Operatorpräzedenzsprachen und ihre Automaten

Jonas Kremer

16. Februar 2019

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 arithmetischen Ausdrücken 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.
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.
Der Hauptaspekt hier liegt allerdings weniger auf den Grammatiken, als auf den Operatorpräzedenzautomaten (OPA), die deutlich neuer sind. 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 der regulären Sprache 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.

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 [siam 25, 38].Die folgenden Definitionen gelten für Operatorpräzedenzgrammatiken (OPGs).

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

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

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. [ASDFGH] 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 $A \to A_1 a_1 ... A_\mathrm{n} a_\mathrm{n} A_{\mathrm{n}+1}$, 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 dh. 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. 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 Operatorpräzedenzautomaten

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.

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 $p \in \delta_{\text{push}}(q, a)$. Ein gestrichelter Pfeil steht für eine Shift-Transition $p \in \delta_{\text{shift}}(q, a)$. Ein doppelter Pfeil mit einem Zustand r als Beschriftung steht dementsprechend für einen Pop-Move $p \in \delta_{\text{pop}}(q, r)$
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 $\Pi = \bot\pi_1\pi_2...pi_{\text{n}}$ mit $\pi_{\text{i}} \in \Gamma$. Die symbol-Funktion wird auf Strings erweitert, indem das Symbol des letzten Elements ausgegeben wird: $symbol(\Pi) = symbol(\pi_{\text{n}})$.
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.
Um den Automaten zu vervollständigen muss die akzeptierte Sprache definiert werden. Eine Konfiguration $(\bot, q_{\text{I}}, w\#)$ mit q_{I} *inI* heisst *initial* und eine Konfiguration $(\bot, q_{\text{F}}, \#)$ mit $q_{\text{F}} \in F$ 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 $x = a_0x_0a_1x_1a_2...a_nx_na_n$ 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 ${}^{a_i}\left[x\right]^{a_{n+1}}$ 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:*

- *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.

2.3 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.

Definition 2.13 (Deterministischer OPA)**.** *Ein OPA* $A = (\Sigma, M, Q, I, F, \delta)$ *ist deterministisch, wenn gilt:*

- *I besteht aus nur einem Element* [1]
- $\delta : Q \times \{Q \cup \Sigma\} \to Q\}$
  (δ bildet auf Q ab, anstatt auf $\mathcal{P}(Q)$)

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 2.1. *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 den Basis der Berechnungen (leerer Stack) steht. [2]
- $\tilde{I} = I \times \{\top\}$ 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:

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

[1] Hier wird nicht zwischen einem einzelnen Element und einer Menge mit nur einem Element unterschieden.

[2] Zustände in $\tilde{Q}$ werden mit K bezeichnet

$\tilde{\delta}_{shift} : \tilde{Q} \times \Sigma \to \tilde{Q}$ 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 2.2. *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 $i = 2, ..., n$ 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 2.3. *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 $K \overset{x_0}{rightsquigarrow} K'_0$, 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 2.1 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 2.3 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 2.3 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. □

2.4 Äquivalenz von OPG und OPA

3 (Abschluss-) Eigenschaften von OPLs

3.1 Determistischer vs. Nondeterministischer OPA

3.2 Abgeschlossenheit

3.3 Visibly Pushdown Sprache als Teilklasse

4 Implementation of OPLs

Anhang