrightarrow \exists A \to \alpha a B b \beta, B \in N \cup \{\epsilon\}$
Übernimmt Präzedenz: $a \gtrdot b \Leftrightarrow \exists A \to \alpha D b \beta, D \in N$ *and* $a \in \mathcal{R}_G(D)$
Gibt Präzedenz ab: $a \lessdot b \Leftrightarrow \exists A \to \alpha a D \beta, D \in N$ *and* $b \in \mathcal{L}_G(D)$

Es muss beachtet werden, dass diese Relationen im Gegensatz zu ähnlichen arithmetischen Relationen (<,>,=) keine transitiven, symmetrischen oder reflexiven Eigenschaften vorliegen. Weiterhin schließt die Gültigkeit einer Relation die andere nicht aus, sodass zB. sowohl $a \lessdot b$ als auch $a \doteq b$ gelten kann. Für eine OG G kann eine Operatorpräzedenzmatrix (OPM) $M = OPM(G)$ als $|\Sigma| \times |\Sigma|$ erstell werden, die für jedes geordnete Paar (a,b) die Menge M_{ab} der Operatorpräzedenzrelationen beinhaltet. Für solche Matrizen sind Inklusion und Vereinigung natürlich definiert.

Definition 2.3. *Eine OG G ist eine OPG oder auch FG gdw. M = OPM(G) eine konfliktfreie Matrix ist.*
Also: $\forall a, b, |M_{ab}| \leq 1$
Eine OPL ist eine Sprache, die durch eine OPG gebildet wird.

Für solche OPMs werden nun weitere Namenskonventionen vereinbart. Zwei Matrizen sind *kompatibel*, wenn ihre Vereinigung konfliktfrei ist. Eine Matrix heisst *total*, wenn gilt $\forall a, b : M_{\text{ab}} \neq \emptyset$. Für eine OPG wird die *Fischer Normalform (FNF)* definiert.

Definition 2.4 (Fischernormalform)**.** *Eine OPG G ist in Fischer Normalform gdw. gilt:*

- *G ist invertierbar*
- *G hat keine leeren Regeln außer dem Startsymbol, sofern dies nicht weiter verwendet wird*
- *G hat keine umbenennenden Regeln*

Zu jeder OPG kann eine äquivalente Grammatik in FNF gebildet werden. [9] In Ausblick auf die Operatorpräzedenzautomaten erweitern wir die OPM um ein Symbol $\# \notin \Sigma$, welches Start und Ende eines Strings bezeichnet. Dieses Symbol beeinflusst die Grammatik nicht weiter, da für jedes Terminal gilt: $\forall a \in \Sigma : \# \lessdot a$ und $a \gtrdot \#$. Ferner gilt: $M_{\# \#} = \{\doteq\}$.

Definition 2.5. *Ein OP Alphabet ist ein Paar* (Σ, M) *mit:*
Σ *ist ein Alphabet*
M *ist eine konfliktfreie OPM, erweitert zu einem* $|\Sigma \cup \{\#\}|^2$ *Array, das zu jedem geordneten Paar (a,b) höchstens eine OP Relation enthält.*

Zum Schluss sollte noch eine Einschränkung bezüglich der $\doteq$-Relation getroffen werden. Aus 2.2 folgt, dass bei einer Regel $A \to A_1 a_1 ... A_n a_n A_{n+1}$, bei der alle A_i potenziell fehlen können, die Relationen $a_1 \doteq a_1 \doteq ... \doteq a_n$ gebildet werden. Problematisch wird dies, wenn die $\doteq$-Relation *zyklisch* ist dh. es gibt $a_1, a_2, ..., a_m \in \Sigma (m \geq 1)$, sodass $a_1 \doteq a_2 \doteq ... \doteq a_m \doteq a_1$. Das führt dazu, dass die Länge der rechten Seite einer Regel keine Begrenzung hat. Wie in meinen Quellen gehe ich auch davon aus, dass die $\doteq$-Relation azyklisch ist. Dies kann einen Einfluss auf die Konstruktion mancher Grammatiken haben, allerdings betrifft es keine der hier verwendeten Grammatiken.
Nachdem hier die Grundlagen für die OPGs geschaffen wurden geht es weiter mit den Operatorpräzedenzautomaten.

2.2 Operatorpräzedenzautomat

Die Operatorpräzedenzautomaten(OPA) verhalten sich ähnlich wie Pushdown-Automaten, aber erkennen genau die Operatorpräzedenzsprachen. OPAs arbeiten Bottom-Up, sind aber deutlich einfacher als zB. LR(k) Parser. Die folgende Definition [2, 6] unterscheidet sich etwas von der originalen von [1], was sie etwas anschaulicher und einfacher macht.

Definition 2.6 (Operatorpräzedenzautomat)**.** *Ein nichtdeterministischer OPA ist ein 6-Tupel* $A = (\Sigma, M, Q, I, F, \delta)$ *mit*

- (Σ, M) *ist ein OP Alphabet*
- Q *ist eine Menge von Zuständen*
- $I \subseteq Q$ *ist die Menge der Startzustände*
- $F \subseteq Q$ *ist die Menge der akzeptierenden Zustände*
- $\delta : Q \times (\Sigma \cup Q) \to \mathcal{P}(Q)$ *ist die Übergangsfunktion, die aus drei Teilfunktionen besteht:*
  $\delta_{shift} : Q \times \Sigma \to \mathcal{P}(Q)$ $\delta_{push} : Q \times \Sigma \to \mathcal{P}(Q)$ $\delta_{pop} : Q \times Q \to \mathcal{P}(Q)$

Wie bei den meisten Automaten, bietet sich auch bei OPAs an eine graphische Darstellung einzuführen. Dabei werden die Zustände als Kreis mit

einer Namenskennzeichnung dargestellt. Im weiteren Verlauf der Arbeit werden die Zustände meistens mit q_i, q_i oder r_i bezeichnet.
Die drei Transitionen werden mit verschiedenen Arten von Pfeilen dargestellt: Ein normaler Pfeil von q nach p mit einem Terminal a als Beschriftung steht für eine Push-Transition $p \in \delta_{\mathrm{push}}(q, a)$. Ein gestrichelter Pfeil steht für eine Shift-Transition $p \in \delta_{\mathrm{shift}}(q, a)$. Ein doppelter Pfeil mit einem Zustand r als Beschriftung steht dementsprechend für einen Pop-Move $p \in \delta_{\mathrm{pop}}(q, r)$
Als nächstes wird der Stack des Automaten definiert. Für die Elemente des Stacks definieren wir $\Gamma = \Sigma \times Q$ und erweitern dies um das Symbol für den leeren Stack $\Gamma' = \Gamma \cup \{\bot\}$. Elemente von Γ' haben die Form $[a, q]$ oder $\bot$.
Für Γ' werden ein paar Hilfsfunktionen einegführt: $symbol([a, q]) = a$, $symbol(\bot) = \#$ und $state([a, q]) = q$. Der Stack ist ein String der Form $\Pi = \bot\pi_1\pi_2...pi_\mathrm{n}$ mit $\pi_\mathrm{i} \in \Gamma$. Die symbol-Funktion wird auf Strings erweitert, indem das Symbol des letzten Elements ausgegeben wird: $symbol(\Pi) = symbol(\pi_\mathrm{n})$.
Eine *Konfiguration* eines OPA ist ein Tripel $C = (\Pi, q, w)$, wobei $\Pi \in \bot\Gamma^*$ den aktuelle Stack, $q \in Q$ den Zustand und $w \in \Sigma^*\#$ das übrige Eingabewort bezeichnet.
Bei einem *Lauf* des Automaten handelt es sich um eine endliche Folge von Transitionen/Moves $C_1 \vdash C_2$.

Definition 2.7 (Übergangsfunktionen)**.** *Die drei verschiedenen Moves sind wie folgt definiert:*

push move: if $symbol(\Pi) \lessdot a$ *then* $(\Pi, p, ax) \vdash (\Pi\,[a, p]\,, q, x)$ *mit* $q \in \delta_{push}(p, a)$
push move: if $a \doteq b$ *then* $(\Pi\,[a, p]\,, q, bx) \vdash (\Pi\,[b, p]\,, r, x)$ *mit* $r \in \delta_{shift}(q, b)$
pop move: if $a \gtrdot b$ *then* $(\Pi\,[a, p]\,, q, bx) \vdash (\Pi, r, bx)$ *mit* $r \in \delta_{pop}(q, p)$

Einfach gesagt bedeutet das: Bei einem Push-Move wird der Stack aufgebaut, während das Eingabewort weiter verbraucht wird. Der Shift-Move verarbeitet ebenfalls das Eingabewort, tauscht allerdings nur das Symbol im vorderen Stackelement aus. Der Pop-Move arbeitet dann den Stack ab. Shift- und Pop-Moves werden bei einem leeren Stack nicht ausgeführt.
Die δ-Funktion besteht hier aus drei anstatt zwei Teilfunktionen, was gerade die Pop-Transition deutlich einfacher macht. Im Original markiert die Pushtransition zusätzlich die Terminale und die Flush-(Pop)funktion baut den Stack bis zur Markierung ab. Das führt zu einer komplizierteren Verion der letzten Teilfunktion, weshalb hier die Definition aus [2] verwendet wird.
Um den Automaten zu vervollständigen muss die akzeptierte Sprache definiert werden. Eine Konfiguration $(\bot, q_\mathrm{I}, w\#)$ mit $q_\mathrm{I} \in I$ heisst *initial* und eine Konfiguration $(\bot, q_\mathrm{F}, \#)$ mit $q_\mathrm{F} \in F$ heisst akzeptierend.

Definition 2.8 (Akzeptanz der Sprache)**.** *Die Sprache, die von einem OPA*

A erkannt wird ist definiert als
$L(A) = \left\{ w | (\bot, q_I, w\#) \overset{*}{\vdash} (\bot, q_F, \#), q_I \in I, q_F \in F \right\}$

Damit ist die Definition eines nichtdeterministischen OPAs vollständig.

Weitere Strukturen

Um das Verhalten von OPAs beschreiben zu können und die interessanteren Eigenschaften zu beweisen müssen ~~allerdings~~ noch ein paar weitere Grundlagen von OPAs benannt und definiert werden.

Definition 2.9 (Einfache Ketten)**.** *Sei (Σ, M) ein Präzedenz-Alphabet. Eine einfache Kette ist ein Wort $a_0a_1a_2...a_na_{n+1}$, welches wir als $^{a_0}\left[a_1a_2...a_n\right]^{a_{n+1}}$ notieren, mit $a_0, a_{n+1} \in \Sigma \cup \{\#\}, a_i \in \Sigma$ für $1 \leq i \leq n, M_{a_0\ a_{n+1}} \neq \emptyset$ und $a_0 \lessdot a_1 \doteq a_2...a_n \gtrdot a_{n+1}$*

Definition 2.10 (Zusammengesetzte Ketten)**.** *Sei (Σ, M) ein Präzedenz-Alphabet. Eine zusammengesetzte Kette ist ein Wort $a_0x_0a_1x_1a_2...a_nx_na_{n+1}$ mit $x_i \in \Sigma^*$, bei dem $^{a_0}\left[a_1a_2...a_n\right]^{a_{n+1}}$ eine einfache Kette ist und jedes x_i entweder leer(ϵ) oder wieder eine (einfache oder zusammengesetzte) Kette ist.*
Die Schreibweise ist $^{a_0}\left[x_0a_1x_1a_2...a_nx_n\right]^{a_{n+1}}$

Für diese Strukturen benennt man weitere Eigenschaften. Bei einer Kette $^{\mathrm{a}}\left[x\right]^{\mathrm{b}}$ wird x der *Rumpf* genannt. Die *Tiefe* d(x) einer Kette wird rekursiv definiert: $d(x) = 1$, wenn x eine einfache Kette ist und $d(x_0a_1x_1...a_{\mathrm{n}}x_{\mathrm{n}}) = \max_i(d(x_{\mathrm{i}})) + 1$ bei zusammengesetzten Ketten. Die Struktur der Ketten ist gleich der internen Struktur von Eingabewörtern und so kann die Tiefe einfach abgelesen werden, wenn ein Strukturbaum gebildet wird. Bei einer manuellen Konstruktion eines Strukturbaumes kann Bottom-Up vorgegangen werden: Zunächst identifiziert man die einfachen Ketten und schreibt diese als Blätter auf. Jetzt geht man in mehreren Iterationen das Eingabewort durch identifiziert zusammengesetzte Ketten. Hilfreich ist dabei die Ketten mit Variablen abzukürzen. Das Eingabewort wird immer weiter durch Ketten ausgedünnt bis die finale Kette mit dem #-Symbol stehen bleibt.
Mit diesen Ketten lässt sich jetzt die Kompatibilät eines Wortes w mit einer OPM M definieren, die allerdings nicht mit der Akzeptanz für einen OPA verwechselt werden darf.

Definition 2.11 (Kompatibilität mit OPM)**.** *Ein Wort w über (Σ, M) ist kompatibel mit M, wenn die beiden Bedingungen gelten:*

- *Für jedes aufeinanderfolgende Paar von Nichtterminalen b,c in w muss* $M_{bc} \neq \emptyset$ *gelten*
- *Für jeden Substring x von #w# mit* $x = a_0 x_0 a_1 x_1 a_2 ... a_n x_n a_n$ *gilt: Wenn* $a_0 \lessdot a_1 \doteq a_2 ... a_{n\text{-}1} \doteq a_n \gtrdot a_{n+1}$ *und für alle* $0 \leq i \leq n$ *ist* x_i *entweder* ϵ *oder* ${}^{a_i}[x]^{a_{n+1}}$ *ist eine Kette, dann ist* $M_{a_0 a_{n+1}} \neq \emptyset$

Das Konzept für einfache und zusammengesetzte Ketten wird nun in Form von *Stützen* auf OPAs übertragen. Dabei werden die gleichen Konventionen für Pfeile verwendet wie oben beschrieben.

Definition 2.12 (Stützen)**.** *Sei A ein OPA. Eine Stütze für eine einfache Kette* ${}^{a_0}[a_1 a_2 ... a_n]^{a_{n+1}}$ *ist ein beliebiger Pfad in A der Form*

$$q_0 \overset{a_1}{\rightarrow} q_1 \dashrightarrow ... \dashrightarrow q_{n-1} \overset{a_n}{\dashrightarrow} q_n \overset{q_0}{\Rightarrow} q_{n+1} \tag{1}$$

Eine Stütze für eine zusammengesetzte Kette ${}^{a_0}[x_0 a_1 x_1 a_2 ... a_n x_n]^{a_{n+1}}$ *ist ein beliebiger Pfad in A der Form*

$$q_0 \overset{x_0}{\rightsquigarrow} q'_0 \overset{a_1}{\rightarrow} q_1 \overset{x_1}{\rightsquigarrow} q'_1 \dashrightarrow ... \dashrightarrow q_{n-1} \overset{a_n}{\dashrightarrow} q_n \overset{x_n}{\rightsquigarrow} q'_n \overset{q'_0}{\Rightarrow} q_{n+1} \tag{2}$$

wobei für jedes $0 \leq i \leq n$ *gilt:*

- *if* $x_i \neq \epsilon$*, dann ist* $q_i \overset{x_i}{\rightsquigarrow} q'_i$ *eine Stütze für die Kette* ${}^{a_i}[x_i]^{a_{i+1}}$
- *if* $x_i = \epsilon$*, dann ist* $q'_i = q_i$

Wichtig zu beachten ist, dass der einzige Pop-Move am Ende mit genau dem ersten Zustand der Kette q_0 beschriftet ist, bzw. bei zusammengesetzten Ketten q'_0. Der Automat nutzt die Informationen der Ketten um die Stützen zu bilden. Durch die Stützen wird der Lauf eines Eingabewortes eindeutig festgelegt. Für jedes Wort $w \in L(A)$ existiert eine Stütze $q_I \overset{w}{\rightsquigarrow} q_F$ mit $q_I \in I, q_F \in F$. Dabei entspricht die Tiefe d der Kette ${}^{\#}[x_w]^{\#}$, die als äußerste Kette zu verstehen ist der maximalen Stackgröße, die während eines Laufes erreicht wird.
Damit sind die wesentlichen Definitionen für das Verhalten der Operatorpräzedenzautomaten erstmal abgeschlossen.

3 Eigenschaften von OPLs

In diesem Abschnitt geht es um die wichtigsten Eigenschaften von Operatorpräzedenzautomaten, wie die Äquivalenz zwischen OPG und OPA, der Äquivalenz von deterministischen und nichtdeterministischen OPAs und den Abschlusseigenschaften, sowie eine Einordnung von anderen Automatenklassen wie endliche Automaten und Visibly-Pushdown Automaten. Weiterhin werden zu allen Eigenschaften die Beweise geführt.

3.1 Äquivalenz von OPG und OPA

Eine wesentliche Eigenschaft vieler Sprachklassen ist die Äquivalenz ihrer Grammatiken mit den Automaten. Mit dem vorgestellten OPA haben die OPL nun einen Automaten, der genau die Grammatik erkennt. Die Äquivalenz wird in beiden Richtungen gezeigt. [2, 1]

3.1.1 Von OPGs zu OPAs

Lemma 3.1. *Sei $G = (N, \Sigma, P, S)$ eine OPG. Dann kann ein OPA A konstruiert werden, sodass $L(G) = L(A)$. Weiterhin sei m die Summer der rechten Seite von Produktionsregeln in G. Dann hat A genau $O(m^2)$ Zustände.*

Beweis. Zunächst wird der nichtdeterministische OPA $A = (\Sigma, M, Q, I, F, \delta)$ aus einer gegebenen Grammatik mit derselben OPM konstruiert. Ohne Beschränkung der Allgemeinheit wird angenommen, dass die Grammatik G keine leeren oder umbenennenden Regeln hat. Der OPA A wird nun so konstruiert, dass eine akzeptierende Berechnung dem Aufbauen eines Bottom-Up Ableitungsbaums in G gleicht. Der Automat führt eine Pushtransition aus, wenn das erste Terminal einer neuen rechten Regelseite gelesen wird. Eine Shift-transition wird ausgeführt bei Terminalsymbolen innerhalb der rechten Seite einer Regel und schätzt nichtdeterministisch das Nichtterminal der linken Seite. Jeder Zustand enthält dabei zwei Informationen: Die erste Komponente repräsentiert dabei das Präfix der rechten Seite unter Konstruktion, während die zweite genutzt wird um die rechte Seite, die vorher unter Konstruktion war, wiederherzustellen, wenn alle verschachtelten rechten Seiten fertiggestellt wurden. Genau definiert wird die Konstruktion folgendermaßen: Sei

$$\mathbb{P} = \alpha \in (N \cup \Sigma) * \Sigma | \exists a \to \alpha\beta \in P$$

die Menge der Präfixe (die auf ein Terminalsymbol enden) der rechten Regelseiten in G. Weiter sei

$$\mathbb{Q} = \{\epsilon\} \cup \mathbb{P} \cup N,$$
$$Q = \mathbb{Q} \times (\{\epsilon\} \cup \mathbb{P}),\ I = (\epsilon, \epsilon) \text{ und } F = S \times \{\epsilon\} \cup \{(\epsilon, \epsilon | \epsilon \in L(G)\}$$

Anmerkung: $|\mathbb{Q}| = 1 + |\mathbb{P}| + |N|$ ist $O(m)$, also hat Q $O(m^2)$ Zustände Die Transitionen sind folgendermaßen definiert für $a \in \Sigma, \quad \alpha, \alpha_1, \alpha_2 \in \mathbb{Q}$ und $\beta, \beta_1, \beta_2 \in \mathbb{P} \cup \{\epsilon\}$:

- $\delta_{push}\big((\alpha, \beta), a\big) \ni \begin{cases} (a, \alpha) & if; \alpha \notin N, \\ (\alpha a, \beta) & if; \alpha \in N, \end{cases}$
- $\delta_{shift}\big((\alpha, \beta), a\big) \ni \begin{cases} (\alpha a, \beta) & if; \alpha \notin N, \\ (\beta \alpha a, \beta) & if; \alpha \in N, \end{cases}$
- $\delta_{pop}\big((\alpha_1, \beta_1), (\alpha_2, \beta_2)\big) \ni (A, \gamma)$, für jedes A, sodass $\begin{cases} A \to \alpha_1 \in P & if; \alpha_1 \notin N, \\ A \to \beta_1 \alpha_1 \in P & if; \alpha_1 \in N, \end{cases}$ und $\gamma = \begin{cases} \alpha_2 & if; \alpha_2 \notin N, \\ \beta_2 & if; \beta_2 \in N. \end{cases}$

Die entstandenen Zustände von δ_{shift} und δ_{shift} sind eindeutig, während δ : *pop* mehrere Zustände produzieren könnte, abhängig von wiederholten rechten Regelseiten. Die Zustände, die von push und shift Transisitionen erreicht werden haben ihre erste Komponente in $\mathbb{P}$. Wenn der Zustand (α, β) nach einer push Transition erreicht wird, dann ist α das Präfix der rechten Regelseite, die gerade in Konstruktion ist und β das Präfix, was vorher in Konstruktion war. In diesem Fall ist α entweder ein Terminalsymbol oder ein Nichtterminal gefolgt von einem Terminal.

Wenn der Zustand (α, β) nach einer Shift-Transition erreicht wird, dann ist α die Konkatenation der ersten Komponente des vorherigen Zustands und dem gelesenen Terminal und β ändert sich nicht vom vorigen Zustand.

Die Zustände, die von Pop-Transitionen erreicht werden, haben die erste Komponente in N: if (A, γ) so ein Zustand ist, dann ist A die linke Regelseite und γ das Präfix, was vorher unter Konstruktion war.

Soviel zur Konstruktion des Automaten. Die Äquivalenz zwischen G und A leitet sich nun aus den folgenden beiden Lemmas ab.

Lemma 3.2. *Sei x der Rumpf einer Kette und $\beta, \gamma \in \mathbb{P} \cup \{\epsilon\}$. Dann gilt für alle $h \geq 1$: Aus $(\beta, \gamma) \overset{x}{\rightsquigarrow} q$ folgt die Existenz von $A \in N$, sodass $A \overset{*}{\Rightarrow} x$ in G und $q = (A, \beta)$*

Beweis. Beweis durch Induktion der Tiefe $h = d(x)$.

Induktionsanfang: (h=1) Wenn h=1 ist, dann ist $x = a_1a_2...a_n$ der Rumpf einer einfachen Kette mit einer Stütze wie in 2.12 (1) mit $q_0 = (\beta, \gamma)$ und $q_n{+}1 = q$. Weiter folgt aus der Definition der Push- und Shiftfunktionen, dass $q_i = (a_1...a_i, \beta)$ für alle $1 \leq i \leq n$ und aufgrund der Definition der Popfunktion ($\beta \notin N$, aufgrund der Induktionssannahme) ist $q = (A, \beta)$, mit $A \to a_1...a_n = x$ in P. Die so konstruierte Stütze hat die Form

$$(\beta, \gamma) \overset{a_1}{\to} (a_1, \beta) \dashrightarrow ... \dashrightarrow (a_1...a_{n-1}, \beta) \overset{a_n}{\dashrightarrow} (a_1...a_n, \beta) \overset{(\beta,\gamma)}{\Rightarrow} (A, \beta).$$

Somit ist $A \overset{*}{\Rightarrow} x$ in G und der Induktionsanfang ist abgeschlossen.

Induktionsschritt Für h>1 ist $x = x_0a_1x_1...a_nx_n$ der Rumpf einer zusammengesetzten Kette mit einer Stütze wie in 2.12 (2) mit $q_0 = (\beta, \gamma)$ und $q_{n+1} = q$. Weiterhin sei $q_i = (\beta, \gamma)$ für alle $0 \neq i \neq n$ (Genauer ist $\beta_0 = \beta$ und $\gamma_0 = \gamma$). Aus der Induktionssannahme folgt nun für jeden nichtleere Rumpf x_i einer Kette mit einer geringeren Tiefe als h, existiert $X_i \in N$, sodass es $X_i \overset{*}{\Rightarrow} x_i$ in G gibt und $q_i = (X_i, \beta)$ ist. Somit kann die Stütze wie folgt konstruiert werden:

$$(\beta, \gamma) \overset{x_0}{\rightsquigarrow} q_0' \overset{a_1}{\to} (\beta_1, \gamma_1) \overset{x_1}{\rightsquigarrow} q_1' \dashrightarrow ... \dashrightarrow q_{n-1} \overset{a_n}{\dashrightarrow} (\beta_n, \gamma_n) \overset{x_n}{\rightsquigarrow} q_n' \overset{q_0'}{\Rightarrow} q$$

mit

$$q_i' = \begin{cases} (\beta_i, \gamma_i) & if\ x_i = \epsilon \\ (X_i, \beta_i) & sonst \end{cases}$$

Nun wird aufgrund der Definition der Push- und Shiftfunktionen klar, dass für alle $i \neq 0$, $\beta_i = X_0a_1...X_{i-1}a_i$ gilt, egal ob x_i leer ist oder nicht ($X_i = \epsilon$, wenn $x_i = \epsilon$). Um jetzt den Zustand q, der mit der finalen Pop-Transition $\delta_{pop}(q_n', q_0')$ erreicht wird zu berechnen, müssen vier verschiedene Fälle betrachtet werden (Sind x_0, x_n leer oder nicht?), was genau in der Definition von δ_{pop} abgebildet ist. In allen Fällen hat q aber die Form (A, β) mit $A \to X_0a_1X_1...X_na_nX_n$ in G.
Somit ist der Beweis abgeschlossen. □

Lemma 3.3. . *Sei x der Rumpf einer Kette und $A \in N$. Dann folgt aus $A \overset{*}{\Rightarrow} x$ in G, dass $(\beta, \gamma) \overset{x}{\rightsquigarrow} (A, \beta)$ für jedes $\beta, \gamma \in \mathbb{P} \cup \{\epsilon\}$.*

Beweis. Beweis durch Induktion auf die Tiefe h der Kette.

Induktionsanfang: (h=1) Wenn h=1 ist, dann ist x der Rumpf einer einfachen Kette und somit bedeutet $A \overset{*}{\Rightarrow} x$, dass $A \to x$ eine Produktion ist. Somit kann aufgrund der Definition von δ ($\beta \notin N$ aufgrund der Annahme) eine Stütze wie in 2.12 (1) mit $q_0 = (\beta, \gamma)$, $q_{n+1} = q$ und $q_i = (a_1...a_i, \beta)$ für alle $i = 1, 2, ...n$.

Induktionsschritt: Wenn h > 1 ist, dann ist x der Rumpf einer zusammengesetzten Kette mit $x = x_0a_1x_1...a_nx_n$. Weiter folgt aus $A \overset{*}{\Rightarrow} x$ in G die Existenz von $X_0, X_1, ..., X_n \in \{\epsilon\} \cup N$ $(X_i = \epsilon,\ if\ x_i = \epsilon)$, sodass $A \to X_0a_1X_1...a_nX_n$ und $X_i \overset{*}{\Rightarrow} x_i$. Der erste Schritt der Berechnung hängt davon ab, ob x_0 leer ist oder nicht. In beiden Fällen hat der Pfad jedoch die Form

$$(\beta, \gamma) \overset{x_0}{\rightsquigarrow} q_0' \overset{a_1}{\to} (X_0a_1, \beta), \quad mit \quad q_0' == \begin{cases} (\beta, \gamma) & if\ x_0 = \epsilon \\ (X_0, \beta) & sonst \end{cases}$$

Die Berechnung hängt auch weiter davon ab ob $x_1, x_2, ...x_{n-1}$ leer oder nicht sind. Jedoch gilt aufgrund der Induktionsannahme und der Definition von δ_{shift}, dass der Automat nach dem Lesen von a_i den Zustand $(X_0a_1...X_{i-1}a_i, \beta)$ für jedes $i = 1, ..., n$ erreicht mit dem Pfad

$$(\beta, \gamma) \overset{x_0}{\rightsquigarrow} q_0' \overset{a_1}{\to} (X_0a_1, \beta) \overset{x_1}{\rightsquigarrow} q_1 \overset{a_2}{\dashrightarrow} (X_0a_1X_1a_2, \beta) \overset{x_2}{\rightsquigarrow} q_2' \overset{a_3}{\dashrightarrow} ... \\ ... \overset{a_n}{\dashrightarrow} (X_0a_1...X_{n-1}a_n, \beta)$$

Wenn $x_n \neq \epsilon$ geht die Berechnung weiter mit dem letzten Schritt

$$(X_0a_1...X_{n-1}a_n, \beta) \overset{x_n}{\rightsquigarrow} (X_n, X_0a_1...X_{n-1}a_n).$$

Damit endet die Berechnung nun mit einer Pop-Transition. Die vier Fälle, die durch leere x_0, x_n entstehen sind durch die Defintion von δ_{pop} abgedeckt. In allen Fällen wurde aber eine Stütze für der auf den Zustand (A, β) endet konstruiert und der Beweis ist abgeschlossen. □

Somit wurde in beide Richtungen gezeigt, dass die Konstruktion des Automaten genau die Ketten der Grammatik repräsentiert und es für jede Stütze eine Ableitung in G gibt. Damit ist der Beweis abgeschlossen □

3.1.2 Von OPAs zu OPGs

Die Konstruktion einer äquivalenten Grammatik G aus einem OPA A ist deutlich einfacher als andersrum, was aus der Struktur, die durch OPMs vorgegeben ist resultiert.

Lemma 3.4. *Für einen OPA A kann eine äquivalente Grammatik G konstruiert werden, sodass L(G)=L(A) gilt.*

Beweis. Die Konstruktion setzt das Prinzip der Stützen einfach und direkt in die Grammatik um. Für einen gegebenen Automaten $A = (\Sigma, M, Q, I, F, \delta)$ wird nun die Grammatik $G = (\Sigma, N, P, S)$ konstruiert:

- Die Nichtterminale N sind 4-Tupel $(a, q, p, b) \in \Sigma \times Q \times Q \times \Sigma$ und werden als $({}^{b}p, q^{b})$ beschrieben.
  Die Regeln in P werden wie folgt konstruiert:

- Für jeden Support 2.12(1) einer einfachen Kette wird die Regel

$$({}^{a_0}q_0, {q_{n+1}}^{a_{n+1}}) \to a_1 a_2 ... a_n \tag{3}$$

  hinzugefügt. Wenn $a_0 = a_{n+1} = \#$, $q_0 \in I$ und $q_{n+1} \in F$, dann ist $({}^{\#}q_0, q_{n+1}^{\#})$ in S.

- Für jeden Support 2.12(2) einer zusammengesetzten Kette wird die Regel

$$({}^{a_0}q_0, {q_{n+1}}^{a_{n+1}}) \to \Delta_0 a_1 \Delta_1 a_2 ... a_n \Delta_n \tag{4}$$

  hinzugefügt, wobei für alle $0 \leq i \leq n$, $\Delta_i = ({}^{a}_{i}q_i, q_i'^{a_{i+1}})$, wenn x_i der Kette nicht leer ist, sonst $\Delta_i = \epsilon$. Wenn $a_0 = a_{n+1} = \#$, $q_0 \in I$ und $q_{n+1} \in F$, dann ist $({}^{\#}q_0, q_{n+1}^{\#})$ in S.

- Wenn der Automat ϵ akzeptiert, wird eine Regel $S \to \epsilon$ hinzugefügt. S darf nicht in anderen Regeln verwendet werden.

Durch die Definition der Zustände ist $|N|$ maximal $O(|\Sigma|^2 * |Q|^2)$. Für die Konstruktion wurden die Stützen direkt in Regeln übersetzt. Die Äquivalenz sollte daher offensichtlich sein.

□

3.2 Deterministischer vs. Nichtdeterminstischer OPA

Die vorherige Definition von OPAs war nichtdeterministisch. Eine vorteilhafte Eigenschaft von OPAs ist nun die Äquivalenz der deterministischen und nichtdeterministischen Version. Dadurch unterscheidet sich diese Familie der Automaten wesentlich von den Kellerautomaten.
In diesem Abschnitt geht es um die Definition und die Konstruktion eines solchen deterministischen OPAs und um die Lemmata, die nötig sind um die Korrektheit zu beweisen. [1]

Definition 3.1 (Deterministischer OPA)**.** *Ein OPA* $A = (\Sigma, M, Q, I, F, \delta)$ *ist deterministisch, wenn gilt:*

- *I besteht aus nur einem Element*
- $\delta : Q \times \{Q \cup \Sigma\} \to Q\}$
  (δ bildet auf Q ab, anstatt auf $\mathcal{P}(Q)$)

Anmerkung: Bei dem Startzustand wird nicht zwischen dem einzelnen Element und einer Singleton-Menge unterschieden.
Bei der Konstruktion eines deterministischen OPA aus einem nichtdeterministischen wird folgende Grundidee verwendet: Es muss sichergestellt werden, dass die Pop-Moves von verschiedene Läufen des nichtdeterministischen Automaten nicht mit falschen initialen und finalen Zuständen vermischt werden. Dazu werden Informationen zu dem Pfad, die der Automat seit dem Push-Move, also dem Beginn einer Kette, genommen hat, gesammelt. Dazu wird jeder Zustand des deterministischen Automaten als Menge von Zustandspaaren defininiert. Der deterministische Automat simuliert den nichtdeterministischen Lauf mithilfe des ersten Zustand des Paares und speichert den Zustand, von dem aus der Push-Move ausging im zweiten Zustand. Die deterministischen Pop-Moves werden anhand der nichtdeterministischen Pop-Moves die für den ersten Zustand und dem Zustand, der vor dem letzten Push-Move erreicht wurde (Was dem obersten Stackeintrag entspricht), definiert sind simuliert.
Das wird nun in einem Lemma formal ausgedrückt und bewiesen.

Lemma 3.5. *Aus einem nichtdeterminischen OPA A mit s Zuständen kann ein äquivalenter deterministischer OPA $\tilde{A}$ mit $2^{O(s^2)}$ Zuständen konstruiert werden.*

Beweis. Sei $A = (\Sigma, M, Q, I, F, \delta)$ ein nichtdeterministischer OPA. Der deterministsiche OPA $\tilde{A} = (\Sigma, M, \tilde{Q}, \tilde{I}, \tilde{F}, \tilde{\delta}$ wird wie folgt definiert:

- $\tilde{Q} = \mathcal{P}(Q \times (Q \cup \{\top\}))$, wobei $\top \notin Q$ für den Basis der Berechnungen (leerer Stack) steht. Zustände in $\tilde{Q}$ werden mit K bezeichnet
- $\tilde{I} = I \times \{\top\}$ ist der Startzustand
- $\tilde{F} = \{K | K \cap (F \times \{\top\}) \neq \emptyset\}$
- $\tilde{\delta} : \tilde{Q} \times (\Sigma \cup \tilde{Q}) \to \tilde{Q}$ ist die Übergangsfunktion ist die Vereinigung aus den drei Teilfunktionen:

$\tilde{\delta}_{push} : \tilde{Q} \times \Sigma \to \tilde{Q}$ ist definiert als:

$$\tilde{\delta}_{push}(K, a) = \bigcup_{(q,p)\in K} \{(h, q) | h \in \delta_{push}(q, a)\}$$

$\tilde{\delta}_{shift} : \tilde{Q} \times \Sigma \to \tilde{Q}$ ist definiert als:

$$\tilde{\delta}_{shift}(K, a) = \bigcup_{(q,p)\in K} \{(h, p) | h \in \delta_{shift}(q, a)\}$$

$\tilde{\delta}_{pop} : \tilde{Q} \times \tilde{Q} \to \tilde{Q}$ ist definiert als:

$$\tilde{\delta}_{pop}(K_1, K_2) = \bigcup_{(r,q)\in K_1,(q,p)\in K_2} \{(h, p) | h \in \delta_{pop}(r, q)\}$$

Die Zahl s der Zustände wächste exponential: Sei s = $|Q|$ die Anzahl der Zustände vom nichtdeterministischen Automaten A, dann ist $|\tilde{Q}| = 2^{|Q|*|Q\cup\{\top\}|}$. Also hat $\tilde{A}$ genau $2^{O(s^2)}$ Zustände.

Die Äquivalenz der beiden Automaten basiert auf den Aussagen der beiden folgenden Lemmata, die in jeweils Richtungen zeigen, dass äquivalente Supports gebildet werden:

Lemma 3.6. *Sei y der Rumpf einer Kette mit der Stütze* $q \overset{y}{\rightsquigarrow} q'$ *in A. Dann gilt für alle* $p \in Q$ *und* $K \in \tilde{Q}$*, wenn* $(q, p) \in K$*, dann existiert eine Stütze* $K \overset{y}{\rightsquigarrow} K'$ *mit* $(q', p) \in K'$

Beweis. Beweis durch Induktion der Tiefe h = $d(y)$.

Induktionsanfang: (h = 1)
Wenn h = 1 ist, dann ist $y = a_1a_2...a_n$ und die Stütze in A hat die Form $q \overset{a_1}{\to} q_1 \dashrightarrow ... \dashrightarrow q_{n-1} \overset{a_n}{\dashrightarrow} q_n \overset{q_0}{\Rightarrow} q'$.

Sei
$K_1 = \tilde{\delta}_{push}(K, a_1)$,
$K_i = \tilde{\delta}_{shift}(K_{i-1}, a_1)$, für jedes $i = 2, ..., n$
$K' = \tilde{\delta}_{pop}(K_n, K)$.

Dann ist
$$K \overset{a_1}{\to} K_1 \overset{a_2}{\dashrightarrow} ... \overset{a_{n-1}}{\dashrightarrow} K_{n-1} \overset{a_n}{\dashrightarrow} K_n \overset{q_0}{\Rightarrow} K'.$$
die Stütze in $\tilde{A}$. Weiterhin gilt für $(q, p) \in K$ aufgrund der Definition von $\tilde{\delta}$:
$(q_1, q) \in K_1$, da $q_1 \in \delta_{push}(q, a_1)$,
$(q_i, q) \in K_i$, da $q_i \in \delta_{shift}(q_{i-1})$,
$(q', p) \in K'$, da $q' \in \delta_{pop}(q_n, q)$

Induktionsschritt
Nun gelte Induktionsannahme für Stützen mit einer geringeren Tiefe als h. Weiter sei $y = x_0a_1x_1a_2...a_nx_n$ mit der Tiefe h und der Stütze $q \overset{x_0}{\rightsquigarrow} q'_0 \overset{a_1}{\rightarrow} q_1 \overset{x_1}{\rightsquigarrow} q'_1 \dashrightarrow ... \dashrightarrow q_{n-1} \overset{a_n}{\dashrightarrow} q_n \overset{x_n}{\rightsquigarrow} q'_n \overset{q'_0}{\Rightarrow} q'$, wobei $q'_i = q_i$, wenn $x_i = \epsilon$ und jedes nichtleere x_i hat eine geringere Tiefe als h. Dann kann man aufgrund der Induktionsannahme und der Definition von $\tilde{\delta}$ eine Stütze

$$K \overset{x_0}{\rightsquigarrow} K'_0 \overset{a_1}{\rightarrow} K_1 \overset{x_1}{\rightsquigarrow} K'_1 \dashrightarrow ... \dashrightarrow K_{n-1} \overset{a_n}{\dashrightarrow} K_n \overset{x_n}{\rightsquigarrow} K'_n \overset{K'_0}{\Rightarrow} K'$$

konstruieren und weil $(q, p) \in K$ ist, gilt:

$(q'_0, p) \in K'_0$ durch die Induktionsannahme auf die Stütze $q \overset{x_0}{\rightsquigarrow} q'_0$
$(q_1, q'_0) \in K_1$, denn $q_1 \in \delta_{push}(q'_0, a_1)$
$(q'_1, q'_0) \in K'_1$, durch die Induktionsannahme auf die Stütze $q_1 \overset{x_1}{\rightsquigarrow} q'_1$
$(q_i, q'_0) \in K_i$, denn $q_i \in \delta_{shift}(q'_{i-1}, a_1)$ für jedes $i = 2, ..., n$
$(q'_i, q'_0) \in K'_i$, durch die Induktionsannahme auf die Stütze $q_i \overset{x_i}{\rightsquigarrow} q'_i$
$(q', p) \in K'$, denn $q' \in \delta_{pop}(q_n, q'_0)$

Und

dadurch ist der Beweis abgeschlossen. □

Die Aussage des nächsten Lemmas führt genau anders herum, vom Automaten $\tilde{A}$ zu A.

Lemma 3.7. *Sei y der Rumpf einer Kette mit der Stütze $K \overset{y}{\rightsquigarrow} K'$ in $\tilde{A}$. Dann gilt für alle $p, q' \in Q$: Wenn $(q', p) \in K'$, dann existiert eine Stütze $q \overset{y}{\rightsquigarrow} q'$ in A.*

Beweis. Zunächst ein paar Anmerkungen, die für den Beweis verwendet werden:

i) Aus der Definition von δ_{push} folgt: Wenn es $\bar{K} \overset{a}{\rightarrow} K$ in $\tilde{A}$ gibt mit $(\bar{q}, q) \in K$, $(q, p) \in \bar{K}$, dann gibt es $q \overset{a}{\rightarrow} \bar{q}$ in A.

ii) Aus der Definition von δ_{shift} folgt: Wenn es $\bar{K} \overset{a}{\dashrightarrow} K$ in $\tilde{A}$ gibt mit $(r, q) \in K$, dann existiert ein $\bar{q} \in Q$, sodass $\bar{q} \overset{a}{\dashrightarrow} r$ in A und $(\bar{q}, q) \in K$.

iii) Aus der Definition von δ_{pop} folgt: Wenn es $\bar{K} \overset{K}{\Rightarrow} K$ in $\tilde{A}$ gibt mit $(q'q) \in \bar{K}$, dann existiert ein Paar $(r, q) \in \bar{K}$, sodass $(q, p) \in K$ und es gibt $r \overset{q}{\Rightarrow} q'$ in A.

Beweis durch Induktion auf die Höhe $h = d(y)$

Induktionsanfang: (h=1) Wenn h=1 ist, dann ist $y = a_1a_2...a_n$ mit der Stütze $K \overset{a_1}{\to} K_1 \overset{a_2}{\dashrightarrow} ... \overset{a_{n-1}}{\dashrightarrow} K_{n-1} \overset{a_n}{\dashrightarrow} K_n \overset{q_0}{\Rightarrow} K'$. Sei $(q', p) \in K$, dann gibt es wie in Anmerkung 3 in $\tilde{A}$ ein Paar $(q_n, q) \in K$, sodass $(q, p) \in K$ und $q_n \overset{q}{\Rightarrow} q'$. Weiterhin folgt aus Anmerkung 2 mit $(q_n, q) \in K$ und $K_{n-1} \overset{a_n}{\dashrightarrow} K_n$ die Existenz eines Zustandes $q_{n-1} \in Q$, sodass $(q_n, q) \in K_{n-1}$ und $q_{n-1} \overset{a_n}{\dashrightarrow} q_n$. Auf gleiche Weise gilt für alle $i = n-2, ..., 1$, dass es $q_i \in Q$ gibt, sodass $(q_i, q) \in K_i$ und $q_i \overset{a_{i+1}}{\dashrightarrow} q_{i+1}$ Schließlich folgt aufgrund Anmerkung 1 aus $K \overset{a_1}{\to} K_1$, $(q_1, q) \in K_1$ und $(q, p) \in K$, dass es $q \overset{a_1}{\to} q_1$ in A gibt. So wurde gezeigt, dass es eine Stütze für y gibt.

Induktionsschritt Nun gelte die Induktionsannahme für eine geringere Tiefe als h. Weiter sei $y = x_0a_1x_1a_2...a_nx_n$ mit der Tiefe h und der Stütze $K \overset{x_0}{\rightsquigarrow} K'_0 \overset{a_1}{\to} K_1 \overset{x_1}{\rightsquigarrow} K'_1 \dashrightarrow ... \dashrightarrow K_{n-1} \overset{a_n}{\dashrightarrow} K_n \overset{x_n}{\rightsquigarrow} K'_n \overset{K'_0}{\Rightarrow} K'$, wobei $K'_i = K_i$, wenn $x_i = \epsilon$ und jedes nichtleere x_i hat eine geringere Tiefe als h. Sei $(q', p) \in K'$. Da es aufgrund von Anmerkung 3 $K'_n \overset{K'_0}{\Rightarrow} K'$ gibt, existiert ein Paar $(q'_n, q'_0) \in K'_n$ in $\tilde{A}$ mit $(q'_0, p) \in K'_0$ und $q'_n \overset{q'_0}{\Rightarrow} q'$. Wenn $x_n \neq \epsilon$ existiert aufgrund der Induktionsannahme, da $(q'_n, q'_0) \in K'_n$, eine Stütze $q_n \overset{x_n}{\rightsquigarrow} q'_n$ mit $(q_n, q'_0 \in K_n$. Auf gleiche Weise gilt für alle $i = n-1, ...2, 1$: Es existieren q'_i und q_i ($q'_i = q_i$, wenn $x_i = \epsilon$), sodass es $q_i \overset{x_i}{\rightsquigarrow} q'_i \overset{a_{i+1}}{\dashrightarrow} q_{i+1}$ in A gibt,

mit $(q'_i, q'_0) \in K'_i$ (Aufgrund von Anmerkung 2, da $K'_i \overset{a_{i+1}}{\dashrightarrow} K_{i+1}$ in $\tilde{A}$ und $(q_{i+1}, q'_0) \in K_{i+1}$)

und $(q'_i, q'_0) \in K_i$ (Aufgrund der Induktionsannahme, da $K_i \overset{x_i}{\rightsquigarrow} K'_i$ in $\tilde{A}$ und $(q'_i, q'_0) \in K'_i$).

Das gilt speziell auch für $q_1 \overset{x_1}{} q'_1$ mit $(q_1, q'_0) \in K_1$. Weiterhin gibt es $(q'_0 \overset{a_1}{\to} q_1$, da $K'_0 \overset{a_1}{\to} K_1$ und $(q'_0, p) \in K'_0$ (Anmerkung 1).

Schließlich folgt aus der Induktionsannahme, da $(q'_0, p) \in K'_0$ und $K \overset{x_0}{} K'_0$, wenn $x_0 \neq \epsilon$ existiert ein Zustand $q \in Q$, sodass $q \overset{x_0}{\rightsquigarrow} q'_0$ in $\tilde{A}$ mit $(q, p) \in K$. Somit haben wir eine Stütze nach 2.12 (2) konstruiert und der Beweis ist abgeschlossen. □

Um den Beweis von 3.5 zu vervollständigen muss noch bewiesen werden, dass eine akzeptierende Berechnung für y in A gibt, dies auch in $\tilde{A}$ zutrifft.

Sei $y \in L(A)$. Dann gibt es eine Stütze $q \overset{y}{\rightsquigarrow} q'$, wobei $q \in I, q' \in F$. Dann folgt aus 3.2 für $(q_0, \top) \in K = I \times \{\top\}$, dass eine Stütze $K \overset{y}{\rightsquigarrow} K'$ in $\tilde{A}$ mit $(q', \top) \in K'$. $q' \in F$ impliziert $K' \in F'$ Andersherum sei $y \in L(\tilde{A})$. Dann

hat y eine Stütze $\tilde{K} \overset{y}{\rightsquigarrow} K'$ in $\tilde{A}$ mit $K' \in \tilde{F}$. Das heißt es existiert ein $q' \in F$, sodass $(q', \top) \in K'$. Aufgrund von 3.7 gibt es eine Stütze $q \overset{y}{\rightsquigarrow} q'$ in A mit $(q', \top) \in \tilde{K}$ und daraus folgt $q \in I$. Somit definiert $q \overset{y}{\rightsquigarrow} q'$ eine akzeptierende Berechnung für y in A. Damit ist der Beweis abgeschlossen. □

An dieser Stelle folgt eine weitere interessante Eigenschaft in Bezug auf Determinismus der Automaten, die durch die Konstruktion aus einer Grammatik wie in 3.1.1 entstehen.

Korollar 1. *Wenn die Ausgangsgrammatik in Fischer Normalform ist, dann ist der konstruierte Automat deterministisch.*

Diese Behauptung folgt direkt aus der Konstruktion, in der die Werte die durch δ_{push} und δ_{shift} definiert sind immer Singletons sind und die δ_{pop}-Funktion produziert soviele Zustände wie es linke Regelseiten mit denselben rechten Regelseiten gibt. Eben diese letzte Eigenschaft wird in Fischernormalform verhindert, da es keine sich wiederholenden rechten Regelseite gibt. Also ist der resultierende Automat deterministisch.

3.3 Abgeschlossenheit

Die folgenden Eigenschaften werden auf Ebene der Grammatiken geführt. Dazu müssen zunächst noch ein paar Definitionen getätigt werden. [4]

Definition 3.2. *Eine OPG G ist frei, wenn sie in homogener Normalform ist und für alle Nichtterminale A,B $\neq$ S gilt $\mathcal{L}_G(A) = \mathcal{L}_G(B) \wedge \mathcal{R}_G(A) = \mathcal{R}_G(B)$ impliziert $A = B$. Für eine OPM M ist $\mathcal{F}(M) = \{F | F \text{ ist } frei \wedge OPM(F) \subseteq M\}$ die Klasse der freien präzedenzkompatiblen OPGs.*

Lemma 3.8. *Für alle OPM M und $k \geq 1$ existiert eine eindeutige freie OPG $G_{M,k}$, sodass $OPM(G_{M,k}) = M$ und für alle $G \in C_{M,k}$ gilt $L(G) \subseteq L(G_{M,k}$*

Eine solche Grammatik $G_{M,k}$ wird als Max-Grammatik bezeichnet.

3.4 OPL im Vergleich mit anderen Sprachen

In diesem Abschnitt geht es um eine Einordnung in die verschiedenen wichtigen Sprachklassen. Sie sind offensichtlich eine Unterklasse der deterministisch-kontextfreien Sprachen (DCF), da die OPGs aufbauend aus einer kontextfreien Grammatik definiert werden und durch die Relationen, bzw. die Konfliktfreiheit der Matrix eingeschränkt sind. Eine typische DCF-Sprache, die aber nicht von OPGs erkannt wird ist

$$L_1 = \{a^n b a^n | n \geq 0\}.$$

Nachdem D. Knuth die LR(k)-Grammatiken eingeführt, die genau DCF erkennen und mit denen man effiziente Bottom-Up Parser konstruieren kann, ist das Interesse an OPGs fast erloschen. Dies änderte sich etwas mit der Einführung des OPA und den vorteilhaften Eigenschaften die diese Sprachklasse bietet.

3.4.1 Reguläre Sprachen

Eine interessante Eigenschaft betrifft die regulären Sprachen, die eine Untermenge der OPLs sind.[3]

Lemma 3.9. *Sei $L \subseteq \Sigma^*$ eine reguläre Sprache. Dann existiert eine Grammatik für L in der Familie $C_{M,2}$ der präzedenzkompatiblen Grammatiken mit einer Präzedenzmatrix, sodass $\forall a, b \in \Sigma : M_{ab} = \lessdot$*

Dies folgt aus der Darstellung von regulären Sprachen als kontextfreie Grammatik in der alle Regeln die Form $A \to aB,\ a \in \Sigma, B \in N$. Daher ergibt sich die Beschränkung der rechten Seiten auf 2.

3.4.2 Visibly Pushdown Sprachen

Eine weitere interessante Sprachklasse sind die Visibly Pushdown (VP) Sprachen. [5]

Definition 3.3 (Visibly Pushdown Automat)**.** *Ein VP Alphabet is ein Tripel $\hat{\Sigma} = (\Sigma_c, \Sigma_r, \Sigma_i)$ wobei die drei Teilalphabete disjunkt sind und calls, returns und internals genannt werden und $\Sigma = \Sigma_c \cup \Sigma_r \cup \Sigma_i$. Ein Visibly Pushdown Automat (VPA) ist ein Kellerautomat $A = (Q, \Sigma, \Gamma, \delta, q_0, F)$ mit dem VP Alphabet $\hat{\Sigma}$ und der Transitionsfunktion die zwischen folgenden Fällen unterscheidet:*

- *1. pop transition: $\delta(p, a, X) \ni (q, \epsilon), \quad a \in \Sigma_r$. Kurzform: $\delta(p, a, X) \ni q$*
- *2. pop on empty stack: $\delta(p, a, \bot) \ni (q, \bot), \quad a \in \Sigma_r$. Kurzform: $\delta(p, a, \bot) \ni q$*
- *3. internal transition: $\delta(p, a, X) \ni (q, X), \quad a \in \Sigma_i$. Kurzform: $\delta(p, a) \ni q$*
- *4. push transition: $\delta(p, a, X) \ni (q, XY), \quad a \in \Sigma_c$. Kurzform: $\delta(p, a) \ni (q, Y)$*

Eine Sprache über $\Sigma = \Sigma_c \cup \Sigma_r \cup \Sigma_i$ ist eine VPL, wenn sie von einem VPA mit VP alphabet $\hat{\Sigma}$ erkannt wird. Mit diesen Sprachklassen kann man sich ebenso ausführlich beschäftigen, wie mit den OPLs. Für diese Arbeit ist allerdings folgende Eigenschaft interessant[3]

Lemma 3.10. *Für jeden VPA A kann eine OPG G konstruiert werden, sodass* $L(G) = L(A)$.

Beweis. Zuerst wird ein Verfahren zur Konstruktion der Grammatik beschrieben, dann wird gezeigt, dass es sich dabei um eine OPG handelt und letzlich die Äquivalenz zu A. Für die Konstruktion wird zunächst Ergebnisse aus der Umgebung der Balancierten Sprachen und Visibly Pushdown Sprachen verwendet. Dabei werden die Buchstaben c,r und s für calls, returns und internals und die Kurzformen der Transitionen verwendet. Ein String in $\{c, r\}^*$ heisst *richtig geklammert*, wenn man ihn mit der Regel $cr \to \epsilon$ zu ϵ reduzieren kann. Sei ρ die Mappingfunktion von $\{\Sigma_c, \cup\Sigma_r \cup \Sigma_i\}$ auf $\{c, r\}$ als $\rho(c_j) = c, \forall c_j \in \Sigma_c$, $\rho(r_j) = r, \forall r_j \in \Sigma_r$ und $\rho(s_j) = \epsilon, \forall s_j \in \Sigma_i$ definiert. Dann nennt man einen String *richtig balanciert*, wenn $\rho(x)$ richtig geklammert ist und *richtig geschlossen* wenn zusätzlich $first(x) \in \Sigma_c$ und $last(x) \in \Sigma_r$ gilt.

Lemma 3.11. *Jedes Wort* $x \in L(A)$ *kann zerlegt werden als* $x = yc_0z$ *oder* $x = y$ *mit* $c_0 \in \Sigma_c$, *sodass*

- $y = u_1w_1u_2w_2...u_kw_k$, $k \geq 1$, *wobei* $u_j \in (\Sigma_i \cup \Sigma_r)^*$ *und* $w_j \in \Sigma^*$ *ist ein richtig abgeschlossener String*
  oder $y = u_1w_1u_2w_2...u_k$
- $z = v_1c_1v_2c_2...c_{r-1}v_r$, $r \geq 0$, *wobei* $c_j \in \Sigma_c$ *und* $v_j \in \Sigma^*$ *ist ein richtig balancierter String*

Beweis. Sei Á ein passender Visibly Pushdown Automat, den wir als nichtdeterministsich annehmen (da äquivalent zur deterministischen Version [5]) und $x = yc_0z$. Ohne Beschränkung der Allgemeinheit wird zusätzlch angenommen, dass der initiale Zustand q_0 nicht erneut erreicht wird.
Die Berechnung beginnt mit Transitionen zweiter und dritter Art, die u_1 einlesen, den Stack aber leer lassen. Dann beginnt der Automat w_1 einzulesen. Die erste Transition ist ein Push, bei der ein $Z_i \neq \perp$ auf den Stack gelegt wird. Danach folgen verschiedene moves, die einen richtig balancierten String String einlesen, gefolgt von einer Pop-Transition, die genau Z_i entfernt und

so der richtig geschlossene String w_1 eingelesen wurde. Nun beginnt wieder das Einlesen von u_2 und so weiter bis w_k vollständig eingelesen wurde. Es wird die Menge Q_q von Zuständen definiert, die während des Lesens von y erreicht wurden.
An einem bestimmten Punkt, wenn der Stack leer und das Eingabesymbol in Σ_c ist, ändert der Automat nichtdeterministisch sein Verhalten um $c_0 z$ einzulesen. Es wird eine Menge Q_p von Zuständen disjunkt von $Q_q \cup \{q_0\}$. Der Automat führt eine Pushtransition $\delta(r, c_0) \ni (r', Z_U)$ durch, wobei Z_U ein Symbol auf dem Stack bezeichnet, dass nicht mehr durch eine Pop-Transition entfernt wird, also kann c_0 als unbeantworteter Call betrachtet werden. Dann wird z eingelesen, wobei zunächst ein richtig balancierter String v_1 eingelesen wird. Dann wird wieder nichtdeterministisch eine Push-Transition ausgeführt, in der wieder ein Z_U auf den Stack gelegt wird. Schließlich endet die Berechnung irgendwann in einem akzeptierenden Zustand und einem Stack der Form $\perp Z_U^+$. Der String y steht für ein Wort (oder Präfix), dass mit einem leeren Stack endet. □

Bei der Konstruktion der Grammatik G wird für einen String x, der wie in 3.11 zerlegt wird, ein Syntaxbaum wie in 3.11 konstruiert. Zum Beispiel steht dabei das äußere linke s in 3.11 für den Substring u_1 im Lemma, *csr* korrespondiert zu w_1 usw. In der Abbildung steht Y für ein Nichtterminal, dass einen String generiert, bei dem der Automat mit einem leeren Stack startet und endet. Von Nichtterminalen B_1, B_2 werden richtig balancierte Strings abgeleitet. Nichtterminale X leiten Strings ab, sodass bei einem nichtleeren Stack $\perp Z_u^+$ kein Z gepoppt wird und am ende einen String in $\perp Z_u^+$ enthalten ist. Die Nichtterminale der Grammatik (außer S) werden entweder als Paar $(q_i, q_j), (p_i, p_j)$ oder als Tripel $(q_i, Z, q_j), (p_i, Z, q_j)$ mit $Z \in \Gamma$ beschrieben. Die Idee ist nun, dass Nichtterminale der Form $(t_i ... t_j)$ einen Terminalstring u generieren, genau dann wenn es eine Berechnung des Automaten von Zustand t_i zu t_j gibt, der den String einliest und den initialen Stack nicht poppt. Weiterhin steht (q_i, q_j) für Nichtterminale, die den Stack unverändert lassen, Nichtterminale (p_i, p_j) erhöhen wenn überhaupt die Anzahl an Z_us und Nichtterminale (q_i, Z, q_j) oder (p_i, Z, p_j) bedeuten, dass die Berechnung mit Z oben auf dem Stack startet und endet, während ein richtig ausbalancierter String w gelesen wurde. Nun werden die Regeln für G aus den Transitionen in A abgeleitet:
Für die Ableitungen muss zwischen vielen Fällen abhängig von den Zerlegungen aus 3.11wie in [figure] dargestellt, unterschieden werden. Diese Fälle sind in Tabelle 2 dargestellt.

Die so produzierte Grammatik ist nicht zwangsläufig reduziert, das heisst

	Σ_c	Σ_r	Σ_i
Σ_c	$\lessdot$	$\doteq$	$\lessdot$
Σ_r	$\gtrdot$	$\gtrdot$	$\gtrdot$
Σ_i	$\gtrdot$	$\gtrdot$	$\gtrdot$

Tabelle 1: Totale Matrix M_T

sie kann Regeln enthalten, deren linke Seite niemals in einer Ableitung des Startsymbols auftauchen. Diese nutzlosen Regeln können aber durch bekannte Algorithmen entfernt werden [11]. Damit eine Grammatik als Operatorpräzedenzgrammatik gilt muss sie sich in Operatorform befinden und weiterhin eine konfliktfreie OPM haben. Durch die Konstruktion entstehen nur Regeln in Operatorform. Was die OPM angeht, so müssen für alle Regeln die relevanten Terminalmengen ($\mathcal{R}_G(A)$ od[illegible](A)) gebildet werden. Hier wird das nur einmal exemplarisch gezeigt:

Für den Fall $Y \to YcBr$ mit der Regel $(q_0, q_n) \to (q_0, q_i)c(q_j, Z, q_m)r$ produziert $\mathcal{R}_G((q_0, q_i)) \subseteq \Sigma_i \cup \Sigma_r$ die Relationen $s \gtrdot c,\ r \gtrdot c$. Die Menge $\mathcal{L}_G((q_j, Zq_m)) \subseteq \Sigma_i \cup \Sigma_c$ produziert $c \lessdot c,\ c \lessdot s$ und aus $\mathcal{R}_G((q_j, Zq_m)) \subseteq \Sigma_i \cup \Sigma_r$ folgt $s \gtrdot r,\ r \gtrdot r$. Die rechte Regelseite impliziert $c \doteq r$. Allein aus dieser Regel folgt eine konfliktfreie Matrix $M \subseteq M_T$, wobei M_T die totale Matrix ist. Alle weiteren Regeln produzieren ähnliche Matrizen, die aber alle untereinander konfliktfrei sind. Die erwähnte totale Matrix hat immer die Form Tabelle 1 für alle Operatorpräzedenzgrammatiken mit einer solchen Matrix ist L(G) eine Visibly Pushdown Sprache [3]. □

Fall	Transitionen	Regeln in G
$S \to Yc_0X$	$\delta(q_i, c_0) \ni (p_j, Z_U)$	$S \to (q_0, q_i)c_0(p_i, p_f), \forall p_f \in Q_F$
$S \to Y$		$S \to (q_0, q_f), \forall q_f in Q_F$
$S \to Yc_0$	$\delta(q_i, c_0) \ni (p_f, Z_U), q_f in Q_F$	$S \to (q_0, q_f)c_0, \forall p_f \in Q_F$
$S \to c_0Xs$	$\delta(q_0, c_0) \ni (p_j, Z_U)$	$S \to c_0(q, p_j, p_f), \forall p_f \in Q_F$
$S \to c_0$	$\delta(q_0, c_0) \ni (p_f, Z_U), p_f \in Q_F$	$S \to c_0$
$Y \to s$	$\delta(q_0, s) \ni q_i$	$(q_0, q_i) \to s$
$Y \to r$	$\delta(q_0, r, \bot) \ni q_i$	$(q_0, q_i) \to r$
$Y \to Ys$	$\delta(q_i, s) \ni q_j$	$(q_0, q_j) \to (q_0, q_i)s$
$Y \to Yr$	$\delta(q_i, r, \bot) \ni q_j$	$(q_0, q_j) \to (q_0, q_i)r$
$Y \to cBr$	$\delta(q_0, c) \ni (q_t, Z),\ \delta(q_k, r, Z) \ni q_h$	$(q_0, q_h) \to c(q_t, Z, q_k)r$
$Y \to cr$	$\delta(q_0, c) \ni (q_t, Z),\ \delta(q_t, r, Z) \ni q_h$	$(q_0, q_h) \to cr$
$Y \to YcBr$	$\delta(q_i, c) \ni (q_j, Z),\ \delta(q_m, r, Z) \ni q_n$	$(q_0, q_n) \to (q_0, q_i)c(q_j, Z, q_m)r$
$Y \to Ycr$	$\delta(q_i, c) \ni (q_j, Z),\ \delta(q_j, r, Z) \ni q_n$	$(q_0, q_n) \to (q_0, q_i)cr$
$B_1 \to B_1cB_1r$	$\delta(q_i, c) \ni (q_j, Z),\ \delta(q_m, r, Z) \ni q_n$	$(q, q_n) \to (q, q_i)c(q_j, Z, q_m)r, \forall q \in Q_q$
$B_1 \to B_1cr$	$\delta(q_j, c) \ni (q_j, Z),\ \delta(q_j, r, Z) \ni q_n$	$(q, q_n) \to (q, q_n)cr, \forall q \in Q_q$
$B_1 \to cB_1r$	$\delta(q_i, c) \ni (q_j, Z),\ \delta(q_m, r, Z) \ni q_n$	$(q_i, q_n) \to c(q_j, Z, q_m)r$
$B_1 \to cr$	$\delta(q_i, c) \ni (q_j, Z),\ \delta(q_j, r, Z) \ni q_n$	$(q_i, q_n) \to cr$
$B_1 \to B_1cB_1r$	$\delta(q_i, c) \ni (q_j, Z),\ \delta(q_m, r, Z) \ni q_n$	$(q_i, W, q_n) \to (q, q_i)c(q_j, Z, q_m)r, \forall q \in Q_q, W \in \Gamma$
$B_1 \to cB_1r$	$\delta(q_i, c) \ni (q_j, Z),\ \delta(q_m, r, Z) \ni q_n$	$(q, W, q_n) \to c(q_j, Z, q_m)r, \forall q \in Q_q, W \in \Gamma$
$B_1 \to B_1cr$	$\delta(q_i, c) \ni (q_j, Z),\ \delta(q_j, r, Z) \ni q_n$	$(q, W, q_n) \to (q, q_i)cr, \forall q \in Q_q, W \in \Gamma$
$B_1 \to B_1s$	$\delta(q_h, s) \ni q_m$	$(q, W, q_m) \to (q, q_h)s, \forall q \in Q_q, W \in \Gamma$
$B_1 \to s$	$\delta(q_j, s) \ni q_m$	$(q_j, W, q_m) \to s, \forall W \in \Gamma$
$B_2 \to B_2cB_2r$	$\delta(q_i, c) \ni (q_j, Z),\ \delta(q_m, r, Z) \ni q_n$	$(p, q_n) \to (p, q_i)c(q_j, Z, q_m)r, \forall p \in Q_p$
$B_2 \to B_2cr$	$\delta(q_j, c) \ni (q_j, Z),\ \delta(q_j, r, Z) \ni q_n$	$(p, q_n) \to (p, q_n)cr, \forall p \in Q_p$
$B_2 \to cB_2r$	$\delta(q_i, c) \ni (q_j, Z),\ \delta(q_m, r, Z) \ni q_n$	$(q_i, q_n) \to c(q_j, Z, q_m)r$
$B_2 \to cr$	$\delta(q_i, c) \ni (q_j, Z),\ \delta(q_j, r, Z) \ni q_n$	$(q_i, q_n) \to cr$
$B_2 \to B_2cB_2r$	$\delta(q_i, c) \ni (q_j, Z),\ \delta(q_m, r, Z) \ni q_n$	$(q_i, W, q_n) \to (p, q_i)c(q_j, Z, q_m)r, \forall p \in Q_p, W \in \Gamma$
$B_2 \to cB_2r$	$\delta(q_i, c) \ni (q_j, Z),\ \delta(q_m, r, Z) \ni q_n$	$(p, W, q_n) \to c(q_j, Z, q_m)r, \forall p \in Q_p, W \in \Gamma$
$B_2 \to B_2cr$	$\delta(q_i, c) \ni (q_j, Z),\ \delta(q_j, r, Z) \ni q_n$	$(p, W, q_n) \to (p, q_i)cr, \forall p \in Q_p, W \in \Gamma$
$B_2 \to B_2s$	$\delta(q_h, s) \ni q_m$	$(q, W, q_m) \to (p, q_h)s, \forall p \in Q_p, W \in \Gamma$
$B_2 \to s$	$\delta(q_j, s) \ni q_m$	$(q_j, W, q_m) \to s, \forall W \in \Gamma$
$X \to cX$	$\delta(p_i, c) \ni (p_j, Z_U)$	$(p_i, p_f) \to c(p_j, p_f), \forall p_f \in Q_F$
$X \to c$	$\delta(p_i, c) \ni (p_f, Z_U), p_f \in Q_F$	$(p_i, p_f) \to c$
$X \to BcX$	$\delta(p_j, c) \ni (p_h, Z_U)$	$(p, p_f) \to (p, p_j)c(p_h, p_f), \forall p_f \in Q_F, p \in Q_p$
$X \to Bc$	$\delta(p_j, c) \ni (p_f, Z_U), p_f \in Q_F$	$(p, p_f) \to (p, p_j)c$
$X \to BcBr$	$\delta(p_i, c) \ni (p_j, Z), \delta(p_m, r, Z) \ni p_n$	$(p, p_f) \to (p, p_i)c(p_j, Z, p_m)r, \forall p_f \in Q_F, p \in Q_p$
$X \to cBr$	$\delta(p_i, c) \ni (p_j, Z), \delta(p_m, r, Z) \ni p_n$	$(p_i, p_f) \to c(p_j, Z, p_n)r, \forall p_f \in Q_F$
$X \to cr$	$\delta(p_i, c) \ni (p_j, Z), \delta(p_j, r, Z) \ni p_n$	$(p_i, p_f) \to c(p_j, Z, p_n)r, \forall p_f \in Q_F$
$X \to Bs$	$\delta(p_j, s) \ni p_f, p_f \in Q_F$	$(p, p_f) \to (p, p_j)s, \forall p \in Q_p$
$X \to s$	$\delta(p_j, s) \ni p_f, p_f \in Q_F$	$(p, p_f) \to s$

Tabelle 2: Übersicht der Regelgeneration

4 Implementierung der OPA

Wird eh überbewertet

5 Fazit

In dieser Arbeit wurden die grundlegenden Funktionen und Definitionen der Operatorpräzedenzsprachen zusammengefasst in Hinblick auf die Grammatiken und den Automaten. Weiterhin sind wichtige sprachtheoretische Eigenschaften beleuchtet und bewiesen worden, sowie in einer praktischen Anwendung implementiert worden. Allerdings gibt es in dem Bereich noch deutlich mehr Potenzial, was in dieser Bachelorarbeit nicht betrachtet wurde, aber zumindest an dieser Stelle als Ausblick erwähnt werden sollte:

- Eine monadische Prädikatenlogik zweiter Stufe wurde für die OPL definiert [2].
- Die Sprache der OPL wurde erweitert zu ω-OPL für unendliche Wörter. Auch dazu wurde die Prädikatenlogik zweiter Stufe entworfen.[2].
- Ansätze für ModelChecking von Operatorpräzedenzsprachen [6]
- Weitere Algebraische Eigenschaften der Operatorpräzedenzsprachen [4]
- Betrachtung der Sprachen, mit Konflikt in der Matrix
- ...

All das sind Gründe warum erneutes Interesse an diesen Sprachen angebracht wäre.

Literatur

[1] Violetta Lonati, Dino Mandrioli, and Matteo Pradella. Precedence automata and languages. *Computer science – theory and applications. 6th international computer science symposium in Russia, CSR 2011, St. Petersburg, Russia, June 14–18, 2011. Proceedings*, 2011.

[2] Violetta Lonati, Federica Panella Dino Mandrioli, and Matteo Pradella. Operator precedence languages: Their automata-theoretic and logic characterization. *SIAM Journal on Computing*, 2015.

[3] Stefano Crespi-Reghizzi and Dino Mandrioli. Operator precedence and the visibly pushdown property. *Journal of Computer and System Sciences*, 2012.

[4] Stefano Crespi-Reghizzi, Dino Mandrioli, and David F. Martin. Algebraic properties of operator precedence languages. *Information and control*, 1978.

[5] Rajeev Alur and P.Madhusudan. Visibly pushdown languages. *ACM Symposium on Theory of Computing*, 2004.

[6] Michele Chiari, Dino Mandrioli, and Matteo Pradella. Temporal logic and model checking for operator precedence languages. *9th Symposium on Games, Automata, Logics and Formal Verification*, 2018.

[7] Stefano Crespi Reghizzi and Dino Mandrioli. Algebraic properties of structured context-free languages: old approaches and novel developments. *WORDS*, 2009.

[8] Robert W. Floyd. Syntactic analysis and operator precedence. *Journal of the ACM*, 1963.

[9] Michael A. Harrison. *Introduction to Formal Language Theory*. Addison-Wesley Longman Publishing Co, 1978.

[10] A. Salomaa. *Formal Languages*. Acm monograph series. Academic Press, 1973.

[11] John Hopcroft and Jeffrey Ullman. *Introduction to AUtomata and Formal Languages*. Addison-Wesley, 1979.

Plagiatserklärung der / des Studierenden

Hiermit versichere ich, dass die vorliegende Arbeit über Titten _____ selbstständig verfasst worden ist, dass keine anderen Quellen und Hilfsmittel als die angegebenen benutzt worden sind und dass die Stellen der Arbeit, die anderen Werken – auch elektronischen Medien – dem Wortlaut oder Sinn nach entnommenen wurden, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht worden sind.

29 02 19, nAh
(Datum, Unterschrift)

Ich erkläre mich mit einem Abgleich der Arbeit mit anderen Texten zwecks Auffindung von Übereinstimmungen sowie mit einer zu diesem Zweck vorzunehmenden Speicherung der Arbeit in eine Datenbank einverstanden.

Fuckoff, da kannst du widersprechen
(Datum, Unterschrift)