WWU
MÜNSTER

Operator Präzedenz Sprachen

und ihre Automaten

wissen.leben
WWU Münster

Jonas Kremer

5. Februar 2019

Agenda

1. Einleitung
2. Vorbereitung
3. Operator Präzedenz Grammatik
4. Operator Präzedenz Automaten
5. (Abschluss-) Eigenschaften von OPLs
6. Ausblick

Einleitung

1. Operator Precedence Languages (OPL) sind eine Subklasse der kontextfreien Sprachen

Einleitung

1. Operator Precedence Languages (OPL) sind eine Subklasse der kontextfreien Sprachen
2. Sie wurden 1963 von Robert W. Floyd vorgestellt

Einleitung

1. Operator Precedence Languages (OPL) sind eine Subklasse der kontextfreien Sprachen
2. Sie wurden 1963 von Robert W. Floyd vorgestellt
3. Größte bekannte Klasse von kontextfreien Sprache, die Abschlusseigenschaften von regulären Sprachen hat

Motivation

1. Als Beispiel dient eine einfache Sprache für arithmetische Ausdrücke
   zB. $n + n \times (n + n)$

Motivation

1. Als Beispiel dient eine einfache Sprache für arithmetische Ausdrücke
   zB. $n + n \times (n + n)$
2. Präzedenz der Multiplikation über die Addition

Motivation

1. Als Beispiel dient eine einfache Sprache für arithmetische Ausdrücke
   zB. $n + n \times (n + n)$
2. Präzedenz der Multiplikation über die Addition
3. Parsing basiert auf Operator Präzedenz Relationen, wodurch lokales Parsing und erhöhte Parallelisierung möglich ist

Definition kontextfreie Grammatik

- Eine kontextfreie Grammatik ist ein 4-Tupel $G = (N, \Sigma, P, S)$

Definition kontextfreie Grammatik

- Eine kontextfreie Grammatik ist ein 4-Tupel $G = (N, \Sigma, P, S)$
  1. N ist die Menge der Nichtterminale

Definition kontextfreie Grammatik

- Eine kontextfreie Grammatik ist ein 4-Tupel $G = (N, \Sigma, P, S)$
  1. N ist die Menge der Nichtterminale
  2. Σ ist die Menge der Terminalsymbole

Definition kontextfreie Grammatik

- Eine kontextfreie Grammatik ist ein 4-Tupel $G = (N, \Sigma, P, S)$
  1. N ist die Menge der Nichtterminale
  2. Σ ist die Menge der Terminalsymbole
  3. P sind die Produktionsregeln

Definition kontextfreie Grammatik

- Eine kontextfreie Grammatik ist ein 4-Tupel $G = (N, \Sigma, P, S)$
  1. N ist die Menge der Nichtterminale
  2. Σ ist die Menge der Terminalsymbole
  3. P sind die Produktionsregeln
  4. $S \in N$ ist das Startsymbol

Definition kontextfreie Grammatik

- Eine kontextfreie Grammatik ist ein 4-Tupel $G = (N, \Sigma, P, S)$
  1. N ist die Menge der Nichtterminale
  2. Σ ist die Menge der Terminalsymbole
  3. P sind die Produktionsregeln
  4. S $\in$ N ist das Startsymbol
- Produktionen haben die Form $A \rightarrow \beta$, wobei $\beta \in (\Sigma \cup N) \cup \epsilon$

Definition kontextfreie Grammatik

- Eine kontextfreie Grammatik ist ein 4-Tupel $G = (N, \Sigma, P, S)$
  1. N ist die Menge der Nichtterminale
  2. Σ ist die Menge der Terminalsymbole
  3. P sind die Produktionsregeln
  4. S $\in$ N ist das Startsymbol
- Produktionen haben die Form $A \rightarrow \beta$, wobei $\beta \in (\Sigma \cup N) \cup \epsilon$
- Ableitungen werden mit $\Rightarrow$ bzw. $\overset{*}{\Rightarrow}$ beschrieben

Definition kontextfreie Grammatik

- Eine kontextfreie Grammatik ist ein 4-Tupel $G = (N, \Sigma, P, S)$
  1. N ist die Menge der Nichtterminale
  2. Σ ist die Menge der Terminalsymbole
  3. P sind die Produktionsregeln
  4. S $\in$ N ist das Startsymbol
- Produktionen haben die Form $A \rightarrow \beta$, wobei $\beta \in (\Sigma \cup N) \cup \epsilon$
- Ableitungen werden mit $\Rightarrow$ bzw. $\overset{*}{\Rightarrow}$ beschrieben
- Definitionen und Namenskonventionen

Namenskonventionen 1

1. $a, b, \cdots \in \Sigma$ sind einzelne Terminalsymbole
2. $u, v, \cdots \in \Sigma^*$ sind beliebige Terminalstrings
3. $A, B, \cdots \in N$ sind einzelne Nichtterminale
4. $\alpha, \beta, \cdots \in (\Sigma \cup N)^*$ sind beliebige Reststrings
5. $A \to \epsilon$ ist die *leere Regel*
6. Eine *umbenennende* Regel hat nur ein Nichtterminal als rechte Seite ($A \to B$)

Namenskonventionen 2

- Eine Grammatik ist *reduziert*, wenn jede Regel benutzt werden kann um ein Wort aus Σ^* zu erzeugen

Namenskonventionen 2

- Eine Grammatik ist *reduziert*, wenn jede Regel benutzt werden kann um ein Wort aus Σ^* zu erzeugen
- Eine. Grammatik ist *invertierbar*, wenn es keine identischen rechten Seiten von Produktionsregeln gibt

Namenskonventionen 2

- Eine Grammatik ist *reduziert*, wenn jede Regel benutzt werden kann um ein Wort aus Σ^* zu erzeugen
- Eine. Grammatik ist *invertierbar*, wenn es keine identischen rechten Seiten von Produktionsregeln gibt
- Eine Regel ist in *Operatorform*, wenn die rechte Seite keine benachbarten Nichtterminale hat

Namenskonventionen 2

- Eine Grammatik ist *reduziert*, wenn jede Regel benutzt werden kann um ein Wort aus Σ^* zu erzeugen
- Eine. Grammatik ist *invertierbar*, wenn es keine identischen rechten Seiten von Produktionsregeln gibt
- Eine Regel ist in *Operatorform*, wenn die rechte Seite keine benachbarten Nichtterminale hat
- Jede kontextfreie Grammatik kann in eine äquivalente *Operatorgrammatik (OG)* umgewandelt werden

Bsp. Operator Grammatik

- $N = \{E, T, F\}$
  E ist das Startsymbol
  $\Sigma = \{+, \times, n, (,)\}$

Bsp. Operator Grammatik

- $N = \{E, T, F\}$
  E ist das Startsymbol
  $\Sigma = \{+, \times, n, (,)\}$
- Produktionsregeln:
  $E \to E + T | T$
  $T \times F | F$
  $F \to n | (E)$

- Definition: *Linke und Rechte Terminalmenge*
  $\mathcal{L}_G(A) = \{a \in \Sigma | A \overset{*}{\Rightarrow} Ba\alpha\}$
  $\mathcal{R}_G(A) = \{a \in \Sigma | A \overset{*}{\Rightarrow} \alpha aB\}$

- Definition: *Linke und Rechte Terminalmenge*
  $\mathcal{L}_G(A) = \{a \in \Sigma | A \overset{*}{\Rightarrow} Ba\alpha\}$
  $\mathcal{R}_G(A) = \{a \in \Sigma | A \overset{*}{\Rightarrow} \alpha aB\}$
- Es werden drei binäre Operator Präzedenz Relationen definiert:

- Definition: *Linke und Rechte Terminalmenge*
  $\mathcal{L}_G(A) = \{a \in \Sigma | A \overset{*}{\Rightarrow} Ba\alpha\}$
  $\mathcal{R}_G(A) = \{a \in \Sigma | A \overset{*}{\Rightarrow} \alpha aB\}$
- Es werden drei binäre Operator Präzedenz Relationen definiert:
- equal in precedence: $a \doteq b \Leftrightarrow \exists A \to \alpha aBb\beta, B \in N \cup \{\epsilon\}$
  takes precedence: $a \gtrdot b \Leftrightarrow \exists A \to \alpha Db\beta, D \in N$ and $a \in \mathcal{R}_G(D)$
  yields precedence: $a \lessdot b \Leftrightarrow \exists A \to \alpha aD\beta, D \in N$ and $b \in \mathcal{L}_G(D)$

- Definition: *Linke und Rechte Terminalmenge*
  $\mathcal{L}_G(A) = \{a \in \Sigma | A \overset{*}{\Rightarrow} Ba\alpha\}$
  $\mathcal{R}_G(A) = \{a \in \Sigma | A \overset{*}{\Rightarrow} \alpha aB\}$
- Es werden drei binäre Operator Präzedenz Relationen definiert:
- equal in precedence: $a \doteq b \Leftrightarrow \exists A \to \alpha aBb\beta, B \in N \cup \{\epsilon\}$
  takes precedence: $a \gtrdot b \Leftrightarrow \exists A \to \alpha Db\beta, D \in N$ and $a \in \mathcal{R}_G(D)$
  yields precedence: $a \lessdot b \Leftrightarrow \exists A \to \alpha aD\beta, D \in N$ and $b \in \mathcal{L}_G(D)$
- Eine Operator Präzedenz Matrix (OPM) M ist eine $|\Sigma| \times |\Sigma|$ Matrix, die für jedes Paar (a,b) die Precedence Relation speichert

- Definition: *Linke und Rechte Terminalmenge*
  $\mathcal{L}_G(A) = \{a \in \Sigma | A \overset{*}{\Rightarrow} Ba\alpha\}$
  $\mathcal{R}_G(A) = \{a \in \Sigma | A \overset{*}{\Rightarrow} \alpha aB\}$
- Es werden drei binäre Operator Präzedenz Relationen definiert:
- equal in precedence: $a \doteq b \Leftrightarrow \exists A \rightarrow \alpha aBb\beta, B \in N \cup \{\epsilon\}$
  takes precedence: $a \gtrdot b \Leftrightarrow \exists A \rightarrow \alpha Db\beta, D \in N$ and $a \in \mathcal{R}_G(D)$
  yields precedence: $a \lessdot b \Leftrightarrow \exists A \rightarrow \alpha aD\beta, D \in N$ and $b \in \mathcal{L}_G(D)$
- Eine Operator Präzedenz Matrix (OPM) M ist eine $|\Sigma| \times |\Sigma|$ Matrix, die für jedes Paar (a,b) die Precedence Relation speichert
- Eine Operator Grammatik ist eine Operator Präzedenz Grammatik, wenn M=OPM(G) *konfliktfrei* ist

Beispiel

- $\mathcal{L}(E) = \{+, \times, n, (\}, \mathcal{L}(T) = \{\times, n, (\}, \mathcal{L}(F) = \{n, (\}$
  $\mathcal{R}(E) = \{+, \times, n,)\}, \mathcal{R}(T) = \{\times, n,)\}, \mathcal{R}(F) = \{n,)\}$

Beispiel

- $\mathcal{L}(E) = \{+, \times, n, (\}, \mathcal{L}(T) = \{\times, n, (\}, \mathcal{L}(F) = \{n, (\}$
  $\mathcal{R}(E) = \{+, \times, n,)\}, \mathcal{R}(T) = \{\times, n,)\}, \mathcal{R}(F) = \{n,)\}$
- OPM(G):

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

Beispiel

- $\mathcal{L}(E) = \{+, \times, n, (\}, \mathcal{L}(T) = \{\times, n, (\}, \mathcal{L}(F) = \{n, (\}$
  $\mathcal{R}(E) = \{+, \times, n,)\}, \mathcal{R}(T) = \{\times, n,)\}, \mathcal{R}(F) = \{n,)\}$
- OPM(G):

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

- $(\lessdot \times \Leftrightarrow \exists A \to \alpha(D\beta D \in N \text{ and } \times \in \mathcal{L}_G(D)$

Beispiel

- $\mathcal{L}(E) = \{+, \times, n, (\}, \mathcal{L}(T) = \{\times, n, (\}, \mathcal{L}(F) = \{n, (\}$
  $\mathcal{R}(E) = \{+, \times, n,)\}, \mathcal{R}(T) = \{\times, n,)\}, \mathcal{R}(F) = \{n,)\}$
- OPM(G):

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

- $(\lessdot \times \Leftrightarrow \exists A \to \alpha(D\beta D \in N \text{ and } \times \in \mathcal{L}_G(D)$
- $n \gtrdot + \Leftrightarrow \exists A \to \alpha D + \beta, D \in N \text{ and } n \in \mathcal{R}_G(D)$

Beispiel

- $\mathcal{L}(E) = \{+, \times, n, (\}, \mathcal{L}(T) = \{\times, n, (\}, \mathcal{L}(F) = \{n, (\}$
  $\mathcal{R}(E) = \{+, \times, n,)\}, \mathcal{R}(T) = \{\times, n,)\}, \mathcal{R}(F) = \{n,)\}$
- OPM(G):

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

- $(\lessdot \times \Leftrightarrow \exists A \to \alpha(D\beta D \in N \text{ and } \times \in \mathcal{L}_G(D)$
- $n \gtrdot + \Leftrightarrow \exists A \to \alpha D + \beta, D \in N \text{ and } n \in \mathcal{R}_G(D)$
- $(\doteq) \Leftrightarrow \exists A \to \alpha(B)\beta, B \in N \cup \{\epsilon\}$

- Eine OPG ist in *Fischer Normalform*, wenn sie invertierbar ist und keine leeren (ausser Startsymbol) oder umbenennenden Regeln hat

- Eine OPG ist in *Fischer Normalform*, wenn sie invertierbar ist und keine leeren (ausser Startsymbol) oder umbenennenden Regeln hat
- Zusätzliches Symbol $\# \notin \Sigma$ um das Ende eines Strings zu markieren
  Alle anderen Symbole übernehmen Precedence über $\#$

- Eine OPG ist in *Fischer Normalform*, wenn sie invertierbar ist und keine leeren (ausser Startsymbol) oder umbenennenden Regeln hat
- Zusätzliches Symbol $\# \notin \Sigma$ um das Ende eines Strings zu markieren
  Alle anderen Symbole übernehmen Precedence über $\#$
- Ein Operator Präzedenz Alphabet ist ein Paar (Σ, M) mit der konfliktfreien OPM $M = |\Sigma \cup \{\#\}|^2$

- Ein nichtdeterministischer Operator Precedence Automat (OPA) ist ein 6-Tupel A = $(\Sigma, M, Q, I, F, \delta)$

- Ein nichtdeterministischer Operator Precedence Automat (OPA) ist ein 6-Tupel A = $(\Sigma, M, Q, I, F, \delta)$
  1. (Σ, M) ist ein OP Alphabet

- Ein nichtdetermistischer Operator Precedence Automat (OPA) ist ein 6-Tupel A = $(\Sigma, M, Q, I, F, \delta)$
  1. (Σ, M) ist ein OP Alphabet
  2. Q ist die Menge der Zustände

- Ein nichtdetermistischer Operator Precedence Automat (OPA) ist ein 6-Tupel A = $(\Sigma, M, Q, I, F, \delta)$
  1. (Σ, M) ist ein OP Alphabet
  2. Q ist die Menge der Zustände
  3. $I \subseteq Q$ ist die Menge der Startzustände

- Ein nichtdetermistischer Operator Precedence Automat (OPA) ist ein 6-Tupel A = $(\Sigma, M, Q, I, F, \delta)$
  1. (Σ, M) ist ein OP Alphabet
  2. Q ist die Menge der Zustände
  3. $I \subseteq Q$ ist die Menge der Startzustände
  4. $F \subseteq Q$ ist die Menge der finalen Zustände

- Ein nichtdeterministischer Operator Precedence Automat (OPA) ist ein 6-Tupel A = $(\Sigma, M, Q, I, F, \delta)$
  1. (Σ, M) ist ein OP Alphabet
  2. Q ist die Menge der Zustände
  3. $I \subseteq Q$ ist die Menge der Startzustände
  4. $F \subseteq Q$ ist die Menge der finalen Zustände
  5. δ ist die Übergangsfunktion, die aus drei Teilen besteht:
     $\delta_{\text{shift}} : Q \times \Sigma \to \mathcal{P}(Q)$ $\delta_{\text{push}} : Q \times \Sigma \to \mathcal{P}(Q)$ $\delta_{\text{pop}} : Q \times Q \to \mathcal{P}(Q)$

- Ein nichtdeterministischer Operator Precedence Automat (OPA) ist ein 6-Tupel A = $(\Sigma, M, Q, I, F, \delta)$
  1. (Σ, M) ist ein OP Alphabet
  2. Q ist die Menge der Zustände
  3. $I \subseteq Q$ ist die Menge der Startzustände
  4. $F \subseteq Q$ ist die Menge der finalen Zustände
  5. δ ist die Übergangsfunktion, die aus drei Teilen besteht:
     $\delta_{\text{shift}} : Q \times \Sigma \to \mathcal{P}(Q)$ $\delta_{\text{push}} : Q \times \Sigma \to \mathcal{P}(Q)$ $\delta_{\text{pop}} : Q \times Q \to \mathcal{P}(Q)$
- Weiterhin wird das Stackalphabet definiert als $\Gamma = (\Sigma \times Q)$ mit $[a, q] \in \Gamma$ und dem Symbol für den leeren Stack $\perp$

- Ein nichtdeterministischer Operator Precedence Automat (OPA) ist ein 6-Tupel A = $(\Sigma, M, Q, I, F, \delta)$
  1. (Σ, M) ist ein OP Alphabet
  2. Q ist die Menge der Zustände
  3. $I \subseteq Q$ ist die Menge der Startzustände
  4. $F \subseteq Q$ ist die Menge der finalen Zustände
  5. δ ist die Übergangsfunktion, die aus drei Teilen besteht:
     $\delta_{\text{shift}} : Q \times \Sigma \to \mathcal{P}(Q)$ $\delta_{\text{push}} : Q \times \Sigma \to \mathcal{P}(Q)$ $\delta_{\text{pop}} : Q \times Q \to \mathcal{P}(Q)$
- Weiterhin wird das Stackalphabet definiert als $\Gamma = (\Sigma \times Q)$ mit $[a, q] \in \Gamma$ und dem Symbol für den leeren Stack $\perp$
- Der Stack Π ist ein String $\perp \Gamma^*$
  Bsp: $\perp [+, q_1] [n, q_0]$

- 1 Eine *Konfiguration* eines OPA ist ein Tripel $C = (\Pi, q, w)$ mit dem Stack Π, dem aktuellen Zustand q und der Eingabe w

- 1 Eine *Konfiguration* eines OPA ist ein Tripel $C = (\Pi, q, w)$ mit dem Stack Π, dem aktuellen Zustand q und der Eingabe w
- Eine Berechnung des Automaten ist eine endliche Folge von *Transitionen (Moves)* $C_1 \vdash C_2$

- 1 Eine *Konfiguration* eines OPA ist ein Tripel $C = (\Pi, q, w)$ mit dem Stack Π, dem aktuellen Zustand q und der Eingabe w
- Eine Berechnung des Automaten ist eine endliche Folge von *Transitionen (Moves)* $C_1 \vdash C_2$
- Die Sprache die ein OPA A akzeptiert wird definiert als: L(A)= $\{x | (\bot, q_I, x\#) \vdash^* (\bot, q_F, \#), q_I \in I, q_F \in F\}$

OPA Transitionen

- *push move:* (Normaler Pfeil)
  if $sym(\Pi) \lessdot a$ then $(\Pi, p, ax) \vdash (\Pi\,[a, p]\,, q, x)$ mit $q \in \delta_{\text{push}}(p, a)$

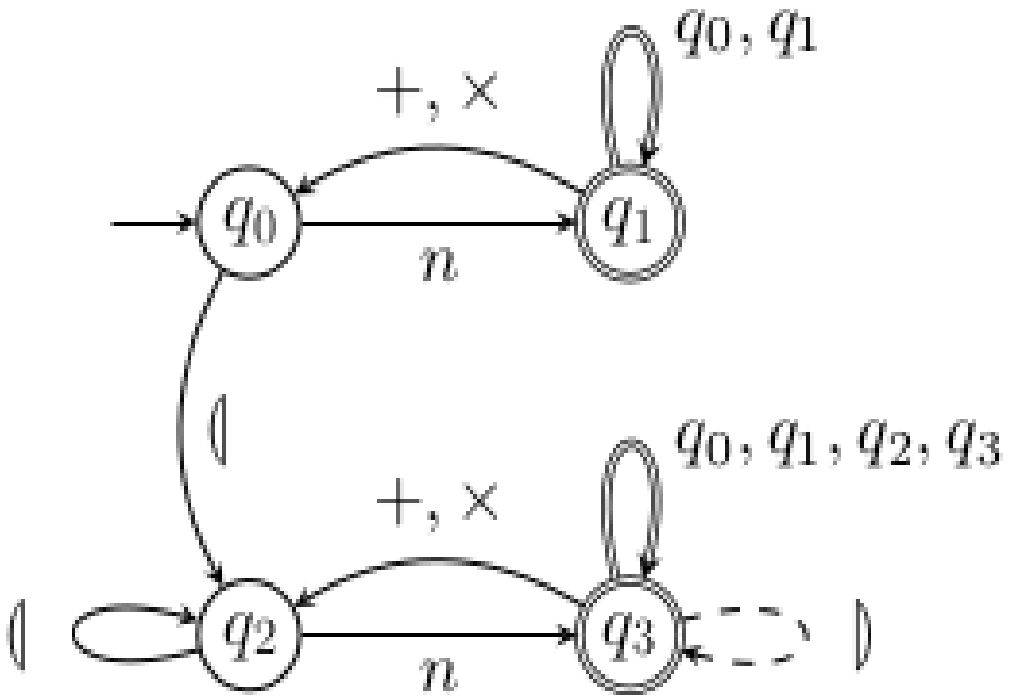
OPA Transitionen

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

OPA Transitionen

- *push move:* (Normaler Pfeil)
  if $sym(\Pi) \lessdot a$ then $(\Pi, p, ax) \vdash (\Pi\,[a, p]\,, q, x)$ mit $q \in \delta_{\text{push}}(p, a)$
- *shift move:* (Gestrichelter Pfeil)
  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:* (Doppelter Pfeil)
  if $a \gtrdot b$ then $(\Pi\,[a, p]\,, q, bx) \vdash (\Pi, r, bx)$ mit $r \in \delta_{\text{pop}}(q, p)$

Operator Präzedenz Automaten



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

OPG $\rightarrow$ OPA

- Zu jeder OPG G kann ein OPA A konstruiert werden, sodass gilt L(G)=L(A)

OPG $\rightarrow$ OPA

- Zu jeder OPG G kann ein OPA A konstruiert werden, sodass gilt L(G)=L(A)
- Idee:
  Man konstruiert einen Automaten, sodass eine erfolgreiche Berechnung „bottom-up “einen Ableitungsbaum erzeugt

OPG $\rightarrow$ OPA

- Zu jeder OPG G kann ein OPA A konstruiert werden, sodass gilt L(G)=L(A)
- Idee:
  Man konstruiert einen Automaten, sodass eine erfolgreiche Berechnung „bottom-up “einen Ableitungsbaum erzeugt
- Eine Push-Transition wird hinzugefügt, wenn das erste Terminal einer neuen rechten Seite einer Regel gelesen wird.
  Ein Shift Move bei Terminalen innerhalb einer Regel.
  Ein Pop Move am Ende einer rechten Seite.

OPG $\rightarrow$ OPA

- Zu jeder OPG G kann ein OPA A konstruiert werden, sodass gilt L(G)=L(A)
- Idee:
  Man konstruiert einen Automaten, sodass eine erfolgreiche Berechnung „bottom-up “einen Ableitungsbaum erzeugt
- Eine Push-Transition wird hinzugefügt, wenn das erste Terminal einer neuen rechten Seite einer Regel gelesen wird.
  Ein Shift Move bei Terminalen innerhalb einer Regel.
  Ein Pop Move am Ende einer rechten Seite.
- Jeder Zustand enthält das Präfix der rechten Seite einer Regel die gerade konstruiert wird und Informationen zu dem Zustand der vorher konstruiert wurde

OPG → OPA

- Zu jeder OPG G kann ein OPA A konstruiert werden, sodass gilt L(G)=L(A)
- Idee:
  Man konstruiert einen Automaten, sodass eine erfolgreiche Berechnung „bottom-up “einen Ableitungsbaum erzeugt
- Eine Push-Transition wird hinzugefügt, wenn das erste Terminal einer neuen rechten Seite einer Regel gelesen wird.
  Ein Shift Move bei Terminalen innerhalb einer Regel.
  Ein Pop Move am Ende einer rechten Seite.
- Jeder Zustand enthält das Präfix der rechten Seite einer Regel die gerade konstruiert wird und Informationen zu dem Zustand der vorher konstruiert wurde
- Sei m die Summe der Längen der rechten Seiten einer Regel. Dann hat A $O(m^2)$ Zustände

OPA $\rightarrow$ OPG

- Zu jedem OPA A kann eine OPG G konstruiert werden, sodass gilt L(A)=L(G)

OPA $\rightarrow$ OPG

- Zu jedem OPA A kann eine OPG G konstruiert werden, sodass gilt L(A)=L(G)
- Die Nichtterminale haben die Form $\Sigma \times Q \times Q \times \Sigma$

OPA $\rightarrow$ OPG

- Zu jedem OPA A kann eine OPG G konstruiert werden, sodass gilt L(A)=L(G)
- Die Nichtterminale haben die Form $\Sigma \times Q \times Q \times \Sigma$
- Idee:
  Man definiert sogenannte Supports (Transitionspfade) für einfache und zusammengesetzte Ketten von Terminalen

OPA $\rightarrow$ OPG

- Zu jedem OPA A kann eine OPG G konstruiert werden, sodass gilt L(A)=L(G)
- Die Nichtterminale haben die Form $\Sigma \times Q \times Q \times \Sigma$
- Idee:
  Man definiert sogenannte Supports (Transitionspfade) für einfache und zusammengesetzte Ketten von Terminalen
- Eine einfache Kette ist ein String von Nichtterminalen sodass gilt: $a_0 \lessdot a_1 \doteq a_2 \doteq \ldots \doteq a_n \gtrdot a_{n+1}$
  Schreibweise: ${}^{a}{}_{o}(a_1, \ldots, a_n)^{a}{}_{n+1}$

OPA $\rightarrow$ OPG

- Zu jedem OPA A kann eine OPG G konstruiert werden, sodass gilt L(A)=L(G)
- Die Nichtterminale haben die Form $\Sigma \times Q \times Q \times \Sigma$
- Idee:
  Man definiert sogenannte Supports (Transitionspfade) für einfache und zusammengesetzte Ketten von Terminalen
- Eine einfache Kette ist ein String von Nichtterminalen sodass gilt: $a_0 \lessdot a_1 \doteq a_2 \doteq \ldots \doteq a_n \gtrdot a_{n+1}$
  Schreibweise: ${}^{a}{}_{o}(a_1, \ldots, a_n)^{a}{}_{n+1}$
- Bei zusammengesetzten Ketten werden zwischen den einzelnen Terminalen einer einfachen Kette eine weitere Kette eingefügt

OPA $\rightarrow$ OPG

- Zu jedem OPA A kann eine OPG G konstruiert werden, sodass gilt L(A)=L(G)
- Die Nichtterminale haben die Form $\Sigma \times Q \times Q \times \Sigma$
- Idee:
  Man definiert sogenannte Supports (Transitionspfade) für einfache und zusammengesetzte Ketten von Terminalen
- Eine einfache Kette ist ein String von Nichtterminalen sodass gilt: $a_0 \lessdot a_1 \doteq a_2 \doteq \ldots \doteq a_n \gtrdot a_{n+1}$
  Schreibweise: ${}^{a}{}_{o}(a_1, \ldots, a_n)^{a}{}_{n+1}$
- Bei zusammengesetzten Ketten werden zwischen den einzelnen Terminalen einer einfachen Kette eine weitere Kette eingefügt
- Für jeden Support wird in einer bestimmten Weise eine Regel erzeugt

(Abschluss-) Eigenschaften von OPLs

- OPLs sind eine große Subklasse der kontextfreien Sprachen
  $L = \{a^n b a^n | n \geq 0\}$ kann nicht durch OPG dargestellt werden

(Abschluss-) Eigenschaften von OPLs

- OPLs sind eine große Subklasse der kontextfreien Sprachen
  $L = \{a^n b a^n | n \geq 0\}$ kann nicht durch OPG dargestellt werden
- Abgeschlossen unter Vereinigung, Schnitt, Komplement, Konkatenation und Kleene-*

(Abschluss-) Eigenschaften von OPLs

- OPLs sind eine große Subklasse der kontextfreien Sprachen
  $L = \{a^n b a^n | n \geq 0\}$ kann nicht durch OPG dargestellt werden
- Abgeschlossen unter Vereinigung, Schnitt, Komplement, Konkatenation und Kleene-*
- Das Leereproblem ist in PTIME lösbar, da OPLs Subklasse von kfG

Deterministischer vs Nichtdeterministischer OPA

- Ein OPA ist deterministisch, wenn $|I| = 1$ und $\delta_{\text{push}}, \delta_{\text{shift}}, \delta_{\text{pop}}$ auf Q anstatt $\mathcal{P}(Q)$ abbilden
- Zu jedem nichtdeterministischen OPA mit s Zuständen kann ein äquivalenter deterministischer OPA mit $2^{O(s^2)}$ Zuständen erzeugt werden

Ausblick

- Visibly Pushdown Sprachen als Subklasse von OPL

Ausblick

- Visibly Pushdown Sprachen als Subklasse von OPL
- Monadic Second Order Logic Characterization

Ausblick

- Visibly Pushdown Sprachen als Subklasse von OPL
- Monadic Second Order Logic Characterization
- ω-Operator Präzedenz Sprachen

Ausblick

- Visibly Pushdown Sprachen als Subklasse von OPL
- Monadic Second Order Logic Characterization
- ω-Operator Präzedenz Sprachen
- Implementierung eines Operator Präzedenz Automaten

Quellen

[1] Violetta Lonati, Dino Mandrioli, Matteo Pradella *Precedence Automata and Languages*. Milano, Italy

[2] Violetta Lonati, Dino Mandrioli, Matteo Pradella, Frederica Panella *Operator Precedence Languages: their automata-theoretic and logic characterization* [*SIAM J. Comput. 2015 Society for Industrial and Applied Mathematics*].

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