Bachelorarbeit zum Thema:

Operatorpräzedenzsprachen und ihre Automaten

vorgelegt am: 5. März 2019

Westfälische Wilhelms-Universität 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 (sowohl 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 verdeutlicht werden muss. Er definiert dafür drei Präzedenzrelationen ($\lessdot, \gtrdot, \doteq$), die zwischen den Terminalsymbolen gelten und in einer Operatorpräzedenzmatrix (OPM) festgehalten werden.
Als motivierendes Beispiel, was auch in den Definitionen weiterhin genutzt wird, dient ein einfacher arithmetischer Ausdruck mit $\Sigma = \{n, +, \times, (,)\}$:
$n \times (n + n) + n$, 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 besondere Eigenschaften: Sie sind eine echte Teilmenge der kontextfreien Sprachen, genießen aber trotzdem alle typischen Abschlusseigenschaften von regulären Sprachen in Bezug auf Boolsche Operatoren, Konkatenation und Kleene-*. Weiterhin wird gezeigt, dass die Klasse der Visibly-Pushdown Sprachen und Automaten ebenfalls von OPGs und OPAs erkannt werden. Im Gegensatz zu anderen Parsern wie z.B. dem LR(1)- Parser muss hier nicht zwangsläufig von links nach rechts gearbeitet werden, was eine Möglichkeit zur Parallelisierung bietet. Besonderer Fokus liegt auf den Automaten, weswegen eine Bibliothek zum Erstellen von Operatorpräzedenzautomaten implementiert wird. Insgesamt wird in dieser Arbeit ein detaillierter Überblick über die Operatorpräzedenzsprachen geschaffen, da deren Potenzial noch nicht voll ausgeschöpft wurde.

2 Grundlagen und Definitionen

In diesem Abschnitt geht es um die Definitionen und Namenskonventionen, die für Operatorpräzedenzsprachen (OPL) 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 $N \cup \Sigma$. Wenn $S \overset{*}{\Rightarrow} \alpha$, nennt man α eine *Satzform*. 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. Für eine kfG G kann eine *geklammerte Grammatik* $\tilde{G}$ erstellt werden, in der jede rechte Regelseite mit Klammersymbolen $[,] \notin \Sigma$ umgeben ist. Zwei Grammatiken G und G' sind *äquivalent*, wenn sie die gleiche Sprache generieren, $L(G) = L(G')$. Wenn zusätzlich gilt $L(\tilde{G}) = L(\tilde{G'})$ sind sie auch *strukturell äquivalent*.

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 Operatorpräzedenz-

grammatiken (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 $B \in N \cup \{\epsilon\}$

Der Name G der Grammatik wird ausgelassen, sofern der Kontext klar ist. Eines der wichtigsten Merkmale von Operatorpräzedenzgrammatiken ist die Definition von drei binären Operatorpräzedenzrelationen.

Definition 2.2 (Präzedenzrelationen)**.**
Gleiche Präzedenz: $a \doteq b \Leftrightarrow \exists A \to \alpha aBb\beta, B \in N \cup \{\epsilon\}$
Übernimmt Präzedenz: $a \gtrdot b \Leftrightarrow \exists A \to \alpha Db\beta, D \in N$ *and* $a \in \mathcal{R}_G(D)$
Gibt Präzedenz ab: $a \lessdot b \Leftrightarrow \exists A \to \alpha aD\beta, D \in N$ *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 aufweisen. Weiterhin schließt die Gültigkeit einer Relation die andere nicht aus, sodass z.B. sowohl $a \lessdot b$ als auch $a \doteq b$ gelten kann. Für eine OG G kann eine Operatorpräzedenzmatrix (OPM) $M = OPM(G)$ als $|\Sigma| \times |\Sigma|$ erstellt 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 Floyd Grammatik 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 OPM M ist die Klasse der präzedenzkompatiblen OPGs als $C_M = \{G \in OPG | OPM(G) \subseteq M\}$ definiert. 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: $\forall a \in \Sigma : \# \lessdot a \ \wedge \ a \gtrdot \#$. Ferner gilt: $M_{\# \#} = \{\doteq\}$.

Definition 2.5 (OP Alphabet)**.** *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 $A \to A_1 a_1 ... A_\mathrm{n} a_\mathrm{n} A_{\mathrm{n}+1}$, bei der alle A_i potenziell fehlen können, die Relationen $a_1 \doteq a_1 \doteq ... \doteq a_\mathrm{n}$ gebildet werden. Problematisch wird dies, wenn die $\doteq$-Relation *zyklisch* ist d.h. es gibt $a_1, a_2, ..., a_\mathrm{m} \in \Sigma (m \geq 1)$, sodass $a_1 \doteq a_2 \doteq ... \doteq a_\mathrm{m} \doteq a_1$. Das führt dazu, dass die Länge der rechten Seite einer Regel keine Begrenzung hat. Bei einer Einschränkung wird unterschieden zwischen den *rechts begrenzten* Grammatiken der Klasse $C_{M,k}$, bei der die rechten Regelseiten kleiner als k sein müssen und den Grammatiken der Klasse $C_{M,\doteq}$, bei der die $\doteq$-Relation als azyklisch angenommen wird. Dies kann einen Einfluss auf die Konstruktion mancher Grammatiken haben, allerdings betrifft es keine der hier verwendeten Grammatiken. In den folgenden Abschnitten werden alle Grammatiken als Teil der Klasse $C_{M,\doteq}$ angenommen, sofern nicht anders erwähnt.
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 nach [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 es 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 im Regelfall mit q_i, p_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 zweiHilfsfunktionen eingeführt: $symbol([a, q]) = a$, $symbol(\bot) = \#$ 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 (verbleibende) Eingabewort bezeichnet.
Bei einem *Lauf* oder einer *Berechnung* 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* $r \in \delta_{pop}(q, p)$

Praktisch 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 Push-Transition zusätzlich die Terminale und die Flush-(Pop)funktion baut den Stack bis zur Markierung ab. Das führt zu einer komplizierteren Version 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 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 und 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* $x = a_0x_0a_1x_1a_2...a_nx_na_n$ *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* $M_{a_0a_{n+1}} \neq \emptyset$*.*

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* ${}^{a_0}[a_1a_2...a_n]^{a_{n+1}}$ *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 ${}^{a_0}[x_0a_1x_1a_2...a_nx_n]^{a_{n+1}}$ *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:*

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

Es ist wichtig zu beachten, dass der einzige Pop-Move am Ende mit genau dem ersten Zustand der Kette q_0 , bzw. bei zusammengesetzten Ketten

	+	×	(	)	n
+	$\gtrdot$	$\lessdot$	$\lessdot$	$\gtrdot$	$\lessdot$
×	$\gtrdot$	$\gtrdot$	$\lessdot$	$\gtrdot$	$\lessdot$
(	$\lessdot$	$\lessdot$	$\lessdot$	$\doteq$	$\lessdot$
)	$\gtrdot$	$\gtrdot$		$\gtrdot$	
n	$\gtrdot$	$\gtrdot$		$\gtrdot$	

Abbildung 1: OPM der Grammatik G

q_0' beschriftet ist. 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 vorerst abgeschlossen.

2.2.1 Beispiel

Es folgen zur Veranschaulichung ein praktisches Beispiel anhand einer Grammatik G für einfache arithmetische Ausdrücke:
$G = (N, \Sigma, P, E)$ mit $N = \{E, T, F\}, \Sigma = \{+, \times, (,), n\}$ und
$P = \{E \to E + T | E, T \to E \times F | F, F \to n | (E)\}$.
Diese Grammatik befindet sich nicht in FNF, kann aber dazu umgewandelt werden.
Aus dieser Grammatik können jetzt die Terminalmengen abgeleitet werden, um die Präzedenz zwischen den Terminalsymbolen zu berechnen. In Abbildung 1 ist die OPM(G) vollständig aufgeführt. Wie man sieht ist die OPM konfliktfrei, bei der Grammatik handelt es sich also um eine OPG.

Für die OPM wird ein Beispielautomat A wie in Abbildung 2 konstruiert. Als Eingabe dient der arithmetische Ausdruck $n \times n + (n + n) \times n \in L(G)$. Für den Automaten A wird in Tabelle 1 eine Berechnung der Eingabe dargestellt.

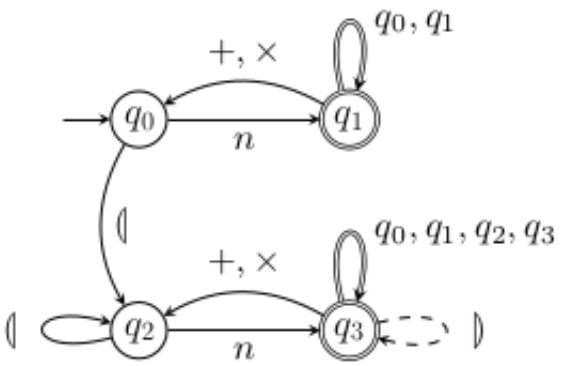

Abbildung 2: Beispiel OPA

Stack	Zustand	Eingabe
$\perp$	q_0	$n \times (n + n) + n\#$
$\perp\,[n, q_0]$	q_1	$\times(n + n) + n\#$
$\perp$	q_1	$\times(n + n) \times n\#$
$\perp\,[\times, q_1]$	q_0	$(n + n) \times n\#$
$\perp\,[\times, q_1]\,[(, q_0]$	q_2	$n + n) + n\#$
$\perp\,[\times, q_1]\,[(, q_0]\,[n, q_2]$	q_3	$+n) + n\#$
$\perp\,[\times, q_1]\,[(, q_0]$	q_3	$+n) + n\#$
$\perp\,[\times, q_1]\,[(, q_0]\,[+, q_3]$	q_2	$n) + n\#$
$\perp\,[\times, q_1]\,[(, q_0]\,[+, q_3]\,[n, q_2]$	q_3	$) + n\#$
$\perp\,[\times, q_1]\,[(, q_0]\,[+, q_3]$	q_3	$) + n\#$
$\perp\,[\times, q_1]\,[(, q_0]$	q_3	$) + n\#$
$\perp\,[\times, q_1]\,[), q_0]$	q_3	$+n\#$
$\perp\,[\times, q_1]$	q_3	$+n\#$
$\perp$	q_3	$+n\#$
$\perp\,[+, q_3]$	q_2	$n\#$
$\perp\,[+, q_3]\,[n, q_2]$	q_3	$\#$
$\perp\,[+, q_3]$	q_3	$\#$
$\perp$	q_3	$\#$

Tabelle 1: Berechnung der Eingabe durch den Automaten. Jede Zeile steht dabei für eine Konfiguration

3 Eigenschaften von OPLs

Dieser Abschnitt widmet sich den 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. [1, 2]

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 Summe 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: Wenn (A, γ) so ein Zustand ist, dann ist A die linke Regelseite und γ das Präfix, was vorher unter Konstruktion war.

Damit ist die Konstuktion des Automaten abgeschlossen. Die Äquivalenz zwischen G und A leitet sich nun aus den folgenden Lemmata 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 Shift-Funktionen, 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 Shift-Funktionen 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 ist 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, die 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 direkt und intuitiv 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 $0 \leq i \leq n, \quad \Delta_i = ({}_{i}^{a}q_i, q_i^{\prime a_{i+1}})$, 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 trivial 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 verschiedenen 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. Weiterhin 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 determinstsiche 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 die 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, definiert als 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 wächst exponential: Sei s = $|Q|$ die Anzahl der Zustände vom nichtdeterministischen Automaten A, dann ist $|\tilde{Q}| = 2^{|Q|*|Q\cup\{\top\}|}$. Also hat $\tilde{A}$ $2^{O(s^2)}$ Zustände.
Die Äquivalenz der beiden Automaten basiert auf den Aussagen der beiden folgenden Lemmata, die in die jeweilige Richtung 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 eine geringere Tiefe als h hat. Dann kann aufgrund der Induktionsannahme und der Definition von $\tilde{\delta}$ eine Stütze

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

konstruiert werden 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 $i = 2, ..., n$ Und
$(q'_i, q'_0) \in K'_i$, durch die Induktionsannahme auf die Stütze $q_i \overset{x_i}{\rightsquigarrow} q'_i$
$(q', p) \in K'$, denn $q' \in \delta_{pop}(q_n, q'_0)$

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 $(\bar{q}, q) \in K, \quad (q, p) \in \bar{K}$, 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 $q_1 \overset{x_1}{\rightsquigarrow} q'_1$ 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 $K \overset{x_0}{\rightsquigarrow} K'_0$, 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 $K' \in \tilde{F}$. D.h., 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 3.1. *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

OPG weisen wie reguläre Sprachen und im Gegensatz zu deterministisch-kontextfreien Sprachen Abgeschlossenheit unter typischen Sprachoperationen auf: Vereinigung, Komplement, Schnitt, Spiegelung, Präfix, Suffix, Konkatenation und Kleene-*. Diese Eigenschaften werden auf Ebene der Grammatiken betrachtet. In [4] formulieren Reghizzi und Mandrioli ein algebraisches Gitter und umfangreiche algebraische Eigenschaften, unter anderem eine partielle Ordnung und folgern daraus die Abgeschlossenheit unter Vereinigung, Schnitt und Komplement. Zuvor müssen zunächst allerdings noch ein paar Definitionen getätigt werden.

Definition 3.2 (Homogene Normalform)**.** *Eine OPG ist in homogener Normalform, wenn sie in Fischer Normalform ist und für alle Regeln $A \to \alpha$, außer $A = S$ gilt $\mathcal{L}(A) = \mathcal{L}(\alpha) \ \wedge \ \mathcal{R}(A) = \mathcal{R}(\alpha)$. Sei $\mathcal{H}$ die Klasse der homogenen OPGs.*

Das bedeutet, dass in einer homogenen Grammatik alle Regeln eines Nichtterminals die gleichen Terminalsets haben. Für jede OPG G kann eine strukturell äquivalente OPG H effektiv konstruiert werden. Die nächsten Definitionen betreffen die rechts-begrenzten Operatorpräzedenzgrammatiken.

Definition 3.3 (freie OPG und Max-Grammatik)**.**

- *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* $\mathcal{F}(M) = \{F | F\ ist\ frei\ \wedge\ OPM(F) \subseteq M\}$ *die Klasse der freien präzedenzkompatiblen OPGs.*

- *Für eine OPM M und* $k \geq 1$*, existiert eine eindeutige freie OPG* $G_{M,k}$ *mit* $OPM(G_{M,k}) = M \wedge \forall G \in C_{M,k} : L(G) \subseteq L(G_{M,k})$
  Eine solche Grammatik $G_{M,k}$ *wird als Max-Grammatik bezeichnet.*

Die Teilmengen Relation zwischen den Regelmengen der Grammatiken $F_1, F_2 \in \mathcal{F}(M)$ induziert eine partielle Ordnung $F_1 \leq F_2$ der freien Grammatiken. In [4] wird ein algebraisches Gitter definiert mit dieser partiellen Ordnung, sodass die Max-Grammatik das oberste Element ist.

Definition 3.4. *Sei* $H = (N, \Sigma, P, S)$ *eine homogene OPG mit OPM M. Die Mappingfunktion*

$$W : N \setminus \{S\} \to \mathcal{P}(\Sigma) \times \mathcal{P}(\Sigma)$$

ist definiert als

$$W(A) = (\mathcal{L}(A), \mathcal{R}(A))$$

und wird intuitiv für die Regeln erweitert ($W(a) = a, \forall a \in \Sigma$*). Eine freie Grammatik* $F = W(H)$ *ist dann* $F = (W(N) \cup \{S\}, \Sigma, W(P), S)$.

Lemma 3.8. *Aus* $(\tilde{H}) \subseteq L(\tilde{F})$ *mit* $H \in \mathcal{H}, F \in \mathcal{F}$ *folgt* $W(H) \leq F$.

Durch die Mappingfunktion W wird $\mathcal{H}$ in disjunkte Klassen aufgeteilt. Für jede dieser Klassen $W^{-1}(F)$gilt: $\forall H \in W^{-1}(F)$ mit $F \in \mathcal{F}$ gilt: $L(H) \subseteq L(F)$.

Die Klasse der freien OPGs, die einer freien OPG F in der partiellen Ordnung vorangehen wird als $\mathcal{F}(F) = \{F' \in \mathcal{F} | F' \leq F\}$ definiert. Weiterhin wird die Klasse der homogenen OPGs für eine freie Grammatik definiert: $\mathcal{H}(F) = \{H \in \mathcal{H} | W(H) \in \mathcal{F}(F)\}$. Praktisch bedeutet das $\mathcal{H}(F)$ enthält alle homogenen Grammatiken, deren Regel die gleiche Form haben wie F. Daraus folgt, dass die OPMs dieser Grammatiken mit der OPM(F) kompatibel sind.

Satz 3.1 (Vereinigung)**.** *Die Menge der Sprachen* $\{L(H) | H \in \mathcal{H}(F)\}$ *ist abgeschlossen unter Vereinigung*

Beweis. Sei $H_1 = (N_1, \Sigma, P_1, S) \in W^{-1}(F_1)$ und $H_2 = (N_2, \Sigma, P_2, S) \in W^{-1}(F_2)$ mit $F_1, F_2 \in \mathcal{F}(F)$. O.b.d.A. wird angenommen, dass $V_1 \cap V_2 = \{S\}$. Es wird nun die vereinigte Grammatik $G = (N_1 \cup N_2, \Sigma, P_1 \cup P_2, S)$ konstruiert und es gilt $L(G) = L(H_1) \cup L(H_2)$. Aus $OPM(H_1), OPM(H_2) \leq OPM(F)$ und den Definitonen der algebraischen Eigenschaften wird $L(\tilde{H_1}) \cup L(\tilde{H_2}) \subseteq L(\tilde{F_1}) \cup L(\tilde{F_2}) \subseteq L(\widetilde{F_1 \cup F_2}) \subseteq \tilde{F'}$ gefolgert. Jetzt wird aus G eine homogene OPG H konstruiert, sodass $L(\tilde{G}) = L(\tilde{H}$. Da $L(\tilde{H}) \subseteq L(\tilde{F})$ folgt aus 3.8, dass $W(H) \leq F$ und damit ist der Beweis abgeschlossen. □

Satz 3.2 (Komplement)**.** *Die Menge der Sprachen* $\{L(H)|H \in \mathcal{H}(F)\}$ *ist abgeschlossen unter Komplement zu* $L(F)$.

Beweis. Sei $H \in \mathcal{H}(F)$, dann ist $L(H) \subseteq L(F)$. Es folgt aus der Definition der algebraischen Defintionen $\bar{L} = L(F) - L(H) = h(L(\tilde{F}) - L(\tilde{H}))$, wobei h der Homomorphismus ist, der die Klammern entfernt. Es kann eine Grammatik G effektiv erzeugt werden, sodass $L(\tilde{G}) = L(\tilde{F}) - L(\tilde{H})$ [12]. Diese ist eine OPG da $L(\tilde{G}) \subseteq L(\tilde{F})$ und $OPM(G) \subseteq OPM(F)$. Daraus kann wiederum eine homogene OPG H' konstruiert werden, sodass $L(\tilde{H'} = L(\tilde{G})$ und $L(H') = \bar{L}$ und damit ist der Beweis abgeschlossen. □

Korollar 3.2 (Schnitt)**.** *Die Menge der Sprachen* $\{L(H)|H \in \mathcal{H}(F)\}$ *ist abgeschlossen unter Schnitt.*
Das folgt aus den Gesetzen von DeMorgan: $L_1 \cap L_2 = \overline{\overline{L_1} \cup \overline{L_2}}$

Satz 3.3 (Spiegelung)**.** *OPG ist abgeschlossen unter Spiegelung.*

Das folgt direkt aus der links-rechts symmetrischen Definition der OPG. Wenn für eine OPG G $a \lessdot b$ gilt, kann eine Grammatik G^R konstruiert werden in der die rechten Regelseiten umgedreht werden und so $b \gtrdot a$ enthält. Gleiches gilt für $a \gtrdot b$ und $a \doteq b$ wird zu $b \doteq a$. G^R ist wiederum eine OPG mit einer „gespiegelten"Matrix $OPM(G^R)$.

Als nächstes werden Präfix- und Suffixoperatoren betrachtet. Für eine beliebige Sprache L ist $L_P = \{y|yz \in L, z \in \Sigma^*\}$ die Präfixsprache und $L_S = \{y|xy \in L, x \in \Sigma^*\}$ die Suffixsprache (kurz $pref(L), suf(L)$).

Satz 3.4 (Präfix und Suffix)**.** *OPG ist abgeschlossen unter Präfix- und Suffixoperationen.*

Beweis. Hier wird der Beweis lediglich für die Präfixoperation geführt, für die Suffixoperation ist der Beweis ähnlich. Zunächst wird eine Grammatik G' aus einer gegebenen OPG G konstruiert, dann wird bewiesen, dass $OPM(G) = OPM(G')$ und $L(G') = pref(L(G))$.

Sei $G = (N, \Sigma, P, S)$ eine OPG in Fischer Normalform. Die neue Grammatik ist dann $G' = (N', \Sigma, P', S')$, wobei N' die Vereinigung der zwei disjunkten Mengen

$$N^C = \left\{A^C | A \in N \setminus \{S\}\right\},\ N^S = \left\{A^S | A \in N \setminus \{S\}\right\}.$$

ist. Sei $\alpha = \beta A$ eine Satzform, dann ist A ein *Suffix einer Satzform* (SSF). Die Nichtterminale A^S werden so definiert, dass sie nur solche SSF von G' ableiten. Es werden zwei Transformationen α^C und α^S für Strings $\alpha = x_0 A x_1 ... A_n x_n, \quad x_i \in \Sigma^*$ definiert:

$$\alpha^C = \begin{cases} x_0 A_1^C x_1 ... A_n^C x_n & x_n \neq \epsilon \\ x_0 A_1^C x_1 ... A_n^C & x_n = \epsilon \end{cases}$$

$$\alpha^S = \begin{cases} undefiniert & x_n \neq \epsilon \\ x_0 A_1^C x_1 ... A_{n-1}^C x_{n-1} A_n^S & x_n = \epsilon \end{cases}$$

Weiterhin wird die Transformation $PREF(\alpha) = \{\beta_i^S | \beta_i \neq \epsilon\ ist\ Praefix\ von\ \alpha\} \cup \{\beta_i^C\ ist\ Praefix\ von\ \alpha \wedge \beta_i \in N^*\Sigma\}$ definiert. Die Menge P' enthält dann:

$$S' \to A^S, \quad \forall S \to A \in P,$$
$$A^C \to \alpha^C, \quad \forall A \to \alpha \in P,$$
$$A^S \to PREF(\alpha), \quad \forall A \to \alpha \in P,$$

Durch die Konstruktion können in G' umbenennende Regeln oder Regeln mit sich wiederholenden rechten Seiten auftreten, daher ist sie nicht mehr in FNF. Nun muss gezeigt werden, dass $OPM(G) = OPM(G')$ ist. Aus der Definition der Transformationen folgt direkt für alle $A \in N$:

$$\mathcal{R}_{G'}(A^C) = \mathcal{R}_G(A), \quad \mathcal{L}_{G'}(A^C) = \mathcal{L}_G(A), \quad \mathcal{L}_{G'}(A^S) = \mathcal{L}_G(A).$$

Die einzige Menge, die sich ändern könnte wäre $R_{G'}(A^S)$. Man nimmt also an es gibt $a \in \Sigma, A \in N$ und $a \in \mathcal{R}_{G'}(A^S) \setminus \mathcal{R}_G(A)$. Somit gibt es eine Ableitung $S' \stackrel{*}{\Rightarrow} \gamma A^S \stackrel{*}{\Rightarrow} \gamma\beta a X, \quad X \in N^S \cup \{\epsilon\}$ aber keine $S' \stackrel{*}{\Rightarrow} \gamma A^C b\eta \stackrel{*}{\Rightarrow} \gamma\beta a X b\eta$. Da A^S nur auf rechten Seiten vorkommt, werden keine Relationen $a \gtrdot b$ produziert und die OPM wird nicht verändert. Zum Schluss muss noch die bewiesen werden, dass $L(G') = pref(L(G))$. Zunächst gelten folgende Implikationen:

$$\forall A^C : S' \stackrel{*}{\Rightarrow} \alpha A^C \beta, \quad wenn \beta \neq \epsilon \tag{5}$$

$$\forall A^S : S' \stackrel{*}{\Rightarrow} \alpha A^S \beta, \quad wenn \beta = \epsilon \tag{6}$$

Es wird durch Induktion auf die Länge k der Ableitungen bewiesen, dass gilt: Wenn $S \stackrel{*}{\Rightarrow} \alpha$, dann $S' \stackrel{*}{\Rightarrow} PREF(\alpha)$.
Induktionsanfang (k=2): Aufgrund der Fischernormalform haben die einfachsten Ableitungen ganz einfach die Form $S \Rightarrow A \Rightarrow \alpha, \quad \forall S \to A, A \Rightarrow \alpha$ in G. Somit folgt aus der Definition von G':$S' \Rightarrow A^S \Rightarrow PREF(\alpha), \quad \forall S' \to A^S, A^S \Rightarrow PREF(\alpha)$.
Induktionsschritt: Die Induktionsannahme gelte für Ableitungen $S \stackrel{k}{\Rightarrow} \alpha$. Jetzt sei $S \stackrel{k+1}{\Rightarrow} \beta$ eine beliebige Ableitung mit $\beta = \zeta\gamma\gamma'\eta$, sodass

$$S \stackrel{k}{\Rightarrow} \zeta A \eta \Rightarrow \zeta\gamma\gamma'\eta = \beta$$

Jetzt folgt aus der Konstruktion von G', dass $A^S \to \gamma \in P'$ und aus der Induktionsannahme

$$S' \stackrel{k}{\Rightarrow} \zeta A^S \Rightarrow \zeta\gamma \in PREF(\beta)$$

Damit ist die Induktion abgeschlossen. Weiterhin kann argumentiert werden, dass G' nur Satzformen generieren kann, die Präfixe von Satzformen aus G sind, da zwangsläufig ein A^S abgeleitet wird, dass die Berechnung des Präfix initiiert. Damit wurde gezeigt, dass $L(G') = pref(L(G))$ und der Beweis ist abgeschlossen. □

Um den Abschnitt abzuschließen, müssen noch die beiden letzten Spracheigenschaften betrachtet werden: Die Konkatenation und der Kleene-*. Allerdings ist die Konstruktion der Grammatiken für diese beiden generierten Sprachen sehr umfangreich und der Beweis äußerst aufwendig. Daher wird an dieser Stelle nur auf den Anhang von [3] verwiesen, in dem diese aufgeführt sind.

Satz 3.5 (Konkatenation)**.** *Seien G_1, G_2 präzedenzkompatible OPGs. Dann kann eine OPG G effektiv konstruiert werden, sodass*

$$L(G) = L(G_1) \bullet L(G_2) \ \wedge \ OPM(G) \supseteq OPM(G_1) \cup OPM(G_2)$$

Satz 3.6 (Kleene-*)**.** *Sei $G = (N, \Sigma, P, S)$ eine OPG mit einer $\doteq$-azyklischen OPM. Dann kann eine OPG $\hat{G} = (\hat{N}, \Sigma, \hat{P}, \hat{S})$ mit $OPM(\hat{G}) \supseteq OPM(G)$ effektiv konstruiert werden, sodass $L(\hat{G}) = (L(G))^*$.*

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

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

Nachdem D. Knuth die LR(k)-Grammatiken eingeführt hat, 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 M, in der $\forall a, b \in \Sigma : M_{ab} = \lessdot$*

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 3.5 (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:* $\delta(p, a, \perp) \ni (q, \perp), \quad a \in \Sigma_r$.
  Kurzform: $\delta(p, a, \perp) \ni q$
- *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 werden 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])

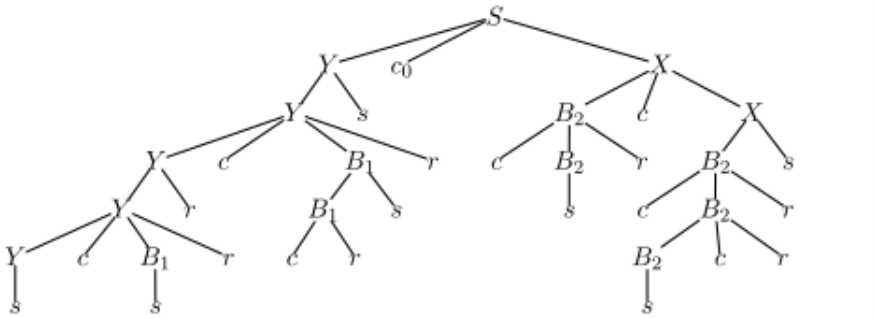

Abbildung 3: Schema des Syntaxbaumes

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_0z einzulesen. Es wird eine Menge Q_p von Zuständen disjunkt von $Q_q \cup \{q_0\}$ definiert. Der Automat führt eine Push-Transition $\delta(r, c_0) \ni (r', Z_U)$ 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 Abbildung 3 konstruiert. Z.B. steht dabei das äußere linke s in Abbildung 3 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 Abbildung 3 dargestellt, unterschieden werden. Diese Fälle sind in Tabelle 3 dargestellt. Die so produzierte Grammatik ist nicht zwangsläufig reduziert, das heißt 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 ($\mathcal{R}_G(A)$ oder $\mathcal{L}_G(A)$) 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 von Tabelle 2. Für alle Operatorpräzedenzgrammatiken mit einer solchen Matrix ist L(G) eine Visibly Pushdown Sprache [3].

□

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

Tabelle 2: Totale Matrix M_T

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 3: Übersicht der Regelgeneration

4 Implementierung der OPA

In diesem Abschnitt geht es um eine Implementierung einer Bibliothek für OPA in einer Programmiersprache. Dazu folgt eine kurze Aufzählung der verwendeten Programme und Ressourcen und anschließend folgt eine Erläuterung zu der Funktionsweise anhand eines Klassendiagramms.

Verwendete Ressourcen Die Bibliothek wurde in der Programmiersprache Java mit dem JDK 1.8 verfasst. Die Sprache Java ist sehr weit verbeitet und sehr gut dokumentiert und vereinfacht strukturiertes und methodisches Vorgehen. Als Umwickelungsumgebung wurde IntelliJ IDEA Ultimate Edition verwendet. Für die Erstellung der Bibliothek als JAR-Datei und die Verwaltung der Abhängigkeien wurde das Werkzeug Maven der Apache Software Foundation verwendet. Außer den Standardbibliotheken der JDK 1.8 wurde noch *java-tuples* [13] verwendet.

Design Für die Entwickelung wurde hauptsächlich die Definition des OPA direkt in Klassen und Attribute umgesetzt. So besteht zB. die Klasse *OP_Automat* aus einer Menge von Terminalen (Character), einer OP_Matrix, einer Menge von Zuständen, der Menge der Start- und Endzustände und einem Objekt der *Transitionen*-Klasse, was der Übergangsfunktion δ entspricht. Die Präzedenzrelation werden als Enumeration dargestellt. In Abbildung 4 ist ein vereinfachtes Klassendiagramm für das Paket model.automation dargestellt. Wie in den Quellen ist auch hier die Berechnung erst einmal vom reinen Automatenmodell getrennt. Die Berechnung findet in den Klassen des Paketes model.computation statt, welches wie in **??** beschrieben aufgebaut ist. Um nun eine Berechnung zu starten muss nunächst der Automat initialisiert werden. Dieser besteht aus vielen verschachtelten Attributen, was die Initialisierung sehr aufwendig macht. Es wurde unter anderem durch Verwendung des Builder-Patterns für den Automaten und einer ähnlichen Form für die Matrix und die Transitionen eine etwas komfortablere und intuitivere Nutzung gewährleistet. Dann muss ein Objekt der Klasse *Computation* mit einem OPA und einem Eingabestring erzeugt werden, welches dann durch die *compute* Methode die Berechnung startet und das Ergebnis in der Konsole ausgibt. Um schnelleres Testen zu gewährleisten, wurde ebenfalls eine OPA_ReaderWriter Klasse erzeugt, die es ermöglicht den Automaten zu serialisieren und somit zu speichern. Das entstandene Format ist dabei allerdings nicht leserlich. Weiterhin wurden zu allen wichtigen Komponenten *print-* Methoden implementiert.
Der Automat arbeitet nichtdeterministisch, indem er in jedem Schritt alle

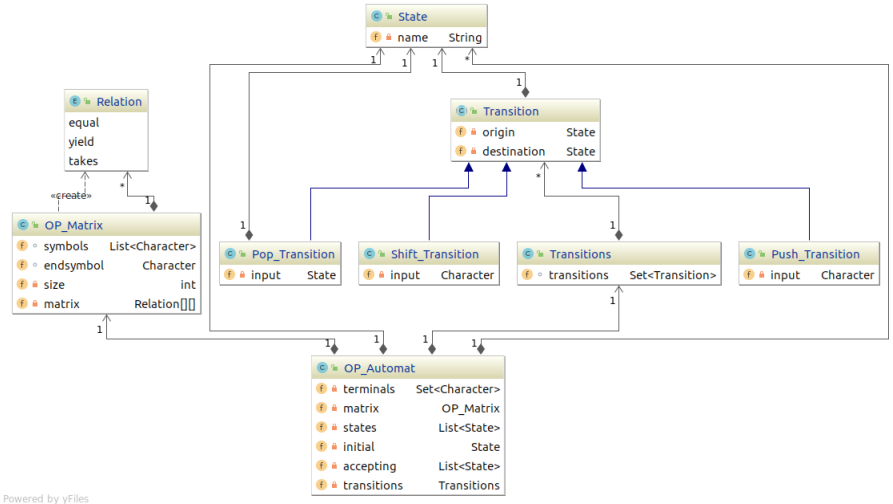

Abbildung 4: UML Klassendiagramm zum automaton package

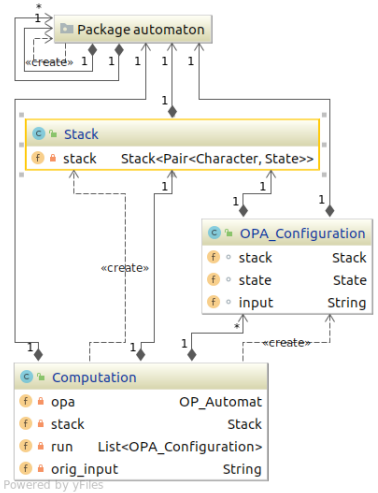

Abbildung 5: UML Klassendiagramm zum computation package

möglichen weiteren Schritte berechnet. Am Anfang wird zu jedem initialen Zustand eine Konfiguration erstellt. Es wird immer nur ein Pfad von Konfigurationen bearbeitet bis dieser akzeptiert oder fehlschlägt. An jeder Stelle werden mögliche Alternativen in einer zweiten Warteschlange gespeichert, mit der Information an welcher Stelle diese Alternative einzusetzen ist. Schlägt ein Pfad fehl, wird der Lauf bis zur jüngsten Alternative in der Warteschlange zurückgesetzt und es geht mit der Alternative weiter. Wenn eine akzeptierende Konfiguration erreicht wird, oder keine Alternativen mehr vorhanden sind bei nichtleerem Input, terminiert der Automat.
Die Implementierung ist allerdings noch sehr rudimentär und es konnten leider nicht alle Anforderungen erfüllt werden. Es wäre schön gewesen, Konstruktionen für die Abschlusseigenschaften auf Automatenebene zu haben, so konnte lediglich die Vereinigung implementiert werden.

5 Fazit

Es war sehr spannend in ein, zwar nicht neues, aber durchaus noch unterrepräsentiertes Thema der Informatik einzutauchen. Ein erneutes Interesse an diesen Sprachen halte ich auf jeden Fall für angebracht. Trotzdem war es eine große Herausforderung in ein so theoretisches Themengebiet einzusteigen und die Definitionen und Beweise zu verinnerlichen. 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, 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 dürfte genügen, damit die Wissenschaft sich noch weiter mit den Operatorpäzedenzsprachen beschäftigen wird und man darf gespannt sein, was in den nächsten Jahren noch an weiteren Ergebnissen folgt.
Damit ist die Arbeit abgeschlossen.

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, Dino Mandrioli, Federica Panella, 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.

[12] Robert McNaughton. Parenthesis grammars. *J. ACM*, 14(3):490–500, July 1967.

Plagiatserklärung der / des Studierenden

Hiermit versichere ich, dass die vorliegende Arbeit über _____ _____ 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.

_____
(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.

_____
(Datum, Unterschrift)