Theoretische Informatik
WS 2024/2025

Kapitel 3
Kontextfreie Sprachen

3.1
Pushdown-Automaten (PDA)

Prof. Dr. Robert Preis
Fachbereich Informatik
Fachhochschule Dortmund
Robert.Preis@fh-dortmund.de

Wozu brauchen wir mehr als reguläre Sprachen?

Typ-3 Sprachen sind einfach und effizient:

- Sie können durch Grammatiken, Automaten oder regulären Ausdrücken beschrieben werden.
- Man kann schnell und einfach überprüfen, ob ein Wort in der Sprache ist oder nicht.

Aber Programmiersprachen sind nicht regulär:

- Die meisten Programmstrukturen enthalten Schachtelungen wie Blöcke, if-then-else, arithmetische Ausdrücke, Klammerausdrücke, …
- Korrekte Klammerausdrücke und Schachtelungen sind nicht regulär.

Syntaxanalyse und Compilation von Programmiersprachen braucht mehr als reguläre Sprachen!

Typ 2: Kontextfreie Grammatiken

Nur Regeln der Form $A \to x$ ($A \in V$, $x \in \Gamma^*$), d.h.

- $A \in V$: **Links steht genau eine Variable**, nichts anderes.
- $x \in \Gamma^*$: Rechts stehen Variablen und Terminale oder das leere Wort.

Beispiele:

- $\{a^n b^n \mid n \in \mathbf{N}_+\}$:
  $G = (\{S\}, \{a, b\}, P, S)$ mit $P = \{S \to aSb \mid ab\}$
- $\{(^n)^n \mid n \in \mathbf{N}_+\}$:
  $G = (\{S\}, \{(,)\}, P, S)$ mit $P = \{S \to (S) \mid ()\}$

Die Sprache $\{(^n)^n \mid n \in \mathbf{N}_+\}$ ist somit vom Typ 2, aber nicht von Typ 3!

- Arithmetische Ausdrücke über $\mathbf{N}_0$:
  $G = (\{E, Z\}, \{0,1,2,3,4,5,6,7,8,9,+,-,*,/,(,)\}, P, E)$ mit
  $P = \{ E \to E+E \mid E-E \mid E*E \mid E/E \mid (E) \mid 0 \mid 1Z|2Z|3Z|4Z|5Z|6Z|7Z|8Z|9Z,$
  $Z \to \varepsilon|0Z|1Z|2Z|3Z|4Z|5Z|6Z|7Z|8Z|9Z\}$

Linksableitung in einer kontextfreien Grammatik

G = ({E, Z}, {0,1,2,3,4,5,6,7,8,9,+,-,*,/,(,)}, P, E) mit
P = { E → E+E | E-E | E∗E | E/E | (E) |
1Z|2Z|3Z|4Z|5Z|6Z|7Z|8Z|9Z,
Z → ε|0Z|1Z|2Z|3Z|4Z|5Z|6Z|7Z|8Z|9Z}

Ableitung des Wortes „(3+2)*(13+2)“, so dass immer die linke Variable als nächstes abgearbeitet wird:

rot: linke Variable

grün: fertiges Teilwort

schwarz: Rest, noch zu tun…

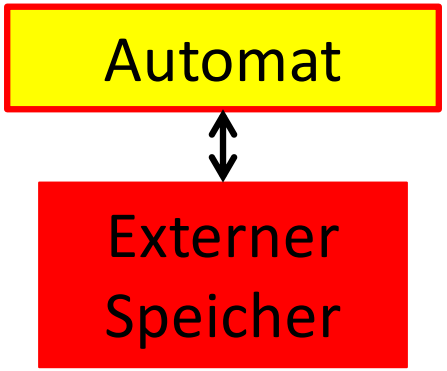
Den schwarzen Rest muss man sich merken, bzw. speichern!

E
→ E*E
→ (E)*E
→ (E+E)*E
→ (3Z+E)*E
→ (3+E)*E
→ (3+2Z)*E
→ (3+2)*E
→ (3+2)*(E)
→ (3+2)*(E+E)
→ (3+2)*(1Z+E)
→ (3+2)*(13Z+E)
→ (3+2)*(13+E)
→ (3+2)*(13+2Z)
→ (3+2)*(13+2)

Ein Maschinenmodell für Typ-2 Sprachen

Welches Maschinenmodell passt zu Typ-2 Sprachen?

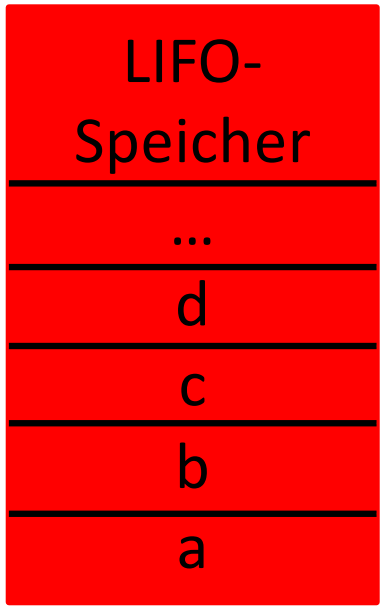
- Kontextfreie Grammatiken können $L = \{ 0^n1^n \mid n \in \mathbf{N}_+ \}$ erzeugen.
- Ohne Speicher können Automaten L nicht erkennen.
- Typ-2 Maschinenmodell benötigt externen Speicher.



Welches Speichermodell brauchen Typ-2 Sprachen?

Idee: Analysiere das Verhalten von „Links“-ableitungen!

- Links von der aktuellen Variablen A stehen nur erzeugte Terminale.
- Bei Abarbeitung von A wird ein Teilwort in den Speicher geschoben.
- Ist A komplett abgearbeitet, so muss das zuletzt gespeicherte Teilwort als nächstes abgearbeitet werden.

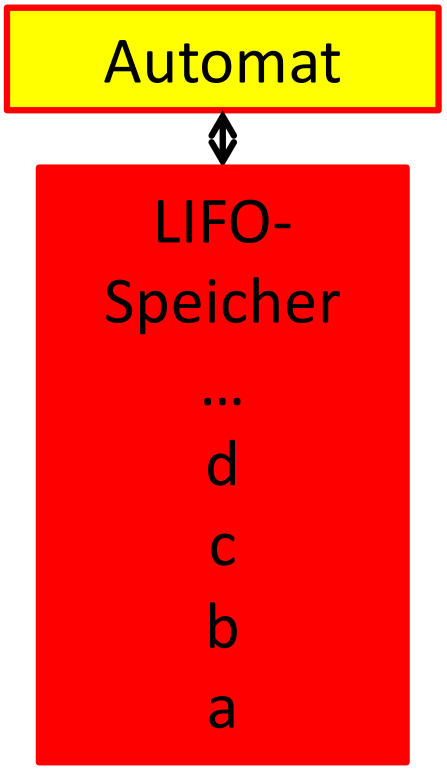


d.h. wir brauchen LIFO- Speicher (last in, first out): Stack, Keller

Pushdown-Automat / Keller-Automat intuitiv

Endlicher Automat + Stack

- Wie ein Automat
- Nur mit zusätzlichem Stack-Speicher



In jedem Schritt wird die Eingabe und der Stack gleichzeitig bearbeitet

- Zustand kann wie bisher verändert werden.
- Es wird zusätzlich zum nächsten Eingabezeichen auch das oberste Zeichen des Stacks gelesen.
- Das oberste Stacksymbol wird durch
  - kein,
  - ein oder
  - mehrere Stacksymbole ersetzt.
- Wie bisher sind nichtdeterministische und ε-Übergänge möglich.

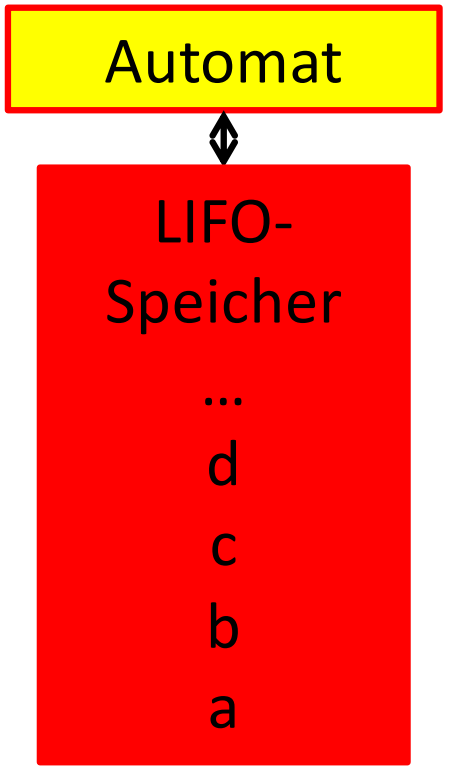
Definition eines Pushdown-Automaten

Ein Pushdown-Automat (PDA, Kellerautomat) ist ein 7-Tupel

$$P = (Q, \sum, \Gamma, \delta , q_0, Z, F)$$

mit

- Q: nichtleere endliche Zustandsmenge
- $\sum$: endliches Eingabealphabet
- Γ: endliches Stackalphabet
- δ: $Q \times (\sum \cup \{\varepsilon\}) \times \Gamma \rightarrow P(Q \times \Gamma^*)$ Überführungsfunktion
- $q_0 \in Q$: Startzustand
- $Z \in \Gamma$: Initialsymbol des Stacks
- $F \subseteq Q$: Menge von akzeptierenden Zuständen



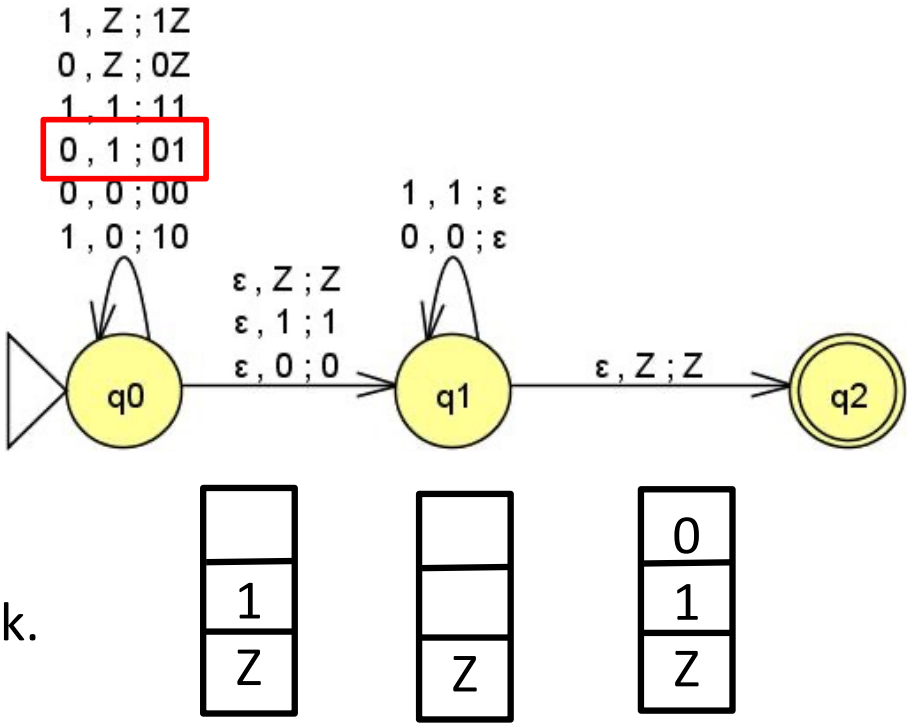
Pushdown-Automaten sind üblicherweise nichtdeterministisch!

Darstellung von Pushdown-Automaten: Übergangsdiagramm

Zustände wie beim Automaten.

Übergang „0,1/01“:

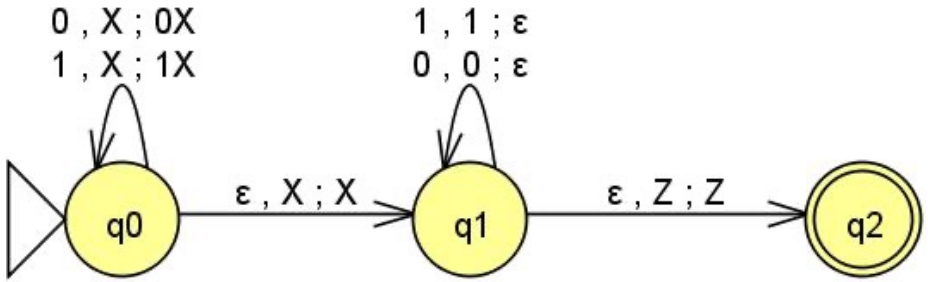
1. Lese Zeichen „0“ vom Wort
2. Lese Zeichen „1“ oben vom Stack (und lösche es vom Stack)
   Der Übergang ist nur dann anwendbar, wenn beides möglich ist !
3. Speicher das Teilwort „01“ auf den Stack.



Mehrere ähnliche Regeln können durch eine Wildcard ersetzt werden:

„0,0/00“, „0,1/01“ und „0,Z/0Z“

kann durch „0,X/0X“ ersetzt werden.



Die Wildcard darf kein Eingabe- oder Stacksymbol sein.

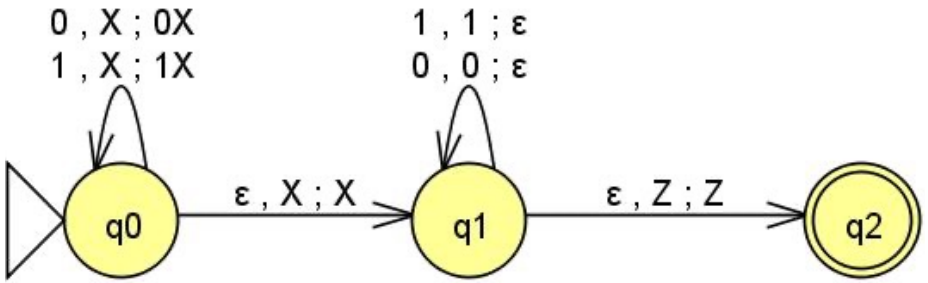
... wie sieht eine erfolgreiche Abarbeitung beim Beispiel 01000010 aus ?

Darstellung von Pushdown-Automaten: Übergangstabelle ?

Man braucht eine 3-dimensionale Tabelle, d.h. für jeden Zustand, jeden Buchstaben im Wort und jeden Buchstaben im Stack muss man einen Eintrag machen.

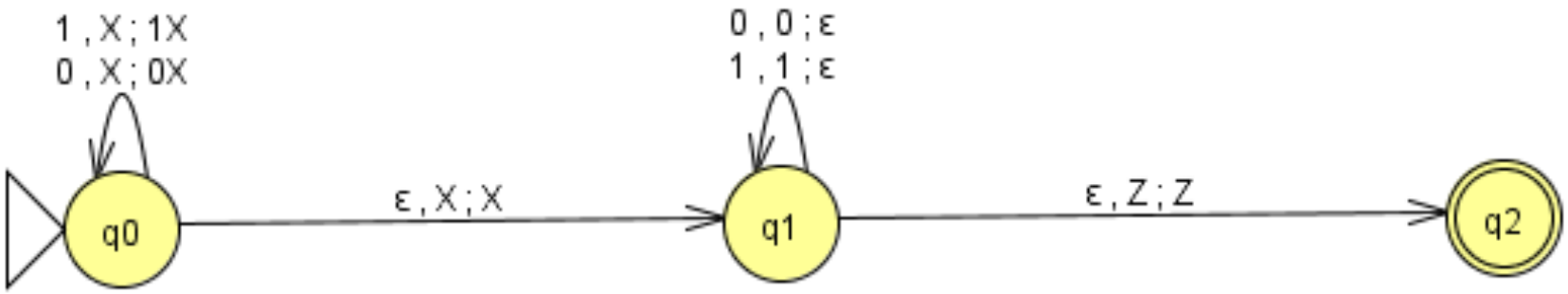
Da das unübersichtlich ist, führt man die Übergänge einzeln auf:

$\delta(q_0,0,X) = \{(q_0,0X)\}$
$\delta(q_0,1,X) = \{(q_0,1X)\}$
$\delta(q_0,\varepsilon,X) = \{(q_1,X)\}$
$\delta(q_1,1,1) = \{(q_1,\varepsilon)\}$
$\delta(q_1,0,0) = \{(q_1,\varepsilon)\}$
$\delta(q_1,\varepsilon,Z) = \{(q_2,Z)\}$



PDA für gerade Palindrome
$\{ ww^R \mid w \in \{0,1\}^* \}$

$P = (\{q_0, q_1, q_2\}, \{0,1\}, \{0,1,Z\}, \delta, q_0, Z, \{q_2\})$



Speichere w in q_0

Es wird je ein Symbol gelesen und auf den Stack gelegt.

Wort w steht dann umgekehrt im Stack.

Spontaner Wechsel „in der Mitte"

Nichtdeterministischer ε-Übergang „rät" die Mitte.

Verarbeite w^R in q_1

Jedes gelesene Symbol wird dann mit dem obersten Stacksymbol verglichen.

Es wird nur akzeptiert, wenn (nach Abarbeitung des Eingabewortes) nur noch das initiale Stacksymbol im Keller ist

Wenn im Stack keine 0 oder 1, dann in q_2 wechseln und akzeptieren.

Arbeitsweise eines PDA

Konfiguration:

Aktueller Zustand, unverarbeitete Eingabe und Inhalt des Stacks notieren.

Dargestellt als Tripel

$$(q, w, \gamma) \in Q \times \Sigma^* \times \Gamma^*$$

- q: Zustand, in dem man sich befindet
- w: Eingabe, die noch zu verarbeiten ist
- γ: Inhalt des Stacks

Konfigurationsübergang: Wechsel zwischen Konfigurationen

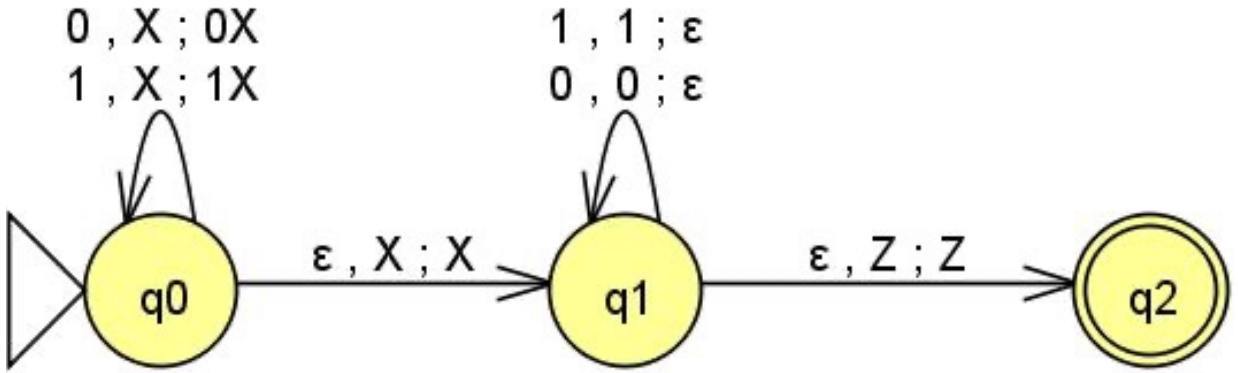
- Konfiguration (q,aw,Xβ) und
- Übergang von q nach p mit a,X/α

$$(q,aw,X\beta) \vdash (p,w,\alpha\beta)$$

- a wird abgearbeitet
- X wird durch α ersetzt

Konfigurationsübergänge des PDA für gerade Palindrome

Konfigurationsübergänge
bei dem Beispiel w = 0110:



$(q_0,0110,Z) \vdash (q_1,0110,Z) \vdash (q_2,0110,Z)$ ↯

⊤

$(q_0,110,0Z) \vdash (q_1,110,0Z)$ ↯

⊤

$(q_0,10,10Z) \vdash (q_1,10,10Z) \vdash (q_1,0,0Z) \vdash (q_1,\varepsilon,Z) \vdash (q_2,\varepsilon,Z)$ √

⊤

$(q_0,0,110Z) \vdash (q_1,0,110Z)$ ↯

⊤

$(q_0,\varepsilon,0110Z) \vdash (q_1,\varepsilon,0110Z)$ ↯

Wie akzeptiert ein PDA?

Es gibt zwei alternative Definitionen:

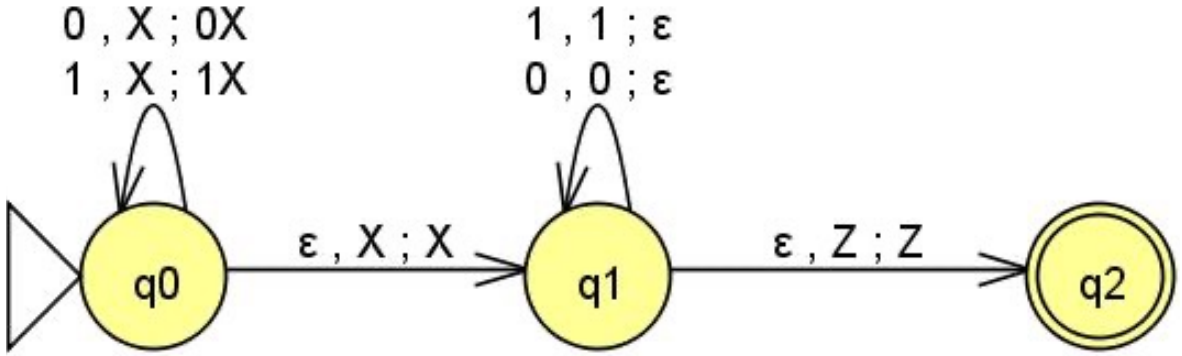
1. Akzeptanz durch akzeptierende Endzustände:
   Standarddefinition: Nach Abarbeitung der Eingabe entscheidet der Zustand, ob das Wort akzeptiert wird.
2. Akzeptanz durch leeren Stack:
   Neu: Nach Abarbeitung der Eingabe entscheidet der Stack: wenn er leer ist, wird das Wort akzeptiert.

Bemerkungen:

- Akzeptanz durch leeren Stack oft praktischer: Nach Abarbeitung der Eingabe sind auch alle zwischengelagerten Symbole verarbeitet.
- Definitionen haben verschiedene Effekte, d.h. die Sprachen können für konkrete PDAs sehr verschieden ausfallen.
- Beide Definitionen sind gleich mächtig (PDA kann passend zur anderen Definition umgewandelt werden).

Beispiel

Jeder PDA akzeptiert zwei Sprachen, die auch unterschiedlich sein können!



Akzeptieren durch Endzustände:

$L_F = \{ ww^R \mid w \in \{0,1\}^* \}$

Akzeptieren durch leeren Stack:

$L_\varepsilon = \{\}$ (Weil Z nie gelöscht wird)

Aber: Wenn der Übergang von q_1 nach q_2 in „ε,Z/ε“ geändert wird, dann gilt: $L_\varepsilon(P) = L_F(P) = \{ ww^R \mid w \in \{0,1\}^* \}$

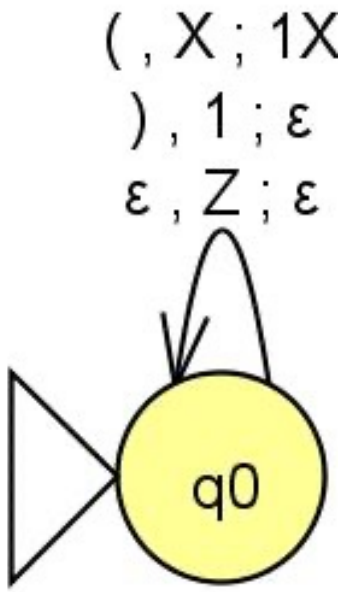
Erkennen durch leeren Stack ist oft einfacher

Konstruiere PDA für korrekte Klammerausdrücke:

- Anzahl geöffneter und geschlossener Klammern muss gleich sein.
- Eine Klammer muss erst geöffnet und dann geschlossen werden.

Idee: Zähle Überschuss geöffneter Klammern im Stack

- Jedes „(„ erhöht die Anzahl, jedes „)“ erniedrigt sie.
- „)“ ist nicht erlaubt, wenn der Stackboden erreicht ist.
- Am Ende des Wortes wird der Stackboden entfernt.



PDA:

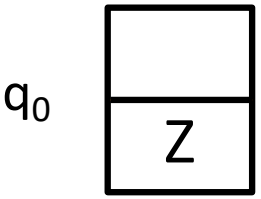
P = ({q}, {(,)}, {Z, 1}, δ, q, Z, {}) mit

- δ(q, (, X) = {(q, 1X)}
- δ(q,), 1) = {(q, ε)}
- δ(q, ε, Z) = {(q, ε)}

... wie ist es bei ()(()(()())), (() und (()()) ?

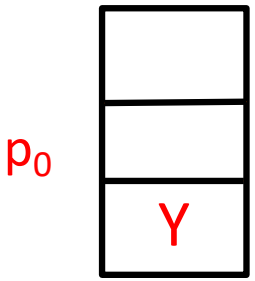
Transformation von L_ε in L_F : Idee

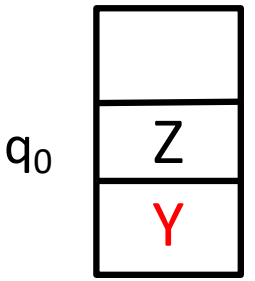
Gegeben: Ein L_ε-Automaten mit q_0 Startzustand und Z Startsymbol.



Wir bauen einen L_F-Automaten (mit p_0 Startzustand und Y Startsymbol), der L_ε verwendet:

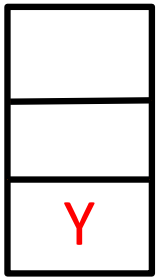
1. Setze ein Z auf das Y und gehe zu q_0 (Start von L_ε):





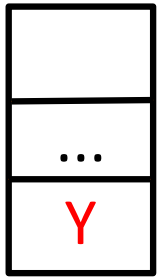
2. Durchlaufe das Wort in L_ε wie gewohnt.

3. Wie sieht danach der Stack aus?
   - Nur Y: Wechsel in neuen Endzustand und akzeptiere.



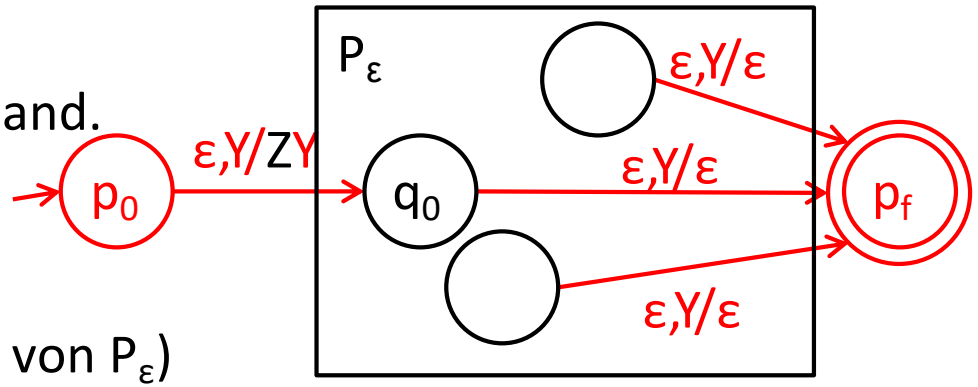
√

   - Mehr als nur Y: Sackgasse



↯

Transformation eines L_ε- in einen L_F-Automaten

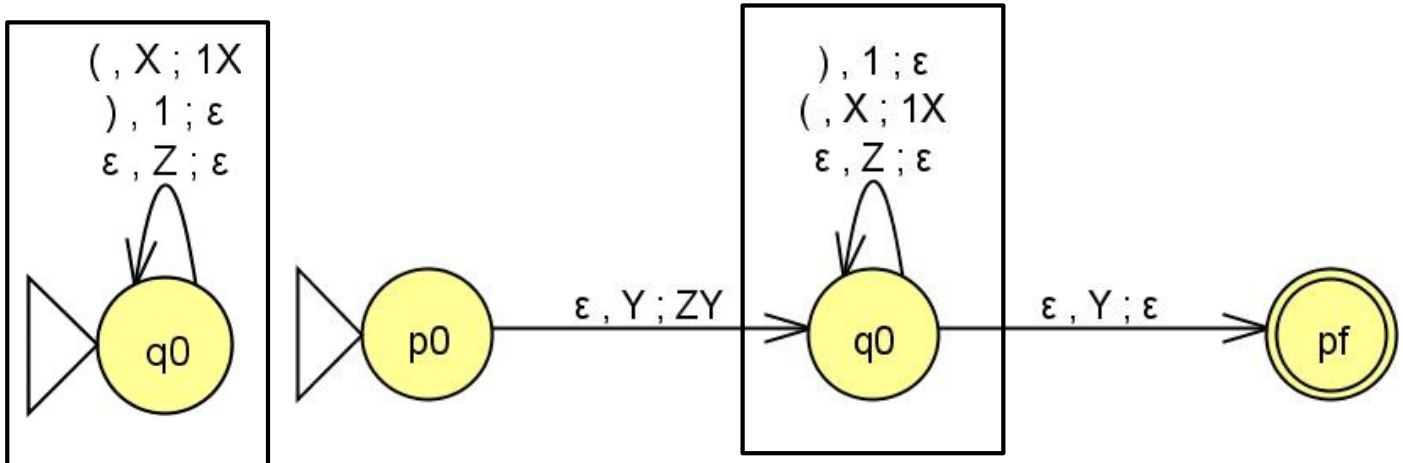


Idee: Bei leerem Stack wechsele in Endzustand.

$P_\varepsilon = (Q_\varepsilon, \sum_\varepsilon, \Gamma_\varepsilon, \delta_\varepsilon, q_0, Z, F_\varepsilon)$

$P_F = (Q_\varepsilon \cup \{p_0, p_f\}, \sum_\varepsilon, \Gamma_\varepsilon \cup \{Y\}, \delta_\varepsilon \cup \{\ldots\}, p_0, Y, \{p_f\})$

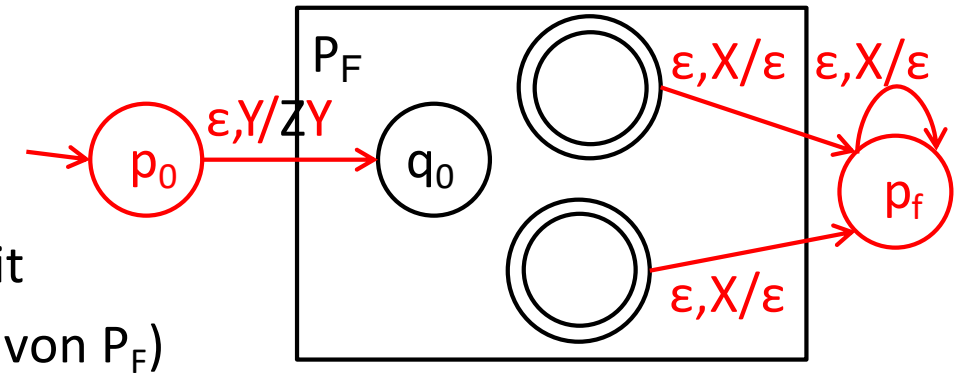
- Neues Stacksymbol Y (nicht in Wildcard von P_ε) markiert unteres Ende des Stacks von P_F.
- Neuer Anfangszustand p_0 für P_F schreibt Z auf Stack.
- Neuer Endzustand p_f wird bei „leerem" P_ε-Stack erreicht.
- Von allen Zuständen neue Übergange ε,Y/ε zum Endzustand.



Zu jedem PDA P_ε kann ein PDA P_F konstruiert werden mit $L_\varepsilon(P_\varepsilon) = L_F(P_F)$!

$(p_0,w,Y) \vdash (q_0,w,ZY) \vdash \ldots \vdash (q_i, \varepsilon, \varepsilon Y) \vdash (p_f, \varepsilon, \varepsilon)$

Transformation eines L_F- in einen L_ε-Automaten

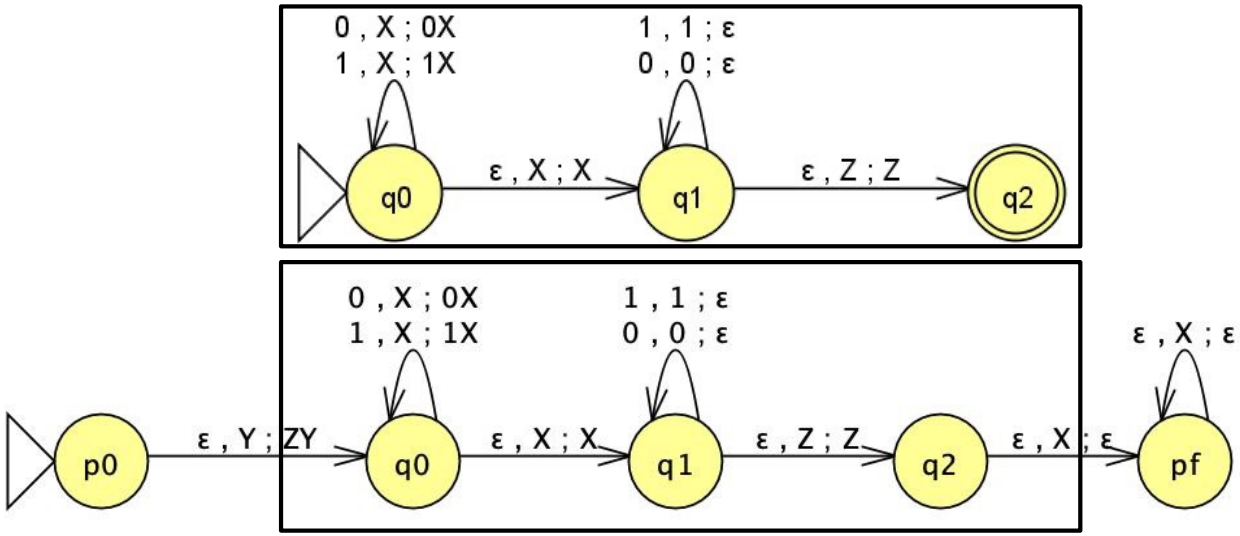


Idee: Im Endzustand leere den Stack.

$P_F = (Q_F, \sum_F, \Gamma_F, \delta_F, q_0, Z, F_F)$

$P_\varepsilon = (Q_F \cup \{p_0,p_f\}, \sum_F, \Gamma_F \cup \{Y\}, \delta_F \cup \{\ldots\}, p_0, Y, \{\})$ mit

- Neues Stacksymbol Y (nicht in Wildcard von P_F) markiert unteres Ende des Stacks von P_ε.
- Neuer Anfangszustand p_0 für P_ε schreibt Z auf Stack.
- Von allen bisherigen Endzuständen neue Übergange ε,X/ε zu p_f.



Zu jedem PDA P_F kann ein PDA P_ε konstruiert werden mit $L_F(P_F) = L_\varepsilon(P_\varepsilon)$!

$(p_0,w,Y) \vdash (q_0,w,ZY) \vdash \ldots \vdash (q_f, \varepsilon,S_1\ldots S_nY) \vdash (p_f, \varepsilon,S_2\ldots S_nY) \vdash \ldots \vdash (p_f, \varepsilon, \varepsilon)$

Sind PDAs wirklich Maschinen für Typ-2 Sprachen?

Die Konfigurationsübergänge entsprechen
Linksableitungen von Typ-2 Grammatiken !

Linksableitung:

Es wird immer die am weitesten links stehende Variable bearbeitet.

Grammatik → Pushdown-Automat:

- PDA muss Linksableitung auf Stack simulieren
- Erzeugte linke Terminalteilwörter müssen mit Teil der Eingabe verglichen werden, um nächste Variable freizulegen

Pushdown-Automat → Grammatik (geht, ist aber etwas kompliziert)

- Grammatik muss Abarbeitung von Symbolen des Stacks simulieren.
- Regeln beschreiben, wie PDA bei Abarbeitung des Stacksymbols X mit δ Zwischenwörter im Stack auf- und schließlich wieder abbaut.

Umwandlung einer Typ-2 Grammatik in einen PDA: Beispiel

$G = (\{E, Z\}, \{0,1,2,3,+,-,*,/,(,)\}, P_G, E)$ mit

$P_G = \{E \rightarrow Z \mid -Z \mid E+E \mid E-E \mid E*E \mid E/E \mid (E),$

$Z \rightarrow 0|1|2|3|1Z|2Z|3Z\}$

Neuer PDA:

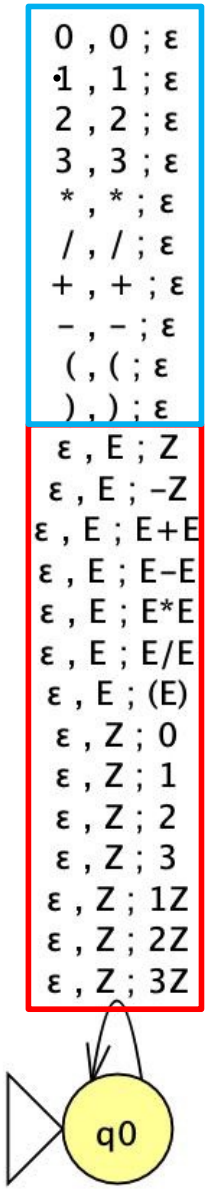
$P = (\{q_0\}, \{0,1,2,3,+,-,*,/,(,)\},$
$\{E,Z, 0,1,2,3,+,-,*,/,(,)\}, \delta, q_0, E, \{\})$

Für jedes Terminal eine Schleife:

$\delta(q_0,0,0)=\{(q_0,\varepsilon)\}$ $\delta(q_0,*,*)=\{(q_0,\varepsilon)\}$ $\delta(q_0,(,()=\{(q_0,\varepsilon)\}$
$\delta(q_0,1,1)=\{(q_0,\varepsilon)\}$ $\delta(q_0,/,/)=\{(q_0,\varepsilon)\}$ $\delta(q_0,),))=\{(q_0,\varepsilon)\}$
$\delta(q_0,2,2)=\{(q_0,\varepsilon)\}$ $\delta(q_0,+,+)=\{(q_0,\varepsilon)\}$
$\delta(q_0,3,3)=\{(q_0,\varepsilon)\}$ $\delta(q_0,-,-)=\{(q_0,\varepsilon)\}$

Jede Regel wird eine ε-Schleife:

$\delta(q_0,\varepsilon,E) = \{(q_0,Z),(q_0,-Z),(q_0,E+E),(q_0,E-E),(q_0,E*E),(q_0,E/E),(q_0,(E))\}$
$\delta(q_0,\varepsilon,Z) = \{(q_0,0),(q_0,1),(q_0,2),(q_0,3),(q_0,1Z),(q_0,2Z),(q_0,3Z)\}$



Umwandlung einer Typ-2 Grammatik in einen PDA: Theorie

Stack simuliert Linksableitungen von $G = (V, T, P_G, S)$

1. Beginne mit Startsymbol von G im Stack.
2. Wiederhole solange bis Eingabe abgearbeitet:
   1. Oben auf dem Stack ist eine Variable $A \in V$:
      A wird im Stack durch rechte Seite β einer Regel $A \rightarrow \beta$ ersetzt.
   2. Oben auf dem Stack ist ein Terminal $a \in T$:
      a wird vom Stack entfernt, wenn a auch als Eingabe erscheint.

Generierter PDA $P = (\{q\}, T, V \cup T, \delta, q, S, \{\})$ mit

- $\delta(q,\varepsilon,A) = \{(q,\beta) \mid A \rightarrow \beta \in P_G\}$ für alle $A \in V$ (ε-Übergang)
- $\delta(q,a,a) = \{(q,\varepsilon)\}$ für alle $a \in T$

Zu jeder kontextfreien Grammatik G
kann ein PDA P konstruiert werden mit $L(G) = L_\varepsilon(P)$

Deterministische Pushdown-Automaten

Ein **Deterministischer** Pushdown-Automat (DPDA) ist ein 7-Tupel

$P = (Q, \sum, \Gamma, \delta, q_0, Z, F)$ mit

- Q: nichtleere endliche Zustandsmenge
- $\sum$: endliches Eingabealphabet
- Γ: endliches Stackalphabet
- δ: $Q \times (\sum \cup \{\varepsilon\}) \times \Gamma \rightarrow Q \times \Gamma^*$ Überführungsfunktion
  $\delta(q, \varepsilon, X)$ nur definiert, wenn $\delta(q,a,X)$ für alle $a \in \sum$ undefiniert.
- $q_0 \in Q$: Startzustand
- $Z \in \Gamma$: Initialsymbol des Stacks
- $F \subseteq Q$: Menge von akzeptierenden Zuständen

Bei Typ-3 Sprachen: **Jeder NEA kann in einen DEA umgewandelt werden.**

Bei Typ-2 Sprachen? ***Kann jeder PDA in einen DPDA umgewandelt werden ?***

NEIN !

DPDAs sind nicht so mächtig wie PDAs

$L(DPDA) \subseteq L_2$: Jeder DPDA ist ein spezieller PDA

DPDAs können $\{ ww^R \mid w \in \{0, 1\}^* \}$ nicht erkennen.

Intuitiv: DPDA P kann nicht entscheiden, wo die Mitte eines Wortes liegt.

- Wenn 0^n110^n (großes n) gelesen ist, ist Stack durchs Zählen geleert
- Wenn noch einmal 0^n110^n gelesen wird, muss P akzeptieren
- Wenn stattdessen 0^m110^m ($m \neq n$) kommt, darf P nicht akzeptieren
- Aber die Information über n ist nicht mehr gespeichert

DPDAs erkennen nur eindeutige Typ-2 Sprachen!

DPDA-Sprachen sind eine echte Teilklasse von L_2

Zusammenfassung

- Reguläre Sprachen sind für Compiler nicht ausreichend, weil Sie keine Schachtelungen bearbeiten können.
- Ein nichtdeterministischer endlicher Automat mit Stack und ε-Übergängen (PDA) ist eine Erweiterung eines ε-NEA durch einen Kellerspeicher.
- PDAs können durch Endzustände oder durch leeren Stack akzeptieren. Dadurch kann ein PDA zwei verschiedene Sprachen akzeptieren.
- Ein PDA, der durch Endzustände akzeptiert, kann in einen PDA, der durch leeren Stack akzeptiert, umgewandelt werden, so dass die Sprache erhalten bleibt. Dies geht auch umgekehrt.
- PDAs sind äquivalent zu kontextfreien Grammatiken (Umwandlung von Konfigurationsübergängen in Regeln und umgekehrt).
- Deterministische PDAs sind weniger mächtig als nichtdeterministische PDAs.
- Für jede Sprache L(P) eines DPDA P gibt es eine eindeutige Grammatik G mit L(P)=L(G).