

# Operatorpräzedenzsprachen und ihre Automaten

Jonas Kremer

14. Februar 2019

## Inhaltsverzeichnis

<b>1</b>	<b>Motivation</b>	<b>3</b>
<b>2</b>	<b>Vorbereitung und Definitionen</b>	<b>4</b>
2.1	Operatorpräzedenzgrammatik . . . . .	4
2.2	Operatorpräzedenzautomaten . . . . .	6
2.3	Deterministischer vs. Nichtdeterministischer OPA . . . . .	10
2.4	Äquivalenz von OPG und OPA . . . . .	12
<b>3</b>	<b>(Abschluss-) Eigenschaften von OPLs</b>	<b>11</b>
3.1	Deterministischer vs. Nondeterministischer OPA . . . . .	11
3.2	Abgeschlossenheit . . . . .	11
3.3	Visibly Pushdown Sprache als Teilklasse . . . . .	11
<b>4</b>	<b>Implementation of OPLs</b>	<b>12</b>

## 2 Vorbereitung 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,  $\Sigma$  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  $a, b, \dots$  bezeichnen einzelne Terminalsymbole; Spätere, kleine lateinische Buchstaben  $u, v, \dots$  bezeichnen Terminalstrings; Große lateinische Buchstaben  $A, B, \dots$  stehen für Nichtterminalsymbole und griechische Buchstaben  $\alpha, \beta, \dots$  für beliebige Strings über  $N \cup \Sigma$ . Sofern nicht explizit eingeschränkt können Strings auch leer sein.

Weiterhin haben Produktionsregeln die Form  $A \rightarrow \alpha$  mit der *leeren Regel*  $A \rightarrow \epsilon$ . *Umbenennende* Regeln haben nur ein Nichtterminalsymbol als rechte Seite. Eine *direkte Ableitung* wird mit  $\Rightarrow$  beschrieben, eine beliebige Anzahl von Ableitungen mit  $\Rightarrow^*$ .

Eine Grammatik heißt *reduziert*, wenn jede Regel aus  $P$  benutzt werden kann um einen String aus  $\Sigma^*$  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 2.1.1.** Für eine OG  $G$  und ein Nichtterminal  $A$  sind die linken

und rechten Terminalmengen definiert als

$$\mathcal{L}_G(A) = \{a \in \Sigma \mid A \xRightarrow{*} Ba\alpha\}, \quad \mathcal{R}_G(A) = \{a \in \Sigma \mid A \xRightarrow{*} \alpha aB\}$$

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

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

**Definition 2.1.2** (Präzedenzrelationen).

*Gleiche Präzedenz:*  $a \doteq b \Leftrightarrow \exists A \rightarrow \alpha a B b \beta, B \in N \cup \{\epsilon\}$

*Übernimmt Präzedenz:*  $a \succ b \Leftrightarrow \exists A \rightarrow \alpha D b \beta, D \in N \text{ and } a \in \mathcal{R}_G(D)$

*Gibt Präzedenz ab:*  $a \lessdot b \Leftrightarrow \exists A \rightarrow \alpha a D \beta, D \in N \text{ and } b \in \mathcal{L}_G(D)$

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  $|\Sigma| \times |\Sigma|$  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 2.1.3.** 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  $\forall a, b : M_{ab} \neq \emptyset$ . Für eine OPG wird die *Fischer Normalform (FNF)* definiert.

**Definition 2.1.4** (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:  $\forall a \in \Sigma : \# \prec a$  und  $a \succ \#$ . Ferner gilt:  $M_{\# \#} = \{\dot{=}\}$ .

**Definition 2.1.5.** *Ein OP Alphabet ist ein Paar  $(\Sigma, M)$  mit:*

*$\Sigma$  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  $\dot{=}$ -Relation getroffen werden. Aus 2.1.2 folgt, dass bei einer Regel  $A \rightarrow A_1 a_1 \dots A_n a_n A_{n+1}$ , bei der alle  $A_i$  potenziell fehlen können, die Relationen  $a_1 \dot{=} a_1 \dot{=} \dots \dot{=} a_n$  gebildet werden. Problematisch wird dies, wenn die  $\dot{=}$ -Relation *zyklisch* ist dh. es gibt  $a_1, a_2, \dots, a_m \in \Sigma (m \geq 1)$ , sodass  $a_1 \dot{=} a_2 \dot{=} \dots \dot{=} a_m \dot{=} a_1$ . 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  $\dot{=}$ -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 2.2.1** (Operatorpräzedenzautomat). *Ein nichtdeterministischer OPA ist ein 6-Tupel  $A = (\Sigma, M, Q, I, F, \delta)$  mit*

- $(\Sigma, M)$  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
- $\delta : Q \times (\Sigma \cup Q) \rightarrow \mathcal{P}(Q)$  ist die Übergangsfunktion, die aus drei Teilfunktionen besteht:  
 $\delta_{shift} : Q \times \Sigma \rightarrow \mathcal{P}(Q)$   $\delta_{push} : Q \times \Sigma \rightarrow \mathcal{P}(Q)$   $\delta_{pop} : Q \times Q \rightarrow \mathcal{P}(Q)$

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  $p \in \delta_{push}(q, a)$ . Ein gestrichelter Pfeil steht für eine Shift-Transition  $p \in \delta_{shift}(q, a)$ . Ein doppelter Pfeil mit einem Zustand  $r$  als Beschriftung steht dementsprechend für einen Pop-Move  $p \in \delta_{pop}(q, r)$ .

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  $\Gamma' = \Gamma \cup \{\perp\}$ . Elemente von  $\Gamma'$  haben die Form  $[a, q]$  oder  $\perp$ .

Für  $\Gamma'$  werden ein paar Hilfsfunktionen eingeführt:  $symbol([a, q]) = a$ ,  $symbol(\perp) = \#$  und  $state([a, q]) = q$ . Der Stack ist ein String der Form  $\Pi = \perp \pi_1 \pi_2 \dots \pi_n$  mit  $\pi_i \in \Gamma$ . Die symbol-Funktion wird auf Strings erweitert, indem das Symbol des letzten Elements ausgegeben wird:  $symbol(\Pi) = symbol(\pi_n)$ .

Eine *Konfiguration* eines OPA ist ein Tripel  $C = (\Pi, q, w)$ , wobei  $\Pi \in \Gamma^*$  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 2.2.2** (Übergangsfunktionen). *Die drei verschiedenen Moves sind wie folgt definiert:*

*push move: if  $\text{symbol}(\Pi) \leq a$  then  $(\Pi, p, ax) \vdash (\Pi[a, p], q, x)$  mit  $q \in \delta_{\text{push}}(p, a)$*

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

*pop move: if  $a \succ b$  then  $(\Pi[a, p], q, bx) \vdash (\Pi, r, bx)$  mit  $r \in \delta_{\text{pop}}(q, p)$*

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  $(\perp, q_I, w\#)$  mit  $q_I \in I$  heisst *initial* und eine Konfiguration  $(\perp, q_F, \#)$  mit  $q_F \in F$  heisst akzeptierend.

**Definition 2.2.3** (Akzeptanz der Sprache). *Die Sprache, die von einem OPA  $A$  erkannt wird ist definiert als*

$$L(A) = \left\{ w \mid (\perp, q_I, w\#) \vdash^* (\perp, q_F, \#), q_I \in I, q_F \in F \right\}$$

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 2.2.4** (Einfache Ketten). *Sei  $(\Sigma, M)$  ein Präzedenz-Alphabet. Eine einfache Kette ist ein Wort  $a_0 a_1 a_2 \dots a_n a_{n+1}$ , welches wir als  ${}^{a_0} [a_1 a_2 \dots a_n]^{a_{n+1}}$  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  $a_0 \leq a_1 \doteq a_2 \dots a_n \succ a_{n+1}$*

**Definition 2.2.5** (Zusammengesetzte Ketten). *Sei  $(\Sigma, M)$  ein Präzedenz-Alphabet. Eine zusammengesetzte Kette ist ein Wort  $a_0 x_0 a_1 x_1 a_2 \dots a_n x_n a_{n+1}$  mit  $x_i \in \Sigma^*$ , bei dem  ${}^{a_0} [a_1 a_2 \dots a_n]^{a_{n+1}}$  eine einfache Kette ist und jedes  $x_i$  entweder leer( $\epsilon$ ) oder wieder eine (einfache oder zusammengesetzte) Kette ist.*

*Die Schreibweise ist  ${}^{a_0} [x_0 a_1 x_1 a_2 \dots a_n x_n]^{a_{n+1}}$*

Für diese Strukturen benennt man weitere Eigenschaften. Bei einer Kette  ${}^a[x]^b$  wird  $x$  der *Rumpf* genannt. Die *Tiefe*  $d(x)$  einer Kette wird rekursiv definiert:  $d(x) = 1$ , wenn  $x$  eine einfache Kette ist und  $d(x_0 a_1 x_1 \dots a_n x_n) = \max_i (d(x_i)) + 1$  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 Kompatibilität eines Wortes  $w$  mit einer OPM  $M$  definieren, die allerdings nicht mit der Akzeptanz für einen OPA verwechselt werden darf.

**Definition 2.2.6** (Kompatibilität mit OPM). *Ein Wort  $w$  über  $(\Sigma, M)$  ist kompatibel mit  $M$ , wenn die beiden Bedingungen gelten:*

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

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

**Definition 2.2.7** (Stützen). *Sei  $A$  ein OPA. Eine Stütze für eine einfache Kette  ${}^a_0[a_1 a_2 \dots a_n]^{a_{n+1}}$  ist ein beliebiger Pfad in  $A$  der Form*

$$q_0 \xrightarrow{a_1} q_1 \dashrightarrow \dots \dashrightarrow q_{n-1} \dashrightarrow^{a_n} q_n \xrightarrow{q_0} q_{n+1} \quad (1)$$

*Eine Stütze für eine zusammengesetzte Kette  ${}^{a_0}_0[x_0 a_1 x_1 a_2 \dots a_n x_n]^{a_{n+1}}$  ist ein beliebiger Pfad in  $A$  der Form*

$$q_0 \xrightarrow{x_0} q'_0 \xrightarrow{a_1} q_1 \xrightarrow{x_1} q'_1 \dashrightarrow \dots \dashrightarrow q_{n-1} \dashrightarrow^{a_n} q_n \xrightarrow{x_n} q'_n \xrightarrow{q'_0} q_{n+1} \quad (2)$$

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



- if  $x_i \neq \epsilon$ , dann ist  $q_i \xrightarrow{x_i} q'_i$  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  $q_I \xrightarrow{w} q_F$  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. Nichtdeterministischer 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 2.3.1** (Deterministischer OPA). *Ein OPA  $A = (\Sigma, M, Q, I, F, \delta)$  ist deterministisch, wenn gilt:*

- $I$  besteht aus nur einem Element <sup>1</sup>
- $\delta : Q \times \{Q \cup \Sigma\} \rightarrow Q$   
( $\delta$  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

---

<sup>1</sup>Hier wird nicht zwischen einem einzelnen Element und einer Menge mit nur einem Element unterschieden.

seit dem Push-Move, also dem Beginn einer Kette, genommen hat, gesammelt. Dazu wird jeder Zustand des deterministischen Automaten als Menge von Zustandspaaren definiert. 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 nichtdeterministischen 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 deterministische 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.<sup>2</sup>
- $\tilde{I} = I \times \{\top\}$  ist der Startzustand
- $\tilde{F} = \{K \mid K \cap (F \times \{\top\}) \neq \emptyset\}$
- $\tilde{\delta} : \tilde{Q} \times (\Sigma \cup \tilde{Q}) \rightarrow \tilde{Q}$  ist die Übergangsfunktion ist die Vereinigung aus den drei Teilfunktionen:

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

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

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

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

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

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

---

<sup>2</sup>Zustände in  $\tilde{Q}$  werden mit K bezeichnet

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

Die Äquivalenz der beiden Automaten basiert auf den Aussagen der Lemmata:

**Lemma 2.2.** *Sei  $y$  der Rumpf einer Kette mit der Stütze  $q \xrightarrow{y} q'$  in  $A$ . Dann gilt für alle  $p \in Q$  und  $K \in \tilde{Q}$ , wenn  $(q, p) \in K$ , dann existiert eine Stütze  $K \xrightarrow{y} K'$  mit  $(q', p) \in K'$*

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

**Induktionsanfang:  $h = 1$**  ANVSKJCKJSNVKJSNDVKLalksmvmlmss

□

□

## 2.4 Äquivalenz von OPG und OPA

## 4 Implementation of OPLs

## Anhang