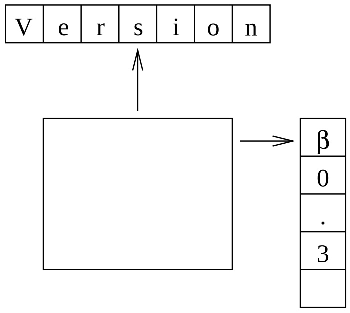
Formale Sprachen

Mirosław Kutyłowski

SS 1994



$\LaTeX\,2_\varepsilon$–Version
von
Dirk Freykamp
Frank Freykamp

Dieses Skriptum ist nach einer Mitschrift der Vorlesung entstanden. Es ist weitestgehend vollständig, aber natürlich nicht fehlerfrei! Falls jemand einen oder mehrere Fehler in diesem Skript findet, so möge er oder sie bitte eine E–Mail an eine der untenstehenden Adressen schicken. Die Fehler werden in regelmäßigen Abständen korrigiert und eine neue Version wird dann in das Netz eingespielt.
Version von 31.5.96

Dirk Freykamp (wyrth@uni–paderborn.de)
Frank Freykamp (lion@uni–paderborn.de)

Inhaltsverzeichnis

1 Reguläre Sprachen **1**

1.1 Endliche Automaten 1
1.2 Besondere endliche Automaten 4
1.3 Softwaresimulation von DEA 6
1.4 Optimierung von DEA 7
1.5 Reguläre Ausdrücke 13
1.6 Eigenschaften der regulären Sprachen 15
1.7 Entscheidungsprobleme für reguläre Sprachen 17

2 Kontextfreie Sprachen **19**

2.1 Normalformen 22
2.2 Kellerautomaten 25
2.3 Eigenschaften von kontextfreien Sprachen 29
2.4 Abgeschlossenheit bei Boolschen Operationen 30

3 Parsing **35**

3.1 Shift–Reduce–Parsing 35
3.2 Top–down–Parsing 41
3.3 Prädiktive Parser 42
3.4 LR(0)–Grammatiken 47
3.5 LR(0)–Parser 50
3.6 LR–Parser 56
3.7 Simple–LR–Parsing–Tabelle (SLR) 57
3.8 Kanonische LR–Parsing–Tabelle 58
3.9 Look–Ahead–LR–Parsing (LALR) 59
3.10 LR(k)–Grammatiken 63

4 Eigenschaften von deterministisch kontextfreien Sprachen **64**

5 Entscheidbarkeitsfragen für kontextfreie Sprachen **68**

6 Anhang **72**

6.1 Zusammenfassung: Abschlußeigenschaften 72
6.2 Zusammenfassung: Entscheidbarkeitsfragen 72
6.3 Literatur 72

1 Reguläre Sprachen

Definition: Translatoren

$$\text{Programmiersprache} \underset{\text{Translator}}{\longrightarrow} \text{Programmiersprache}$$

Programm

↓

Lexikalische Analyse

↓

Syntaktische Analyse

↓

Zwischen–Code

↓

Optimierung

↓

Code

Definition: „Pass“ eines Compilers

(Programm)

↓ zeitversetzt wegen Tokenbearbeitung

↓

Output (Zwischen–Code)

Der Zwischen–Code stellt somit eine „Kopie“ des Programms dar, auf der dann optimiert wird.

Lexikalische Analyse	$\leftrightarrow$	Endliche Automaten
Syntaktische Analyse	$\leftrightarrow$	Grammatiken (KfG)

1.1 Endliche Automaten

Lexikalische Analyse

Suche nach Token

- Schlüsselwörter (z.B. :=, ++)
- Standardkonstrukte (z.B. if ... then ... else)
- Konstante (z.B. 120, 'Hallo')
- Variablen, Arrayelemente (z.B. [1, 120], [1, 'Hallo'])

Problem:

- Konstruktionen wie `if` ... `then` ... `else` können sehr viel Platz in Anspruch nehmen ($\leftrightarrow$ Speicherkapazität)

  (Welches `if` zu welchem `else` gehört, wird hier noch nicht betrachtet.)

- verschiedene Tokens haben gleiche Präfixe

  `blabla[` ← Array
  `blabla` ← Schlüsselwort, Prozedurname

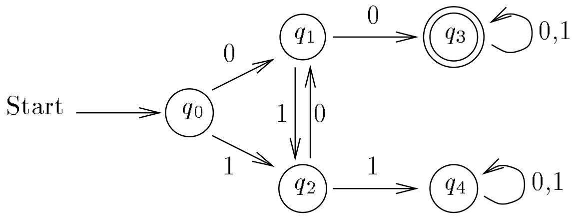
- verschiedene Parameter sind noch nicht vorhanden

Definition: Grundbegriffe

Alphabet	$\Sigma = \{a, b, c, ...\}$
Worte	Strings von Buchstaben, Menge aller Worte = Σ^*
Sprache	Teilmenge von Σ^*
DEA	Deterministischer endlicher Automat

<u>Beispiel:</u>

Die Worte sind zu erkennen, die erst zwei Nullen vor zwei Einsen enthalten.



<u>Andere Darstellung:</u>

Input

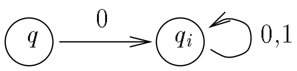
↑ Lesekopf (bewegt sich nur von links nach rechts)

Control Unit (endlicher Speicher)

- Übergangsfunktion δ
  - $\delta : Q \times \Sigma \longrightarrow Q$
  - $\delta(q, a) = q'$ (Nach Lesen von a in Zustand q gelangt der DEA in Zustand q'.)
  - Beispiel (s.o.): $\delta(q_0, 0) = q_1$
- Q ist eine <u>endliche</u> Menge von Zuständen.
- <u>DEA akzeptiert</u> einen Input, falls nach dem Lesen vom letzten Buchstaben der DEA in einen akzeptierenden Zustand gelangt.

  Formal: Input $w = w_1...w_n$ und $\delta(q_i, w_n) = q'$ und $q' \in F$, wobei F = Menge der akzeptierenden Zustände

Definition: Nichtdeterministische endliche Automaten



- δ ist keine Funktion!
- $\delta \subset Q \times \Sigma \times Q$
- $\delta(q, a, q')$

Ein nichtdeterministischer endlicher Automat akzeptiert eine Sprache L, gdw. $L = \{w \in \Sigma^* \mid$ eine Berechnung von A mit Input w existiert, die in einem akzeptierenden Zustand endet$\}$

Beispiel:

Pattern Matching: `blabla ···` `Text`

Satz:

1. Für jeden NEA A existiert ein DEA A' mit $L_A = L_{A'}$.
2. Falls A n Zustände hat, dann hat A' maximal 2^n Zustände.

Beweis:

Für jede Position in dem Input reicht es zu zeigen, welche Zustände für A möglich sind.

Diese Mengen werden durch A' erzeugt.

Genauer: Die Zustände von A' sind Teilmengen von Q (= Menge der Zustände von A) $\Rightarrow$ *Potenzmengenkonstruktion*:

$A = (Q, \Sigma, \delta, q_0, F)$
$A' = (Q', \Sigma, \delta', q_0', F')$
mit $Q' = \mathcal{P}(Q) \Rightarrow |Q'| = 2^{|Q|}$
Bilde δ' von A':
$\delta(\{q_1, \ldots, q_n\}, a) = \{p_1, \ldots, p_i\}$ mit
$\{p_1, \ldots, p_i\} = \{p \mid \exists j : \delta(q_j, a) = p)\}$
$q_0' = \{q_0\}$
$F' = \{T \mid T \in Q'$ und $\exists q \in T$ mit $q \in F\}$

Bemerkung:

Bei der Potenzmengenkonstruktion kann man die Übergangsfunktionen on–line berechnen:

+ wenig Speicherplatz
− Laufzeit wächst

$\rightarrow$ klassischer Trade–off für Compiler (und derartige Programme)

- sehr lange Inputs: Laufzeit wichtig
- kurze Inputs: Speicherplatz wichtig

Definition: Automaten mit ε–Übergängen

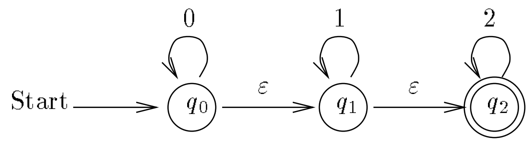
ε bezeichnet das leere Wort. Ein Automat mit ε–Übergängen hat die Möglichkeit, den nächsten Eingabebuchstaben nicht zu „schlucken“:

$$\delta \subset Q \times (\Sigma \cup \{\varepsilon\}) \times Q$$

Beispiel:

- Alle Worte die so aussehen: $0 \cdots 01 \cdots 12 \cdots 2$

- Diagramm für Automaten



Satz:

Frage:	Können Automaten mit ε–Übergängen mehr Sprachen akzeptieren?
Antwort:	Nein!

<u>Beweis:</u> (Skizze)

- Berechne die Liste von Zuständen, die durch ε–Schritte erreichbar sind.
  $\{q_1, \ldots, q_s\}$
  ε–Closure$(\{q_1, \ldots, q_s\})$ = „Liste solch erreichbarer Zustände"

<u>Procedure:</u>

$W = \{q_1, \ldots, q_s\}$, $N = \{q_1, \ldots, q_s\}$

WHILE ($N \neq \varnothing$)

Wähle $q \in N$

$$\left.\begin{array}{ll} W = & W \cup \{p\} \\ N = & N \cup \{p\} \end{array}\right\} \text{ für } p \text{, so daß } \delta(q, \varepsilon, p) \text{ und } p \notin W$$

$N = N \setminus \{q\}$

END

Äquivalenter DEA (ohne ε–Schritte)

- Zustände: Teilmengen der Zustände von A
- Übergangsfunktion:

$$\begin{array}{c} \{q_1, \ldots, q_s\} \\ \downarrow \\ \{r_1, \ldots, r_n\} = \varepsilon\text{–Closure}(\{q_1, \ldots, q_s\}) \\ \downarrow \\ \downarrow \text{ Buchstaben, die} \\ \downarrow \text{ wir lesen, z.B. } a \\ \downarrow \\ \{p_1, \ldots, p_L\} = \{p \mid \exists i : \delta'(r_i, a, p)\} \end{array}$$

1.2 Besondere endliche Automaten

Definition: Mealy–Automaten

Ein Mealy–Automat wird definiert durch $M = (Q, \Sigma, \Delta, \delta, \lambda, q_0)$, wobei

- Q = endliche Menge von Zuständen
- Σ = Eingabealphabet
- Δ = Ausgabealphabet
- δ = Übergangsfunktion
- λ = Ausgabefunktion
- q_0 = Startzustand

Ausgabe:

- bei jedem Schritt schreibt ein Schreibkopf auf dem Ausgabeband ein Wort (schreiben immer am Ende)
- was man genau schreibt: $\lambda(q, a)$ mit $q \in Q, a \in \Sigma$

Beispiel:

Lexical Analysers

Definition: Zwei–Wege–Automaten

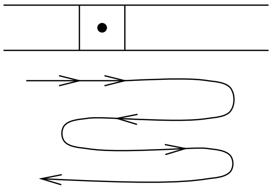
- Lesekopf darf sich in beide Richtungen bewegen
- Übergangsfunktion:
  - $\delta : Q \times \Sigma \to Q \times \{-1, 0, 1\}$
  - $\delta(q_1, a) = (q_2, i)$ bedeutet: „im Zustand q_1 beim Lesen von a $\to$ neuer Zustand: q_2 und der Lesekopf wird i Positionen nach rechts verschoben"

Satz:

Sei L eine Sprache, die von einem Zwei–Wege–Automaten A akzeptiert wird. Es gibt einen DEA A', der L auch akzeptiert.

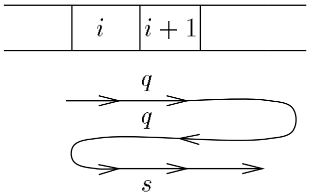
Beweis:

(Crossing sequence (CS))



Crossing sequence:

CS(i) = Folge der Zustände bei denen der Lesekopf die Grenze zwischen Zelle i und $i + 1$ überschreitet.



Bei endlicher Berechnung kommt jeder Zustand in jeder Richtung höchstens einmal vor.

Konstruieren wir NEA, der L akzeptiert:

Zustände:	alle möglichen CS (von endlichen Berechnungen) (Es gibt endlich viele davon.)
Übergang:	– Zuerst raten wir (ndet.) die nächste CS – Korrektheit (lokal) überprüfen

Simulation:

Immer neue CS raten, die lokal übereinstimmt.

Beispiel:

Mustererkennungsautomat A_w erkennt, ob das Wort w als Teilfolge im Input vorkommt.

2–Wegeautomat:

Phase i: A_w untersucht, ob w in der Zelle i anfängt.
Falls ja: Gehe nach rechts.
Falls nein: Gehe zur Position $i+2$ und probiere neu.

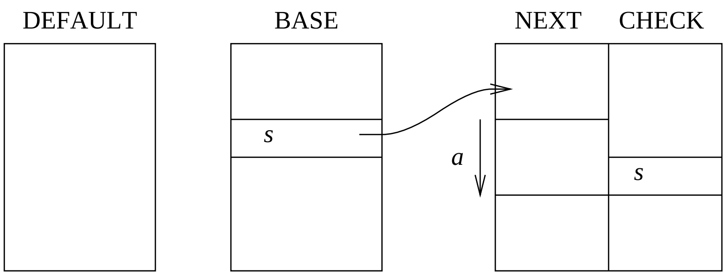
1.3 Softwaresimulation von DEA

Problem:

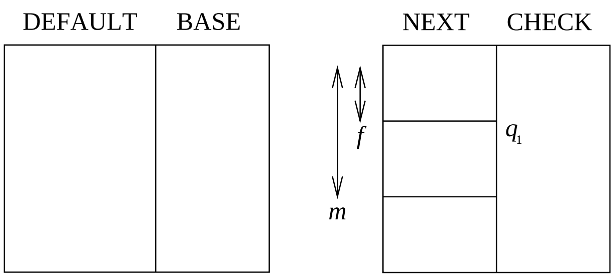
Wie sollen die Informationen über Übergangsfunktion gespeichert werden?

- look–up table $\rightarrow$ sehr viel Platz
- Einige Kombinationen von Zustand+Buchstabe sind nicht zulässig!
- Bei mehreren Kombinationen können die Ergebnisse gleich sein.

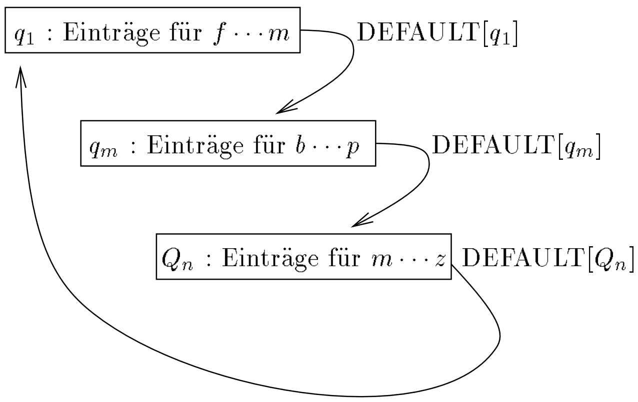
Datenstruktur:




```
procedure NEXTSTATE(s,a)
  if CHECK[Base[s]+a] = s
    return NEXT[BASE[s]+a]
  else
    return NEXTSTATE(DEFAULT[s],a)
```



Struktur:



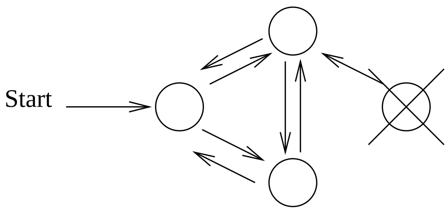
1.4 Optimierung von DEA

Bemerkung:

Automat A mit k Zuständen akzeptiert L.

Gibt es einen Automaten mit weniger Zuständen, der L akzeptiert?

Diagramm malen und Zustände streichen:



Satz:

- Es gibt einen Automaten der L erkennt und eine minimale Anzahl von Zuständen besitzt.
- Man kann diesen Automaten algorithmisch finden.

Definition:

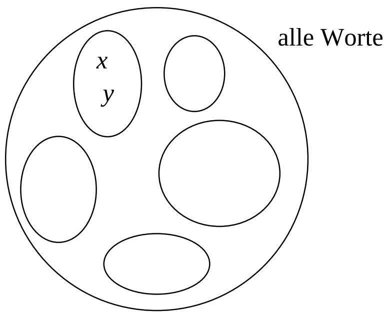
Sei L eine Sprache. R_L ist definiert durch:

$xR_Ly \Leftrightarrow \forall z : (xz \in L \Leftrightarrow yz \in L)$ (z ein Wort.)

R_L ist eine Äquivalenzrelation (d.h. es gilt:

xR_Lx,
$xR_Ly \Rightarrow yR_Lx$,
$xR_Ly \wedge yR_Lz \Rightarrow xR_Lz$

)



Definition:

M sei ein DEA.

- $xR_My \Leftrightarrow$ „nach dem Lesen von Wort x erreicht M denselben Zustand wie nach dem Lesen von Wort y“
- R_M ist eine Äquivalenzrelation.
- $xR_My \Rightarrow \forall z : xzR_Myz$ (rechtsinvariante Relation)

Hilfssatz:

Die folgenden Bedingungen sind äquivalent:

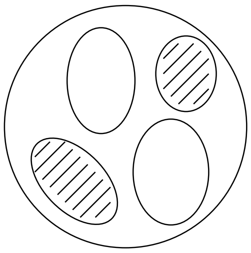
1.) Sprache L wird von einem DEA akzeptiert.

2.) L ist eine Summe von Äquivalenzklassen einer rechtsinvarianten Relation, die endlich viele Äquivalenzklassen besitzt.

3.) R_L besitzt endlich viele Äquivalenzklassen.

Beweis:

1.) $\rightarrow$ 2.) : L ist die Summe einiger Äquivalenzklassen von R_M.

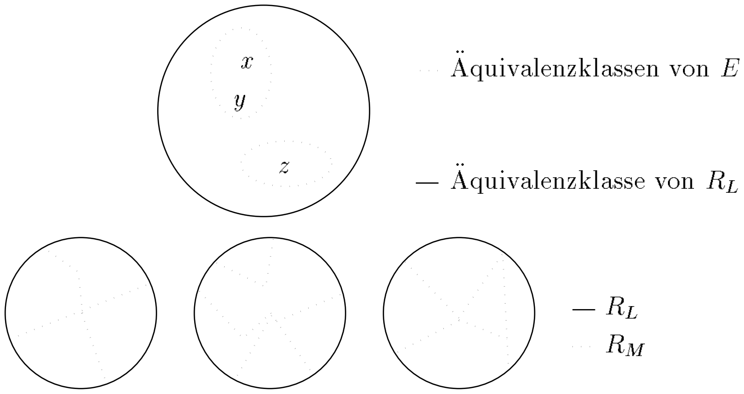


2.) $\rightarrow$ 3.) : Sei E die rechtsinvariante Relation von 2.).

$$
\begin{array}{lcll}
xEy \Rightarrow & \forall z : & xzEyz & \\
 & & \Downarrow & \\
 & \forall z : & xz \in L & \Leftrightarrow yz \in L \\
 & & \Updownarrow & \\
 & & xR_Ly &
\end{array}
$$

$\Rightarrow$ Jede Äquivalenzklasse von R_L ist eine Summe von einigen Äquivalenzklassen von E.

$\Rightarrow$ R_L besitzt endlich viele Äquivalenzklassen.



3.) $\to$ 1.) : Sei x ein Wort. $[x]$ ist Äquivalenzklasse, die x enthält.
Automat:

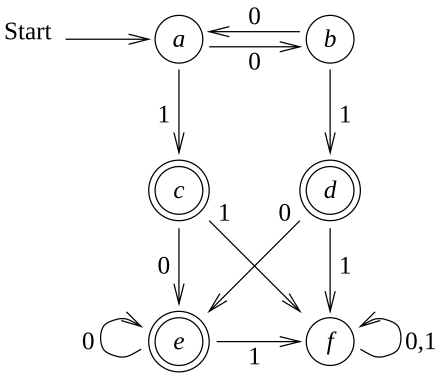
Zustände: Äquivalenzklassen von R_L
Übergangsfunktion: $\delta([x], a) = [xa]$

(Ist das sinnvoll?
$[x] = [y]$, $\delta([x], a) = \delta([y], a) = [ya]$
aber: $xR_Ly \Rightarrow xaR_Lya \equiv [xa] = [ya]$
o.k.)

Eigenschaften:
Der Automat befindet sich nach dem Lesen von einem Wort x in dem Zustand $[x]$.
Bew.: einfache Induktion
Akzeptierende Zustände: $[x]$ ist akzeptierend $\Leftrightarrow [x] \subseteq L$.

Beispiel:

$$L = \{w \mid \underbrace{0\ldots0}_{k}1\underbrace{0\ldots0}_{l},\ k, l \in \mathbb{N}\}$$



R_M :

$$C_a = \{w \mid w = \underbrace{0\ldots0}_{2k},\ k \in \mathbb{N}\}$$

$$C_b = \{w \mid w = \underbrace{0\ldots0}_{2k+1},\ k \in \mathbb{N}\}$$

C_c

$\vdots$

C_f

R_L :

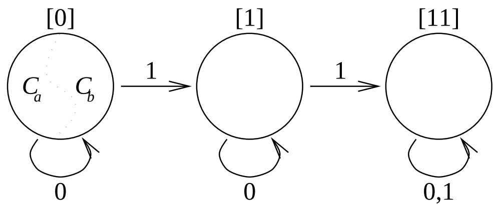
$$\begin{aligned} xR_Ly \Leftrightarrow \quad & (x, y \in \{0\}^* \\ & \vee \text{ (beide, } x \text{ und } y \text{ enthalten je eine 1)} \\ & \vee \text{ (beide, } x \text{ und } y \text{ enthalten mindestens 2 Einsen))} \end{aligned}$$

$x, y \in \{0\}^*$, $z \in \{0,1\}^*$, $xR_Ly : \forall z(xz \in L \Leftrightarrow yz \in L)$

1.) z enthält 2 Einsen $\Rightarrow xz \notin L$, $yz \notin L$

2.) z enthält eine Eins $\Rightarrow \left.\begin{array}{l} xz \text{ enthält eine 1} \\ yz \text{ enthält eine 1} \end{array}\right\} \Rightarrow xz, yz \in L$

3.) z enthält keine Eins $\Rightarrow \left.\begin{array}{c} xz \\ yz \end{array}\right\}$ enthalten keine Eins $\Rightarrow xz, yz \notin L$



$\delta([0], 1) = [01] = [1]$

$\delta([0], 0) = [00] = [0]$

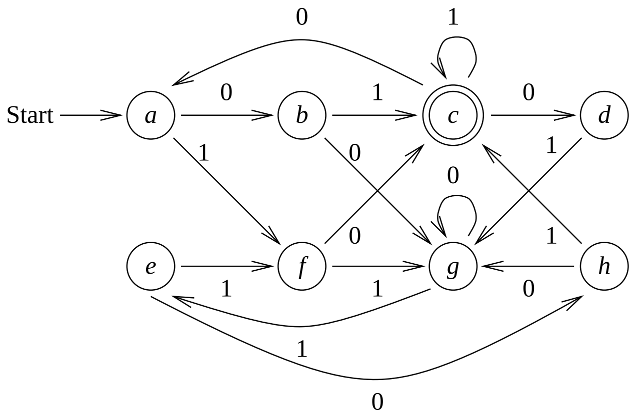
<u>Konstruktion von minimalen Automaten:</u>

Eingabe: ein Automat M (vermutlich nicht optimal)
Ausgabe: ein Automat M' (optimal)

Definition:

- p und q sind <u>äquivalent</u> $\Leftrightarrow$
  $\forall x :\quad (\delta(p, x)$ ist akzeptierend $\Leftrightarrow \delta(q, x)$ ist akzeptierend)
  - x Wort
  - $\delta(p, x) =$ Zustand von M nach dem Lesen von x, falls im Zustand p gestartet
- p und q sind <u>nicht äquivalent</u> $\Leftrightarrow$
  $\exists x\quad [(\delta(p, x)$ – akzeptierend $\wedge\, \delta(q, x)$ – nicht akzeptierend) $\vee$
  $(\delta(p, x)$ – nicht akzeptierend $\wedge\, \delta(q, x)$ – akzeptierend)]

<u>Beispiel:</u>



Acc bezeichne die Menge der akzeptierenden Zustände.

<u>Algorithmus:</u>

for $p \in Acc, q \notin Acc$ Eliminiere (p, q)

for jedes Paar $(p, q) \in (Acc \times Acc) \cup (Q \setminus Acc) \times (Q \setminus Acc)$

  if für Buchstabe a $(\delta(p, a), \delta(q, a))$ ist eliminiert

    Eliminiere (p, q)

    Eliminiere jedes Paar aus der Liste von (p, q) und rekursiv weiter

  else

    for jedes a:

      falls $\delta(p, a) \neq \delta(q, a)$

        Füge (p, q) in die Liste von $(\delta(p, a), \delta(q, a))$ ein

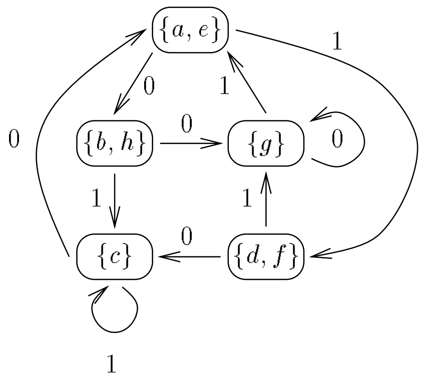
  endif

endfor

b	X	–	–	–	–	–	–
c	X	X	–	–	–	–	–
d	X	X	X	–	–	–	–
e		X	X	X	–	–	–
f	X	X	X		X	–	–
g	X	X	X	X	X	X	–
h	X		X	X	X	X	X
	a	b	c	d	e	f	g

$(a, b) \xrightarrow{0} (b, g)$ $(a, b) \xrightarrow{1} (f, c)$	$(f, a) \xrightarrow{0} (b, c)$
$(d, a) \xrightarrow{0} (b, c)$	$(g, a) \xrightarrow{0} (b, g)$ $(g, a) \xrightarrow{1} (f, e)$
$(e, a) \xrightarrow{0} (f, f)$ $(e, a) \xrightarrow{1} (b, h)$	$(h, a) \xrightarrow{0} (b, g)$ $(h, a) \xrightarrow{1} (c, f)$
$(e, b) \xrightarrow{0} (g, h)$ $(e, b) \xrightarrow{1} (c, f)$	$(d, b) \xrightarrow{0} (c, g)$

$\Rightarrow$



Lemma:

Dieser Algorithmus findet äquivalente Paare.

Beweis:

p, q seien nicht äquivalent:

- nehmen wir kürzestes Wort x, daß p und q unterscheidet
  $\delta(p, x)$ – akzeptierend
  $\delta(q, x)$ – nicht akzeptierend (oder umgekehrt)
  $x = \varepsilon \Rightarrow (p, q)$ werden durch 1. Schritt eliminiert.
  $x = ay$ (a Buchstabe, y Wort) $\Rightarrow u = \delta(p, a), v = \delta(q, a)$, wegen y sind u und v nicht äquivalent.

Aus der Induktionsannahme wird (u, v) irgendwann eliminiert.

Aber (p, q) steht auf der Liste von (u, v) und wird auch eliminiert.

Lemma:

Der Algorithmus eliminiert nur nicht äquivalente Paare.

Beweis:

Ähnlich.

Konstruktion:

$M \rightarrow$ „Automat von äquivalenten Zuständen“ $\rightsquigarrow$ nicht erreichbare Zustände streichen

Satz:

Konstruktion ist korrekt.

1.) Automat erkennt dieselbe Sprache.

2.) Ist neuer Automat minimal?

Beweis:

1.) $Q' = \{[q] : q \in Q\}$
$Acc' = \{[q] : q \in Acc\}$
$\delta'([q], a) = [\delta(q, a)]$
$(p \equiv q \Rightarrow \delta(p, a) \equiv \delta(q, a))$
Durch Induktion auf $|w|$ kann man zeigen:
$\delta'([q_0], w) = [\delta(q_0, w)]$ (q_0 Anfangszustand)

2.) An.: $|Q'| >$ Anzahl der Äquivalenzklassen von R_L.
$[q] \leftrightsquigarrow \{x \mid \delta(q_0, x) \in [q]\} \overset{def.}{\equiv} [q]_{M'}$
$\uparrow$ eine Äquivalenzklasse von $R_{M'}$
R_L ist refinement von $R_{M'}$, d.h. jede Äquivalenzklasse von $R_{M'}$ ist eine Summe von Äquivalenzklassen von R_L.
$\exists p, q \therefore [p]_{M'} \neq [q]_{M'}$, aber $[p]_{M'}$, $[q]_{M'}$ sind Teilmengen von derselben Äquivalenzklasse von R_L.
$x \in [p]_M$, $y \in [q]_{M'} \Rightarrow x R_L y$.

$\boxed{\begin{array}{l} \delta(q_0, x) = p \\ \delta(q_0, y) = q \end{array}}$ $\boxed{\begin{array}{l} p, q \text{ nicht äquivalent.} \\ \exists w \text{ mit } \delta(p, w) \in Acc \wedge \delta(q, w) \notin Acc. \end{array}}$

$\begin{array}{l} \delta(q_0, xw) \in Acc \\ \delta(q_0, yw) \notin Acc \end{array}$ $\quad x R_L y \Leftrightarrow \forall w (xw \in L \Leftrightarrow yw \notin L)$ $\natural$

1.5 Reguläre Ausdrücke

	Ausdrücke:	Entsprechende Sprachen:
1.)	$\varnothing$	$R_\varnothing = \varnothing$
2.)	ε	$R_\varepsilon = \varepsilon$
3.)	$a, \quad a \in \Sigma$	$R_a = a$
4.)	Falls r und s reguläre	$R_{r+s} = R_r \cup R_s$
	Ausdrücke sind $\Rightarrow$	$R_{rs} = \{x \mid \exists u, w : u \in R_r, w \in R_s\}$
	$r + s, rs, r^*, (\neg r)$ sind	$R_{r^*} = \{x \mid \exists i \exists x_1 \ldots x_i \in R_r, x = x_1 \ldots x_i\}$
	auch reguläre Ausdrücke	$R_{\neg r} = \Sigma \setminus R_r$

Definition: Induktiv

x ist regulärer Ausdruck $\Leftrightarrow$ man kann x mit Hilfe von 1.) bis 4.) erzeugen.

Satz:

Sei L eine Sprache. Dann sind folgende Bedingungen äquivalent:

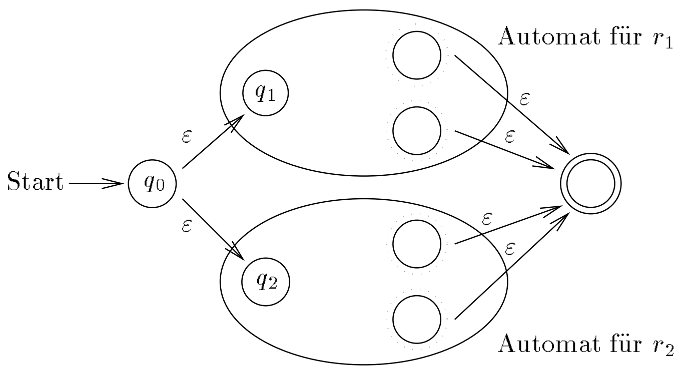
1.) $\exists$ DEA M : M akzeptiert L

2.) $\exists$ r: (r regulärer Ausdruck und $L = R_r$)

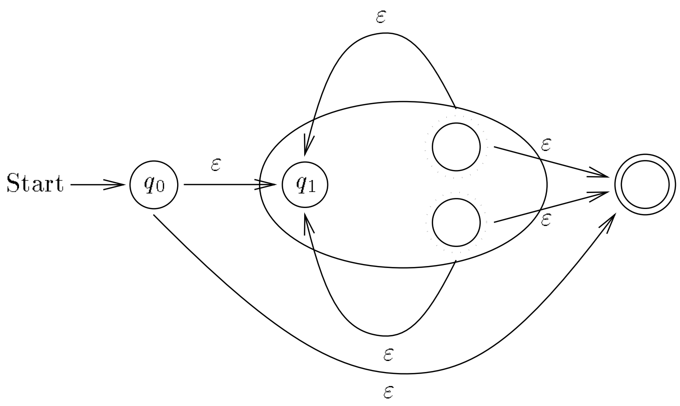
Beweis:

2.) $\Rightarrow$ 1.)

- Suchen wir ε–Schritt NEA
- Existenz von M induktiv
  ε; $\varnothing$; a, $a \in \Sigma$ } einfach
  IS: $r = r_1 + r_2$



$r = r^*$



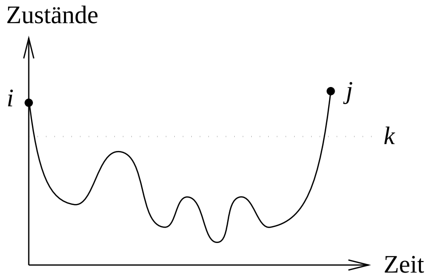
1.) $\Rightarrow$ 2.)

M mit Zuständen $q_1, \ldots, q_m$, δ Übergangsfunktion, q_0 Anfangszustand

Definition:

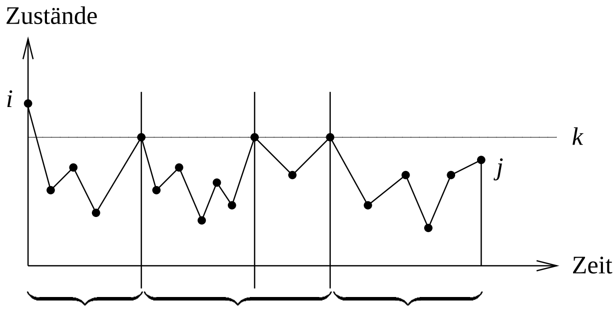
$R_{ij}^k = \{x \mid \delta(q_i, x) = q_j \wedge \forall y$ – Präfix von x $(y \neq \varepsilon, x) : \delta(q_i, y) = q_l \Rightarrow l < k\}$

Bei Eingabe von x:



$L = R_{0,s}^{m+1}$ $\quad$ q_s Endzustand

Beweis: (Induktiv)



$$R_{ij}^k = R_{ik}^{k-1} \quad (R_{kk}^{k-1})^* \quad R_{kj}^{k-1} \quad \cup R_{ij}^{k-1}$$

- R^0_{ij} ist durch regulären Ausdruck dargestellt. (Übung)
- Induktionsschritt: r^a_{bc} – regulärer Ausdruck für R^a_{bc} (falls existent)
  Induktion nach k:
  $$r^k_{ij} = r^{k-1}_{ij} \cdot (r^{k-1}_{kk})^* \cdot r^{k-1}_{kj} + r^{k-1}_{ij} \quad \square$$

Bemerkung:

- Für Automaten mit n Zuständen bekommen wir reguläre Ausdrücke der Länge $\approx 3^n$.
- regulärer Ausdruck der Länge n
  $\Rightarrow$ NEA hat $\approx n$ Zustände
  $\Rightarrow$ DEA hat $\approx 2^n$ Zustände
- Diese Translation ist für praktische Anwendungen nicht zu gebrauchen.
- Es gibt Gründe dafür (man kann beweisen: es geht nicht besser).

Korollar:

Für jeden regulären Ausdruck r gibt es einen regulären Ausdruck $\widetilde{r}$, der die Sprache $\Sigma^* \setminus R_r$ darstellt.

(z.B. $((c^* + ab^*) \cdot abc)^*$ — Wie kann man $\widetilde{r}$ konstruieren?)

Beweis:

Die durch DEA akzeptieren Sprachen sind gegenüber Negation abgeschlossen.

DEA entsprechen regulären Ausdrücken.

Definition:

Reguläre Sprachen $\equiv$ Sprachen, die durch DEA erkennbar sind.
$\equiv$ Sprachen, die durch r.A. darstellbar sind.

1.6 Eigenschaften der regulären Sprachen

Satz: Pumping Lemma

Sei L eine reguläre Sprache.

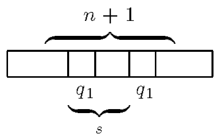
$\exists n : \forall x \in L[x = puq \wedge |u| > n \Rightarrow (\exists r, s, t(s \neq \varepsilon) : u = rst \wedge \forall k \in \mathbb{N} : prs^k tq \in L)]$

$$\boxed{p}\underbrace{\boxed{r}\boxed{s}\boxed{t}}_{n+1}\boxed{q}$$

$$\boxed{p}\boxed{r}\boxed{s}\boxed{s}\boxed{s}\boxed{s}\boxed{s}\boxed{t}\boxed{q}$$

Beweis:

DEA bei der Arbeit:



Beispiel:

$L = \{x \mid \exists n : 0^n \#^* 1^n\}$

$p = \varepsilon \quad \underbrace{0\ldots0}_{u}\underbrace{\#\ldots\#1\ldots1}_{q}$

Satz:

L_1, L_2 regulär $\Rightarrow L_1 \cap L_2$ und $L_1 \cup L_2$ sind regulär

Beweis:

DEA, der gleichzeitig Zustände von 2 Automaten berechnet.

Definition: Substitution

$f : \Sigma \rightarrow r.A.$

Verallgemeinerung für Σ^*:

$f(\varepsilon) = \varepsilon$

$f(a, w) = f(a) \cdot f(w)$ (a — Buchstabe, w — Wort)

$f(L) = \bigcup_{x \in L} f(x)$

Satz:

L regulär $\Rightarrow f(L)$ beschreibt eine reguläre Sprache.

Beweis:

Sei R regulärer Ausdruck für L. In R ersetzen wir jeden Buchstaben a durch $f(a)$. $\Rightarrow$ Ergebnis ist ein regulärer Ausdruck R_f

Lemma:

R_f ist regulärer Ausdruck für $f(L)$.

Beweis: Induktion

$R = a$	$a \in \Sigma$
$L = \{a\}$	$f(L)$ ist nun ein regulärer Ausdruck, also stellt $f(L)$ eine reguläre Sprache dar.

IS:

$$\bullet L = L_1 \cup L_2 : \quad \begin{array}{ccc} f(L_1 \cup L_2) & = & f(L_1) \cup f(L_2) \\ \| & & \| \\ \bigcup\limits_{x \in L_1 \cup L_2} f(x) & & \bigcup\limits_{x \in L_1} f(x) + \bigcup\limits_{x \in L_2} f(x) \end{array}$$

$$\bullet L = L_1 \cdot L_2 : \quad \begin{array}{ccl} f(L_1 \cdot L_2) & = & f(L_1) \cdot f(L_2) \\ \| & & \\ \bigcup\limits_{x \in L_1 \cdot L_2} f(x) & = & \bigcup\limits_{y_1 \in L_1} \bigcup\limits_{y_2 \in L_2} f(y_1 y_2) \\ & = & \bigcup\limits_{y_1 \in L_1} \bigcup\limits_{y_2 \in L_2} f(y_1) \cdot f(y_2) \\ & = & f(L_1) \cdot f(L_2) \end{array}$$

$\bullet L = L_1^* : f(L_1^*) = (f(L_1))^*$

Bew.: nach demselben Prinzip.

Homomorphismus

$f(a)$ — wie vorher ein regulärer Ausdruck, aber $f(a) = \{w\}$.

Also jeder Buchstabe wird durch ein Wort ersetzt.

$h^{-1}(L) = \{x \mid h(x) \in L\}$ inverser Homomorphismus

Satz:

L regulär $\Rightarrow h^{-1}(L)$ ist regulär.

Beweis:

M sei DEA, der L akzeptiert, man akzeptiert $h^{-1}(L)$ wie folgt:

1.) Buchstaben lesen (z.B. x wurde gelesen)
2.) Simuliere M auf $h(x)$
3.) Goto 1.)

Beispiel:

$h : \Sigma \to \Sigma^*$

$h^{-1}(L) = \{x \mid h(x) \in L\}$

$L_0 = \{x \mid \exists n \in \mathbb{N}\ x = 0^n 1^n\}$

$L_1 = \{x \mid \exists n \in \mathbb{N}\ x = a^n b a^n\}$

$h_1(a) = a$, $h_1(b) = ba$, $h_1(c) = a$

$h_1^{-1}(L) \cap a^* b c^* = \{x \mid \exists n \in \mathbb{N}\ x = a^n b c^{n-1}\}$

$h_2(a) = 0$, $h_2(b) = h_2(c) = 1$

$h_2(h_1^{-1}(L_1) \cap a^* b c^*) = L_0$

L_1 regulär $\Rightarrow L_0$ regulär

1.7 Entscheidungsprobleme für reguläre Sprachen

Fragen:

- Ist R_p leer?
- Sind R_p und R_q gleich?

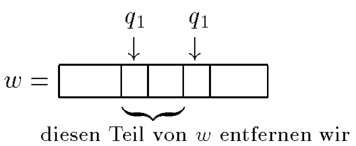
Satz:

M — DEA mit n Zuständen, L_M — die Sprache, die M akzeptiert.

1. $L_M \neq \varnothing \Leftrightarrow M$ akzeptiert ein Wort der Länge $\leq n$.
2. L_M ist unendlich $\Leftrightarrow M$ akzeptiert ein Wort der Länge l mit $n < l \leq 2n$.

Beweis:

(1) „$\Rightarrow$“
M akzeptiert w.



Das so entstandene neue Wort wird auch akzeptiert:

$w' = \boxed{\quad}\boxed{\;}\boxed{\quad}$

(2) „$\Leftarrow$“ nach Pumping–Lemma
„$\Rightarrow$“ Wort, die länger als $2n$ sind, kann man abkürzen, jedesmal um maximal n Zeichen.

Bei Transitionsdiagramm:

Frage: Ist die Sprache leer? $\Leftrightarrow$ Reachability–Problem bei gerichteten Graphen.

Für reguläre Ausdrücke:

Basisoperationen:	Problem:	Komplexität:
$+, \cdot, ^2, *, \neg$	Membership $w \in R_S$?	in Poly–Zeit lösbar: $\exists$ p – Polynom: Berechnungszeit $\leq p(n)$ für Eingabe der Länge n.
$+, \cdot, ^2$	Sind R_S und R_P gleich?	vollständig für ndet. exponentielle Zeit
$+, \cdot, ^2, *$	"	vollständig für exponentiellen Platz($*$)
$+, \cdot, \neg$	"	Platzkomplexität: $\left.2^{2^{\cdot^{\cdot^{2}}}}\right\\} \log n$ mal

($*$) auf 2^n Zellen kann man $\approx 2^{2^n}$ Schritte machen, ohne eine Konfiguration zu wiederholen.

2 Kontextfreie Sprachen

Beispiel:

Definition von Listen:

X ist eine Liste, falls

- $X = []$ oder
- $X = [Y|Z]$, Z eine Liste, $Y \in \Sigma$ oder Y ist eine Liste

Definition: Kontextfreie Grammatik

Eine kontextfreie Grammatik wird durch das Tupel $G = (V, T, P, S)$ definiert, wobei

- V = endliche Menge von Variablen (nichtterminale Symbole)
- T = endliche Menge von Terminalsymbolen
- S = Startsymbol ($\in V$)
- P = endliche Menge von Produktionen der Form $A \to \alpha$, wobei $A \in V$ und $\alpha \in (V \cup T)^*$.

Notation:

$S \Rightarrow S_1 \Rightarrow S_2 \Rightarrow \ldots \Rightarrow S_k$

S ist Startsymbol.

S_k enthält keine nichtterminalen Symbole.

$\forall i \quad S_{i+1}$ von S_i durch Ersetzung der nichtterminalen Zeichen A durch Wort α, wobei $A \to \alpha$ eine Produktion ist.

Kurzschreibweise: $S \stackrel{*}{\Rightarrow} S_k$

$S \stackrel{*}{\Rightarrow} S_k$ heißt *Ableitung*.

Sei G eine kontextfreie Grammatik, dann

- $L_G = \{w \mid S \stackrel{*}{\Rightarrow} w\}$
- L_G ist eine kontextfreie Sprache.

Beispiel:

- Arithmetische Ausdrücke bilden eine kontextfreie Sprache.
- Programme, in denen jede Variable vor dem Gebrauch deklariert wird, sind nicht kontextfrei.
  Grund: $L = \{w\#w \mid w \in \{a,b\}^*\}$ ist nicht kontextfrei.
- Programme, in denen Prozeduren definiert und benutzt werden, sind auch nicht kontextfrei.
  Grund: Die Anzahl von Argumenten muß immer gleich sein und
  $L = \{ \underbrace{a^n b^m}_{\text{Definition von Prozeduren}} \underbrace{c^n d^m}_{\text{Aufruf von Prozeduren}} \mid n, m \geq 1\}$ ist nicht kontextfrei.

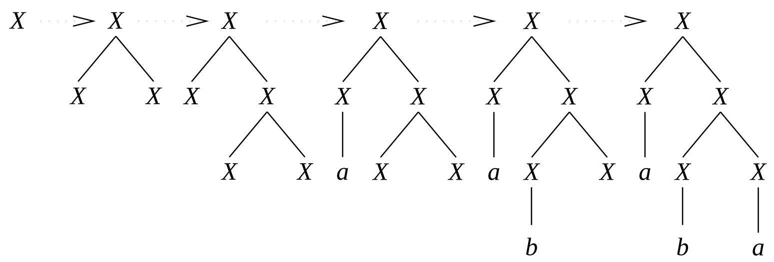
  $a, c \leadsto$ Prozedur 1

  $b, d \leadsto$ Prozedur 2

Ableitungsbäume:

Grammatik: $X \to a, X \to b, X \to XX$

Ableitung: $X \Rightarrow XX \Rightarrow XXX \Rightarrow aXX \Rightarrow abX \Rightarrow aba$



Bemerkung:

- Es gibt mehrere Ableitungsbäume, die zu aba führen.
- Aus dem letzten Ableitungsbaum kann man nicht immer die ganze Ableitung rekonstruieren.

Definition:

- *Linksableitung*:
  Es wird immer das am weitesten links stehende Nichtterminal ersetzt.
- *Rechtsableitung*:
  Es wird immer das am weitesten rechts stehende Nichtterminal ersetzt.

Definition:

Eine Grammatik G heißt *mehrdeutig*, falls $\exists w,\ w \in L_G$, mit w besitzt 2 verschiedene Linksableitungen. Andernfalls ist G *eindeutig*.

Bemerkung:

Eine Grammatik sollte besser eindeutig sein. (Hinter dem Ableitungsbaum steckt die Bedeutung des Programms.)

Beispiel:

$X \to Z,\ X \to a,\ X \to b,\ X \to X,\ Z \to b$

kürzere Notation: $X \to Z \mid a \mid b \mid X,\ Z \to b$

$X \Rightarrow Z \Rightarrow b$

$X \overset{*}{\Rightarrow} b$

Beispiel:

$$L = \{\overbrace{a^n b^n c^m d^m \mid n, m \geq 1}^{1.)}\} \cup \{\overbrace{a^n b^m c^m d^n \mid n, m \geq 1}^{2.)}\}$$

1.) $S \to XY,\ X \to aXb \mid ab,\ Y \to cYd \mid cd$

2.) $S \to M,\ M \to aMd \mid Z,\ Z \to bZc \mid bc$

Beide Grammatiken erzeugen u.a. auch: $a^n b^n c^n d^n$.

Frage: Existiert eine eindeutige Grammatik für L?

Antwort: Nein!

Man zeigt: fast alle Worte $a^n b^n c^n d^n$ $(n \in \mathbb{N})$ haben
2 verschiedene Ableitungen
(mit Grammatik für L ist beliebig).

- für jedes Nichtterminalzeichen $A : \exists x_1 x_2 : A \overset{*}{\Rightarrow} x_1 A x_2$ (Man kann jede Grammatik so modifizieren.)

  $A \to \alpha_1 \mid \alpha_2 \mid \ldots \mid \alpha_k$ Produktionen mit A auf der linken Seite

  $\ldots \to \ldots A \ldots$

  $\uparrow$

  wir ersetzen A mit $\alpha_1, \alpha_2, \ldots, \alpha_k$

- $A \Rightarrow x_1 A x_2$ dann:

  1.)

  - $x_1 \in a^* + b^* + c^* + d^*$

    z.B.: $a, b \in x_1$

    $\ldots a \ldots b \ldots A \ldots$

    $\updownarrow$

    $x_1 A x_2 \Rightarrow \ldots a \ldots b \ldots a \ldots b \ldots A \ldots$

  - $x_2 \in a^* + b^* + c^* + d^*$

  2.)

  - $x_1 \in a^* \Rightarrow x_2 \in b^* \vee x_2 \in d^*$

    Skizze: $x_1 = a \ldots a$

    $A \overset{*}{\Rightarrow} a \ldots a A a \ldots a$

    Ableitung: $S \to \ldots \to \underline{\ldots} A \underline{\ldots} \to \ldots \to w$

    $\Downarrow$

    $\underline{\ldots} a \ldots a A a \ldots a \underline{\ldots}$

    Hier werden nur a's gepumpt, keine d's.

  - $x_1 \in b^* \Rightarrow x_2 \in c^*$
  - $x_1 \in c^* \Rightarrow x_2 \in d^*$

  3.) $|x_1| = |x_2|$ $\qquad$ $\_\_\underbrace{a \ldots a}_{viel} A \underbrace{b \ldots b}_{wenig}\_\_$

- $\left.\begin{array}{l} A \overset{*}{\Rightarrow} x_1 A x_2 \\ A \overset{*}{\Rightarrow} x_3 A x_4 \end{array}\right\}$ dann enthalten sowohl x_1 und x_3, als auch x_2 und x_4, dieselben Zeichen.

  $\Rightarrow \ldots A \ldots \Rightarrow \ldots x_1 A x_2 \ldots \Rightarrow \ldots x_1 x_3 A x_4 x_2 \ldots \Rightarrow$

  $\ldots \underbrace{x_1 x_3 x_1}_{} A x_2 x_4 x_2 \ldots \notin L$

  zwei Blöcke von Zeichen aus x_1

  Also: Für jedes Nichtterminalsymbol definieren wir den Typ:

  z.B. $A \in C_{ab}$, falls $A \overset{*}{\Rightarrow} x_1 A x_2, x_1 \in a^*, x_2 \in b^*$

- Jede Ableitung enthält <u>entweder</u> Zeichen aus C_{ab} und C_{cd} <u>oder</u> Zeichen aus C_{ad} und C_{bc}.

  Grund:

  Ann.: $X \in C_{ab}$, $Y \in C_{ad}$ kommt in einer Ableitung gleichzeitig vor.

  $$\ldots \Rightarrow \ldots X \ldots \Rightarrow \ldots \_\_ Y \Rightarrow \ldots \Rightarrow w$$

- $X \Rightarrow \alpha_1 X \alpha_2$

  Wir vergößern die Anzahl von a, b, d; die Anzahl von c bleibt unverändert.

Korollar:

Gibt es „Subgrammatiken" für $\{a^n b^n c^m d^m \mid n, m \in \mathbb{N}\}$ und $\{a^n b^m c^m d^n \mid n, m \in \mathbb{N}\}$?

Man zeigt, daß jede Subgrammatik fast alle Worte $\{a^n b^n c^n d^n \mid n \in \mathbb{N}\}$ erzeugt.

(Intuition: KfG kann nicht n und m zählen.)

2.1 Normalformen

Definition: Nutzlose Symbole

X ist *nützlich*, falls gilt: $S \Rightarrow ... \Rightarrow \alpha X \beta \Rightarrow ... \Rightarrow w$ und w besteht nur aus Terminalsymbolen.

Satz:

Für jede kontextfreie Grammatik G gibt es eine kontextfreie Grammatik G' mit $L_G = L_{G'}$ und G' enthält keine nutzlosen Symbole.

Beweis:

Konstruktiv: Algorithmus erzeugt G' aus G.

1. Zeichen A entfernen, falls $\neg \exists w \in T^* : A \overset{*}{\Rightarrow} w$
   (Wir berechnen alle A, die nicht zu entfernen sind:
   a) $X \rightarrow p \quad (p_i \notin V)$ Produktion, dann $X \in$ NUTZBAR
   b) $X \rightarrow q \quad (q_i \in T \cup$ Zeichen aus NUTZBAR), dann NUTZBAR = NUTZBAR $\cup \{X\}$)
2. Zeichen A entfernen, falls: $\neg \exists x_1, x_2 : S \overset{*}{\Rightarrow} x_1 A x_2$.
   (Algorithmus: Ähnlich, erreichbare Zeichen berechnen)

Definition:

Eine Produktion der Form $A \rightarrow \varepsilon$, wobei ε das leere Wort bezeichnet, heißt ε–Produktion.

Satz:

Falls $\varepsilon \notin L_G$, dann gibt es eine kfG G', die L_G erzeugt und G' besitzt keine ε–Produktionen.

Beweis:

Produktion: $B \rightarrow X_1 \ldots X_n$
einige X_i's haben die Eigenschaft: $X_i \overset{*}{\Rightarrow} \varepsilon$.

Neue Produktionen: $B \rightarrow \underbrace{\qquad}_{\substack{\text{jede nichtleere Teilfolge von } X_1 \ldots X_n, \\ \text{die alle } X_i \text{ enthalten, so daß } \neg(X_i \overset{*}{\Rightarrow} \varepsilon).}}$

G' erzeugt dieselbe Sprache.

Unit–Produktion:

$X \rightarrow Y$

Elimination von Unit–Produktionen:

$X \rightarrow Y$ weg
$Y \rightarrow \alpha$ ersetzen durch $X \rightarrow \alpha$

Satz: Chomsky–Normalform

Eine kfG ist in Chomsky–Normalform, wenn alle Produktionen die Form
1) $A \to \alpha \quad (A \in V, \alpha \in T)$ oder
2) $A \to BC \quad (A, B, C \in V)$
haben.

Satz:

KfG G , $\varepsilon \notin L_G$, dann

$\exists\, G'$ in CNF mit $L_G = L_{G'}$.

Beweis:

$G \to G''$ ohne ε–Produktionen und Unit–Produktionen.

Produktion von G' aus G'':

1.Schritt:

$A \to X_1 \ldots X_m$

Falls $X_i = a \quad (a \in T^*)$ ersetze X_i durch $C_A \quad (\in V^*)$, noch dazu:

$C_a \to a$.

2.Schritt:

$A \to Y_1 \ldots Y_m$ ersetzen durch:

$A \to Y_1 D_1,\ D_1 \to Y_2 D_2, \ldots, D_{m-2} \to Y_{m-1} Y_m$

$D_1, \ldots, D_{m-2}$ – neue Nichtterminalsymbole

Definition: Greibach–Normalform

Eine kfG ist in Greibach–Normalform, wenn alle Produktionen die Form $A \to a\alpha \quad (a \in T, \alpha \in V^*)$ haben.

Satz:

KfG G , $\varepsilon \notin L_G$, dann

$\exists\, G'$ in GNF mit $L_G = L_{G'}$.

Lemma: (1)

$A \to \alpha_1 B \alpha_2$ – Produktion von G

$B \to \beta_1,\ B \to \beta_2, \ldots, B \to \beta_r$ – alle Produktionen aus G mit B auf der linken Seite.

$$\Downarrow$$

$A \to \alpha_1 B \alpha_2$ kann man durch

$A \to \alpha_1\beta_1\alpha_2 \,|\, \alpha_1\beta_2\alpha_2 \,|\, \ldots \,|\, \alpha_1\beta_r\alpha_2$ ersetzen.

Lemma: (2)

Produktionen mit A:

$A \to A\alpha_1 \,|\, A\alpha_2 \,|\, \ldots \,|\, A\alpha_r$

$A \to \beta_1 \,|\, \ldots \,|\, \beta_s \quad$ (β_i enthält keine A auf der 1. Position links.)

Kann man ersetzen durch:

$A \to \beta_i;\ A \to \beta_i B,\ i \le s$

$B \to \alpha_i;\ B \to \alpha_i B,\ i \le r$

Beweis:

G – alte Grammatik, G' – neue Grammatik

Linksableitung für G:

$$\ldots \Rightarrow Ax \Rightarrow A\alpha_{i_1}x \Rightarrow A\alpha_{i_2}\alpha_{i_1}x \Rightarrow \ldots \Rightarrow A\alpha_{i_p}\ldots\alpha_{i_1}x$$
$$\Rightarrow \beta_j\alpha_{i_p}\ldots\alpha_{i_1}x \Rightarrow \ldots$$

Linksableitung für G':

$$\ldots \Rightarrow Ax \Rightarrow \beta_j Bx \Rightarrow \beta_j\alpha_{i_p}Bx \Rightarrow \beta_j\alpha_{i_p}\alpha_{i_{p-1}}Bx \Rightarrow \ldots$$
$$\Rightarrow \beta_j\alpha_{i_p}\ldots\alpha_{i_2}Bx \Rightarrow \beta_j\alpha_{i_p}\ldots\alpha_{i_1}x \Rightarrow \ldots$$

also: $L_G \subseteq L'_G$

$L'_G \subseteq L_G$ beweist man ähnlich.

Konstruktion der GNF

$A_1 \ldots A_m$ – alleNichtterminalzeichen von G

- Modifikation:

  $A_i \to A_j$ dann $j > i$

  Konstruktion: Induktion für i

$i = 0: \surd$

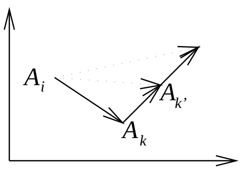
$i > 0:$ $A_i \to A_k\beta$, $k < i$ $\quad A_k \Rightarrow A_{k'}\beta'$ denn $k' > k$

nach Lemma 1 ersetzt man $A_i \to A_k\beta$ durch $A_i \to A_{k'}\beta'\beta$.

Falls $k' < i$, dann wiederholen wir die letzten Schritte bis $k' \geq i$ ist.

Weitere Modifikationen notwendig, falls $k' = i$.

$A_i \to A_i\gamma$.



Lemma 2 anwenden, β_i eingeführt.

- Wir haben Produktionen der Form:

  $A_i \to A_j\gamma \quad j > i$

  $A_i \to a\gamma \quad a$ – Terminalsymbol

  $\beta_i \to \gamma$

  Jedes $A_i \to A_j\beta$ wird modifiziert mit Hilfe von Lemma 1.

  Man bekommt Produktionen der Form:

  $A_i \to A_{j'}\beta' \quad j' > j$

  Diese Prozedur wird wiederholt, bis keine Wiederholung möglich ist. Danach ist keine Produktion des Typs $A_i \to A_j\beta$ möglich.

  Produktionen mit B_i's:

  $B_i \to \alpha_i$ durch Ersetzung von $A_i \to A_i\alpha_i$ $(\alpha_i \neq \varepsilon)$ eingeführt.

  - α_i fängt mit Terminalzeichen an – o.k.
  - α_i fängt mit A_j, $j \leq n$ an.

    Mit Hilfe von Lemma 1 kann man A_j eleminieren.

2.2 Kellerautomaten

Bemerkung:

Grammatik – erzeugt Sprache, keine Antwort für „$w \in L_G$?“.

Definition: Kellerautomat

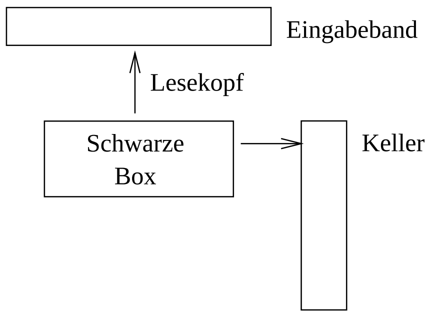
Eine Kellerautomat wird durch das Tupel $K = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ definiert, wobei

- Q = endliche Menge von Zuständen
- Σ = Eingabealphabet
- Γ = Kelleralphabet
- δ = Übergangsfunktion
- q_0 = Startzustand
- Z_0 = Symbol, das den Kellerboden markiert
- F = Menge der akzeptierenden Zuständen ($F \subset Q$)

Die Übergangsfunktion δ:

- $\delta : Q \times (\Sigma \cup \{\varepsilon\}) \times \Gamma \rightarrow$ endliche Teilmenge von $Q \times \Gamma^*$
- $(p, \gamma) \in \delta(q, a, Z) \Rightarrow$ Kellerautomat in Zustand q geht nach dem Lesen von a in Zustand p über und ersetzt (den Kellerinhalt) Z durch γ.

Kellerautomaten sind nichtdeterministisch. Der Lesekopf bewegt sich von links nach rechts, darf aber auch anhalten.



Ein Kellerautomat akzeptiert, falls gilt:

- Keller ist leer. (Definition 1)
- Der Automat befindet sich nach Lesen des letzten Buchstabens (auf dem Band) in akzeptierendem Endzustand. (Definiton 2)

Beide Definitionen sind äquivalent! (s. Übung)

Definition: Deterministischer Kellerautomat

Ein Kellerautomat ist deterministisch, wenn gilt:

1. $\forall q \in Q, \forall Z \in \Gamma : \delta(q, \varepsilon, Z) \neq \varnothing \Rightarrow \forall a \in \Sigma : \delta(q, a, Z) = \varnothing$
2. $\forall q \in Q, \forall Z \in \Gamma, \forall a \in \Sigma \cup \{\varepsilon\} : |\delta(q, a, Z)| \leq 1$

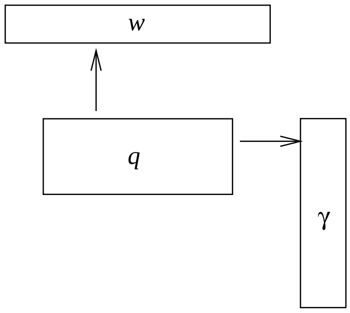
Definition: Deterministisch kontextfreie Sprache

Deterministisch kontextfreie Sprache = Sprache, die durch einen deterministischen Kellerautomaten akzeptiert wird.

Definition: Instantaneous description

Instantaneous description = (q, w, γ) mit

- q = Zustand
- w = Teil des Eingabewortes auf der rechten Seite des Lesekopfes
- γ = Inhalt des Kellers



(Instantaneous description = Momentaufnahme, Konfiguration zu einem bestimmten Zeitpunkt.)

Satz:

G kfG $\Rightarrow \exists$ ein Kellerautomat, der L_G akzeptiert

<u>Beweis:</u>

<u>Ann:</u> $\varepsilon \notin L_G$ (Übung: Ergänzen Sie den Beweis für $\varepsilon \in L_G$)

<u>Ann:</u> G in GNF.

$$\begin{aligned} G &= (V, T, P, S) \\ V &= \text{Menge der Nichtterminalzeichen} \\ T &= \text{Menge der Terminalzeichen} \\ P &= \text{Menge der Produktionen} \\ S &= \text{Startsymbol} \end{aligned}$$

$$\begin{aligned} M = &\ (\{\,q\,\}, T, T \cup V, \delta, q, S, \{\,q\,\}) \\ &\ \delta(q, a, A) \text{ enthält } (q, \gamma), \text{ falls } A \to a\gamma \in P. \end{aligned}$$

<u>Claim:</u>

$S \stackrel{*}{\Rightarrow} x\alpha$ durch Linksableitung

$\Updownarrow$

$(q, x, S) \stackrel{*}{\vdash} (q, \varepsilon, \alpha)$

Definition:

Wir schreiben: $(q, x, S) \vdash (q', x', S')$, falls wir in einem Schritt mit dem Kellerautomaten von der Situation (q, x, S) nach (q', x', S') gelangen können.

$\stackrel{*}{\vdash}$ bedeutet Transitiver Abschluß von $\vdash$.

<u>Beweis:</u> (von Claim)

Induktion nach der Länge der Ableitung bzw. Länge der Berechnung

($\Uparrow$) $(q,x,S) \overset{i}{\vdash} (q,\varepsilon,\alpha)$

$(q,x,S) \overset{i-1}{\vdash} (q,a,\beta) \vdash (q,\varepsilon,\alpha) \qquad x = ya$

Aus $(q,ya,S) \overset{i-1}{\vdash} (q,a,\beta)$ folgt, daß $(q,y,S) \overset{i-1}{\vdash} (q,\varepsilon,\beta)$ gilt.
Und daraus folgt nach Induktionsannahme: $S \overset{*}{\Rightarrow} y\beta$
Wegen der Definition impliziert der letzte Schritt
$(q,a,\beta) \vdash (q,\varepsilon,\alpha)$, daß $\beta = A\gamma$, $A \to a\eta$ und $\alpha = \eta\gamma$ gilt.
Also gilt: $S \overset{*}{\Rightarrow} y\beta = yA\gamma \Rightarrow \underbrace{ya}_{x}\underbrace{\eta\gamma}_{\alpha} = x\alpha$.

($\Downarrow$) $S \overset{i}{\Rightarrow} x\alpha$

$S \overset{i-1}{\Rightarrow} yA\gamma \Rightarrow \underbrace{ya}_{x}\underbrace{\eta\gamma}_{\alpha}$

$(q,y,S) \overset{*}{\vdash} (q,\varepsilon,A\gamma)$ nach Induktionsannahme
Also $(q,ya,S) \overset{*}{\vdash} (q,a,A\gamma)$
Da $A \to a\eta$ eine Produktion ist, ist (q,η) nach Definition in $\delta(q,a,A)$ enthalten.
Hieraus folgt: $(q,a,A\gamma) \vdash (q,\varepsilon,\underset{\eta\gamma}{\overset{\|}{\alpha}})$
Also gilt: $(q,x,S) \overset{*}{\vdash} (q,a,A\gamma) \vdash (q,\varepsilon,\alpha)$

$S \overset{*}{\Rightarrow} x \;\Leftrightarrow\; (q,x,S) \overset{*}{\vdash} (q,\varepsilon,\varepsilon)$

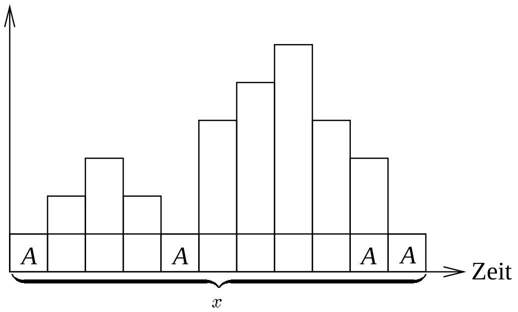
Satz:

Kellerautomat M akzeptiert Sprache $L \Leftrightarrow \exists$ kfG G mit $L = L_G$.

Beweis:

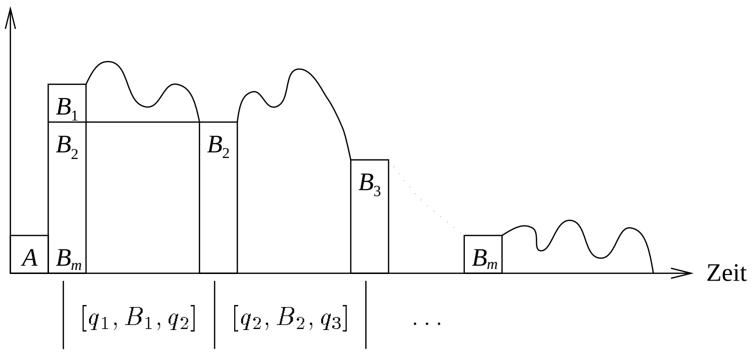
Nichtterminalzeichen von G: $[q,A,p]$ (q,p – Zustände, A – Kellerzeichen)

Bedeutung: $[q,A,p]$ erzeugt alle Worte x, so daß beim Lesen von x (Anfangszustand $= q$, Endzustand $= p$) A vom Keller entfernt wird (inzwischen dürfen andere Zeichen über A gespeichert werden, aber A ist immer im Keller).



Produktionen von G:

1.) $S \to [q_0, Z_0, q]$, q, q_0 beliebig, Z_0 Bodenzeichen des Kellers
2.) $\delta(q,a,A)$ enthält $(q_1, B_1 \dots B_m)$



$[q, A, q_{m+1}] \to a[q_1, B_1, q_2]\,[q_2, B_2, q_3] \ldots [q_m, B_m, q_{m+1}]$
$(q_1, \varepsilon) \in \delta(q, a, A)$
$[q, A, q_1] \to a$

<u>Claim:</u>

$$[q, A, p] \underset{\substack{\uparrow \\ \text{Grammatik}}}{\overset{*}{\Rightarrow}} x \quad (x \in \Sigma^*)$$

$$\Updownarrow$$

$$(q, x, A) \underset{\substack{\uparrow \\ \text{Kellerautomat}}}{\overset{*}{\vdash}} (p, \varepsilon, \varepsilon)$$

<u>Bew:</u> (von Claim)

($\Downarrow$) Induktion über die Länge der Ableitung:

$[q, A, p] \Rightarrow a[q_1, B_1, q_2] \ldots [q_n, B_n, q_{n+1}] \overset{i-1}{\Rightarrow} x$

$x = ax_1 \ldots x_n$ und $[q_i, B_i, q_{i+1}] \overset{*}{\Rightarrow} x_i$

Also gilt nach IA: $(q_i, x_i, B_i) \overset{*}{\vdash} (q_{i+1}, \varepsilon, \varepsilon)$

$(q_i, x_i \underbrace{\phantom{x_{i+1} \ldots x_n}}_{x_{i+1} \ldots x_n}, B_i, B_{i+1} \ldots B_n) \overset{*}{\vdash} (q_{i+1}, \varepsilon \underbrace{\phantom{x_{i+1} \ldots x_n}}_{x_{i+1} \ldots x_n}, B_{i+1} \ldots B_n)$

$$\begin{aligned}
(q, x, A) \ &\vdash (q_1, x_1 \ldots x_n,\ B_1 \ldots B_n) \\
&\overset{*}{\vdash} (q_2, x_2 \ldots x_n,\ B_2 \ldots B_n) \\
&\overset{*}{\vdash} (q_3, x_3 \ldots x_n,\ B_3 \ldots B_n) \\
&\overset{*}{\vdash} \ldots \\
&\overset{*}{\vdash} (q_{n+1}, \varepsilon, \varepsilon) = (p, \varepsilon, \varepsilon)
\end{aligned}$$

($\Uparrow$) Ähnlich.

Aus dem Claim folgt:

$$\begin{aligned}
x \in L_G \quad &\Leftrightarrow \quad S \overset{*}{\Rightarrow} x \\
&\Leftrightarrow \quad S \Rightarrow [q_0, Z_0, p] \wedge [q_0, Z_0, p] \overset{*}{\Rightarrow} x \\
&\overset{\text{Claim}}{\Leftrightarrow} \quad (q_0, x, Z_0) \overset{*}{\vdash} (p, \varepsilon, \varepsilon) \\
&\Leftrightarrow \quad M \text{ akzeptiert } x
\end{aligned}$$

2.3 Eigenschaften von kontextfreien Sprachen

Satz: Pumping–Lemma

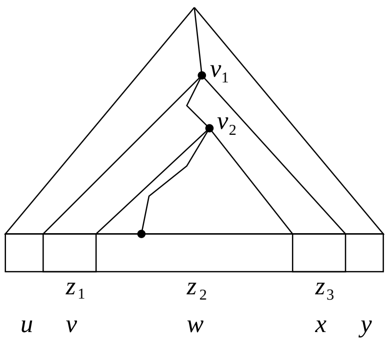
Sei L eine kontextfreie Sprache. Dann gibt es für L eine Konstante $n \in \mathbb{N}$, so daß sich alle Wörter $z \in L$ mit $|z| \geq n$ zerlegen lassen in $z = uvwxy$ mit folgenden Eigenschaften:

1. $|vx| \geq 1$
2. $|vwx| \leq n$
3. Für alle $i \geq 0$ gilt: $uv^iwx^iy \in L$

Beweis:

Sei G Grammatik für L und G in Chomsky–NF.

Ableitungsbaum für z:



Sei i die Tiefe des Baumes $\Rightarrow$ Anzahl der Blätter $\leq 2^i$

k = Anzahl der Nichtterminalsymbole in G und $|z| > 2^k$, denn $2^k \stackrel{\text{Def}}{=} n - 1$.

Es gibt einen Pfad von der Wurzel bis zu einem Blatt der Länge $\geq k+1$, so daß v_1, v_2 demselben Nichtterminalzeichen A entsprechen.

$A \to BC$ folgt: v_1 ist unter B oder C.

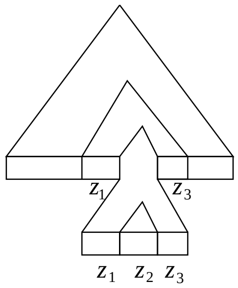
$A \stackrel{*}{\Rightarrow} z_1z_2z_3 \qquad A \Rightarrow z_1Az_3 \Rightarrow z_1z_1Az_3z_3 \Rightarrow \ldots \Rightarrow z_1^iAz_3^i \Rightarrow z_1^iz_2z_3^i$

$A \stackrel{*}{\Rightarrow} z_2$

$A \stackrel{*}{\Rightarrow} z_1^iz_2z_3^i$

1.) z_1, z_2 nicht beide leer:
z.B. falls v_1 unter C : dann alles was von B erzeugt wird gehört zu z_1, deswegen $z_1 \neq \varepsilon$.

2.) $uz_1^iz_2z_3^iy \in L$



3.) $|vwx| \leq n$

v_1 – Wurzel des Baumes mit Tiefe $k+1 \Rightarrow$ mit $2^{k+1} = n$ Blättern

Beispiel:

$$L = \{\, a^i b^j c^k d^l \mid i = 0 \vee j = k = l\}$$

$z \in L$

$$\begin{array}{lll} z = & z_1 z_2 z_3 z_4 z_5 & b^j c^k d^l \\ & z_1 z_2^i z_3 z_4^i z_5 \in L & z = a^i \underline{b^j c^j d^j} \end{array}$$

Satz: Ogden–Lemma

Sei L eine kontextfreie Sprache. Dann gibt es eine Konstante n, so daß wir — wenn $z \in L$ und wir n oder mehr Positionen in z markieren können ($|z| \geq n$) — $z = uvwxy$ schreiben können, wobei folgendes gilt:

1. v und x haben zusammen mindestens eine markierte Position.
2. vwx hat höchstens n markierte Positionen.
3. Für alle $i \geq 0$ gilt: $uv^i wx^i y \in L$

Beispiel:

$L = \{\, a^i b^j c^k \mid i, j, k -$ verschiedene Zahlen$\}$

$z = a^n b^{n+m!} c^{n+2n!}$

Wir markieren alle a's: $z = uvwxy$

- v oder x enthält ein a.
- v oder x enthalten 2 verschiedene Zeichen
  $\Rightarrow uv^2wx^2y \notin L$.
- v oder x ist ein Block von a's.
  $x \in b^* + c^* \Rightarrow v \in a^*$
  $x \in a^+ \Rightarrow v \in a^*$

  Fall: $x \in b^*$ $p = |v|$, $p \leq |n|$
  $\exists q : n! = p \cdot q$
  $z' = uv^{2q+1}y$
  v^{2q+1} enthält $p \cdot (2q+1) = 2n! + p$ a's
  $$z' = \underbrace{uv^{2q+1}w}_{\substack{\text{gibt hier} \\ (n-p)+2n!+p \\ =n+2n!}} \; x^{2q+1}y$$
  alle c's in z' kommen aus y und es gibt $2n! + n$ viele

2.4 Abgeschlossenheit bei Boolschen Operationen

Bemerkung:

Kontextfreie Sprachen sind bzgl. $+, \cdot, *$ abgeschlossen.

Beweis:

$G_1 = (V_1, T_1, P_1, S_1)$

$G_2 = (V_2, T_2, P_2, S_2)$

„+“ $G_3 = (V_1 \cup V_2 \cup \{S_3\}, T_1 \cup T_2, P_3, S_3)$

$P_3 = P_1 \cup P_2 \cup \{S_3 \to S_1 \mid S_2\}$

„·“ $S_3 \to S_1 S_2$

„*“ $S \to \varepsilon \mid SS \mid S_1$

Satz:

Es existieren kfS L, so daß $\Sigma^* \setminus L$ nicht kontextfrei ist.

<u>Beispiel:</u>

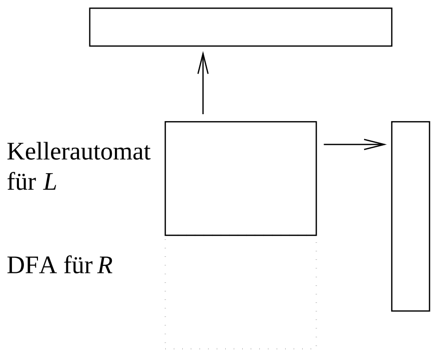
$$\begin{aligned} L_1 &= \{\, a^i b^j c^k \mid i = j \vee i = k \vee j = k \,\} \\ &= \{\, a^i b^i c^k \mid i, k \in \mathbb{N} \,\} \cup \{\, a^i b^j c^j \mid i, j \in \mathbb{N} \,\} \cup \{\, a^i b^j c^i \mid i, j \in \mathbb{N} \,\} \end{aligned}$$

$$\left.\begin{array}{ll} (\Sigma^* \setminus L_1) \cap a^* b^* c^* = L & \notin \text{ kfS} \\ (\Sigma^* \setminus L_1) & \in \text{ kfS} \end{array}\right\} \Rightarrow (\Sigma^* \setminus L_1) \cap a^* b^* c^* \in \text{ kfS!?}$$

Satz: (Hilfssatz)

$L \in$ kfS, R regulär, dann $L \cap R \in$ kfS.

<u>Beweis:</u>



Satz:

Es gibt kfS L_1, L_2, so daß $L_1 \cap L_2 \notin$ kfS.

<u>Beweis:</u>

$L_1 = \{\, a^i b^i c^k \mid i, k \in \mathbb{N} \,\}$

$L_2 = \{\, a^i b^k c^k \mid i, k \in \mathbb{N} \,\}$

$L_1 \cap L_2 = \{\, a^i b^i c^i \mid i \in \mathbb{N} \,\}$ nicht kontextfrei (Ogden's Lemma)

$\neg(\neg L_1 \cup \neg L_2) = L_1 \cap L_2$

Satz:

Die kontextfreien Sprachen sind bzgl. Substitution abgeschlossen.

<u>Beweis:</u>

Def. $abcd \in L$

$C(a)$ – eine Sprache

G – ursprüngliche Grammatik

Für $a \in \Sigma$ haben wir G_a – kfG für $C(a)$

<u>Neue Grammatik:</u>

- $a \in \Sigma$, in allen Produktionen von G ersetzen wir a durch C_a (Nichtterminalzeichen)
- In G_a Anfangszeichen durch C_a ersetzen.

Satz:

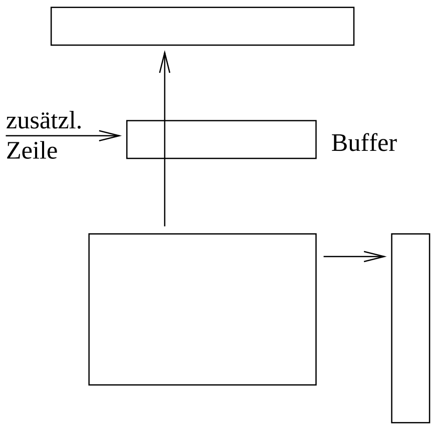
Die kontextfreien Sprachen sind bzgl. inversem Homomorphismus abgeschlossen.

Beweis:

Sei L kfS und sei h ein Homomorphismus.

$h^{-1}(L) = \{ w \mid h(w) \in L \}$

Dann ist M der Kellerautomat, der L akzeptiert.



a) Nach dem Lesen von einem Zeichen x schreiben wir $h(x)$ in den Buffer.

b) Simulation von M (M liest Buffer statt Eingabeband) bis Buffer leer ist.

c) gehe zu a)

Satz:

Es gibt keine Algorithmen, die folgende Fragen beantworten:

1.)	Eingabe:	G, G' kfG
	Frage:	$L_G = L_{G'}$?
2.)	Eingabe:	G kfG
	Frage:	Ist $\Sigma^* \setminus L_G$ endlich?
3.)	Eingabe:	G kfG
	Frage:	Ist G eindeutig?

Satz:

Es gibt Algoritmen für folgende Probleme:

Eingabe: G kfG

Fragen:

1.) Frage: Ist L_G endlich?
2.) Frage: Ist L_G leer?

Bemerkung:

Diese Algorithmen sind praktisch.

Bemerkung:

Z ist nützlich, falls $\exists\, \alpha, \beta, w : S \stackrel{*}{\Rightarrow} \alpha Z \beta \stackrel{*}{\Rightarrow} w$.

(w besteht nur aus Terminalsymbolen.)

Hilfssatz:

$\forall$ kfG G $\exists$ kfG G' :

$L_{G'} = L_G \setminus \{\varepsilon\}$ und in G' sind keine nutzlosen Symbole.

Beweis:

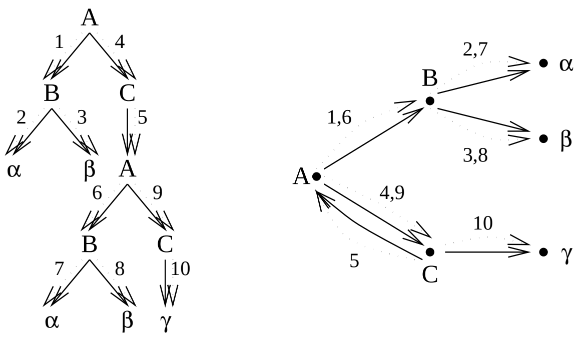
Algorithmus: Wir bauen einen Graphen (gerichtet)!

Knoten: Nichtterminalzeichen von G

Kanten: Falls $A \to BC$ eine Produktion ist, dann gibt es folgende Kante: $[A, B], [A, C]$

Ableitung $\leftrightsquigarrow$ Baum eingebettet in dem Graphen (jede Kante darf mehrmals benutz werden.)

$$A \to BC \to \alpha\beta C \to \alpha\beta A \to \alpha\beta BC \to \alpha\beta\alpha\beta C \to \alpha\beta\alpha\beta\gamma$$



Fall 1: Graph ist azyklisch:

$\Rightarrow$ Dann gibt es nur endlich viele Einbettungen.

$\Rightarrow$ Es gibt endlich viele Ableitungen.

$\Rightarrow$ Es gibt endlich viele Worte.

Fall 2: Es gibt einen Zyklus:

$A \stackrel{*}{\Rightarrow} \gamma A \eta$ – γ, η sind nicht gleichzeitig leer.

$A \to A$

$A \stackrel{*}{\Rightarrow} A$

$S \stackrel{*}{\Rightarrow} \alpha A \beta \Rightarrow \overline{\alpha}\,\overline{w}\,\overline{\beta}$ (Terminalsymbole)

$S \stackrel{*}{\Rightarrow} \alpha A \beta \stackrel{*}{\Rightarrow} \alpha \gamma^i A \eta^i \beta \Rightarrow \overline{\alpha}\,\overline{\gamma}^i\, \underset{\substack{\uparrow \\ bel.lang}}{\overline{w}}\, \overline{\eta}^i\, \overline{\beta}$

Lemma:

Für jedes Nichtterminalsymbol Z existiert ein w aus Terminalsymbolen, so daß gilt: $Z \stackrel{*}{\Rightarrow} w$ und $S \stackrel{*}{\Rightarrow} \alpha Z \beta \stackrel{*}{\Rightarrow} \overline{\alpha} w \overline{\beta}$.

Satz: Membership

Eingabe:	G — Grammatik, x — Wort
Frage:	$x \in L_G$?

Beweis:

Lösung 1:

$G \rightsquigarrow G'$ (äquivalent Grammatik in GNF)

$A \to \underset{\substack{\uparrow \\ n.t.Z.}}{a} \rightsquigarrow$ jede Ableitung von x hat Länge $\leq |x|$

Man kann alle Ableitungen der Länge $|x|$ überlegen.

Zeitkomplexität dieser Methode $\geq 2^{|x|}$.

Lösung 2:

dynamische Programmierung: CYK–Algorithmus, Laufzeit $\approx |x|^3$

$x_{i,j}$ — Teilwort von x der Länge j ab Position i.

$$\underbrace{\boxed{\phantom{x_{i,j}}}}_{i}\underbrace{\boxed{x_{i,j}}}_{j}\boxed{\phantom{x_{i,j}}}$$

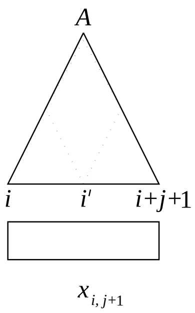
Prinzip:

Durch Induktion nach j berechnen wir für jedes A, ob:
$A \overset{*}{\Rightarrow} x_{i,j}$

Berechnung:

$j = 1$: Wir suchen für jedes Zeichen a alle A, so daß
$A \overset{*}{\Rightarrow} a$. (CNF verwenden)

$j + 1$: $A \overset{*}{\Rightarrow} x_{i,j+1}$, falls $\exists A \to BC$ – eine Produktion und
$\exists i' : B \overset{*}{\Rightarrow} x_{i,i'-i}$ und $C \overset{*}{\Rightarrow} x_{i,i+j+1-i'}$



3 Parsing

Bemerkung:

Parser für Grammatik G

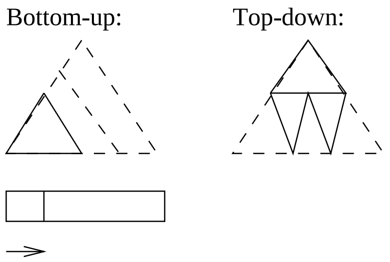
Eingabe:	Wort w	
Ausgabe:	Ableitungsbaum für w	(falls $w \in L_G$)
	Fehlermeldung	(falls $w \notin L_G$)

Bemerkung:

- Kellerautomaten sind nicht geeignet (da ndet.)
- deterministische Kellerautomaten sind gut geeignet
  - + Laufzeit ist linear und proportional zur Eingabelänge
  - − nicht alle kfS können durch dKA erkannt werden
  - − Zusammenhang zwischen Grammatiken und dKA noch unklar (für uns)

Bemerkung:

Parserarten:
- Bottom–Up: Ableitungsbaum von unten nach oben bauen
- Top–Down: umgekehrt



3.1 Shift–Reduce–Parsing

Beispiel:

Produktionen:	$S \to aAcBe, A \to Ab \mid b, B \to d$
Eingabewort:	$abbcde$
Ableitung:	$abbcde \Leftarrow aAbcde \Leftarrow aAcde \Leftarrow aAcBe \Leftarrow S$
Aber auch:	$abbcde \Leftarrow aAbcde \Leftarrow aAAcde$ (hier Abbruch!)

Die zweite Ableitung läßt sich nicht vervollständigen, weil bei Ableitung von S genau ein A erzeugt wird.

Problem:

Viele Teilworte entsprechen der rechten Seite der Produktionen.

Welche sollen wir wählen ?

(Alle können nicht gewählt werden, da dann zuviel Zeit benötigt würde und der Aufwand nicht mehr linear wäre.)

Definition:

- *Rechtssatzform* – ein Wort, das während der Rechtsableitung vorkommt.

- *Handle* – Sei γ eine Rechtssatzform. Ein Handle für γ ist eine Produktion $A \to \beta$ zusammen mit einer Position in γ, so daß gilt

$$\exists\, \alpha, w \text{ mit } S \overset{*}{\underset{\text{r}}{\Rightarrow}} \alpha A w \underset{\text{r}}{\Rightarrow} \underbrace{\alpha\beta w}_{\gamma}$$

(Das r steht für Rechtsableitung.)

Bemerkung:

Bei eindeutiger Grammatik: Jede Rechtssatzform enthält maximal ein Handle.

Beispiel:

Shift–Reduce–Parsing des Wortes $id_1 + id_2 * id_3$

Grammatik:

$E \to E + E \,|\, E * E \,|\, (E) \,|\, id$

Ableitung:

$$\begin{aligned} E \;&\Rightarrow \underline{E + E} \\ &\Rightarrow E + \underline{E * E} \\ &\Rightarrow E + E * \underline{id_3} \\ &\Rightarrow E + \underline{id_2} * id_3 \\ &\Rightarrow \underline{id_1} + id_2 * id_3 \end{aligned}$$

Die unterstrichenen Stellen kennzeichnen die Handles.

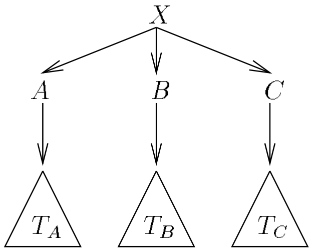
Datenstruktur:

Keller	Eingabe	Operationen
$\$$	$id_1 + id_2 * id_3\$$	shift
$\$id_1$	$+id_2 * id_3\$$	Reduktion durch $E \to id$
$\$E$	$+id_2 * id_3\$$	shift
$\$E+$	$id_2 * id_3\$$	shift
$\$E + id_2$	$*id_3\$$	Reduktion durch $E \to id$
$\$E + E$	$*id_3\$$	shift
$\$E + E*$	$id_3\$$	shift
$\$E + E * id_3$	$\$$	Reduktion durch $E \to id$
$\$E + E * E$	$\$$	Reduktion durch $E \to E * E$
$\$E + E$	$\$$	Reduktion durch $E \to E + E$
$\$E$	$\$$	Accept

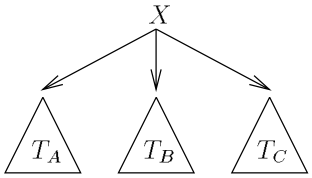
Ableitungsbaum:

- Inhalt des Kellers + nicht gelesener Teil der Eingabe = Rechtssatzform
- Handle kann sich nur im Oberteil des Kellers befinden.
- Jedes Terminalsymbol im Keller entspricht im Baum einem Blatt, wo dieses Zeichen drin ist.

Reduktion:



wird zu



Problem:

Wie kann man Handles finden?

Definition: Operatorengrammatiken

- bei einer Produktion: keine benachbarten Nichtterminale auf der rechten Seite
- es gibt keine ε–Produktionen

Beispiel:

$E \to EAE \mid id$

$A \to + \mid * \mid -$ Keine Operatorengrammatik!

Äquivalent: $E \to E + E \mid E * E \mid E - E \mid id$ diese aber

Relationen $<\!\cdot$, $\doteq$, $\cdot\!>$

- Für zwei Terminalsymbole kann jede von $<\!\cdot$, $\doteq$, $\cdot\!>$ gelten.
- Es kann passieren, daß $a <\!\cdot\, b$, $a \cdot\!> b$ vorkommt.
- Intuition: $<$ $>$ sind die Klammern, die das Handle identifizieren.

Beispiel:

Arithmetische Ausdrücke:

	id	$+$	$*$	$\$$
id		$\cdot\!>$	$\cdot\!>$	$\cdot\!>$
$+$	$<\!\cdot$	$\cdot\!>$	$<\!\cdot$	$\cdot\!>$
$*$	$<\!\cdot$	$\cdot\!>$	$\cdot\!>$	$\cdot\!>$
$\$$	$<\!\cdot$	$<\!\cdot$	$<\!\cdot$	

$\$\, id + id * id\, \$$

Wir setzen richtige Relationen zwischen benachbarte Zeichen.

$\$ <\!\cdot\, id \cdot\!> + <\!\cdot\, id \cdot\!> * <\!\cdot\, id \cdot\!> \$$

Was wir machen:

- Suche nach $\cdot\!>$, die am weitesten links steht.

- Wir gehen weiter nach links, bis wir $<\cdot$ finden.
  (Wir überspringen dabei alle $\doteq$.)
- Handle befindet sich zwischen diesen $<\cdot$, $\cdot>$

Nach Reduktion:

$E + id * id \leadsto$ Wir ignorieren alle Nichtterminalsymbole.
$\$ + id * id \$ \leadsto$ $\$ <\cdot + <\cdot id \cdot> * <\cdot id \cdot> \$$
$\$ <\cdot + <\cdot * <\cdot id \cdot> \$$
$\$ <\cdot + <\cdot \ * \ \cdot> \$$
$\$ <\cdot \ + \ \cdot> \$$
$\$\$$

<u>Auswahl von $<\cdot$, $\doteq$, $\cdot>$</u>

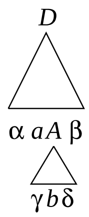
- Falls Θ_1 höhere Priorität als Θ_2 besitzt, dann
  $\Theta_1 \cdot> \Theta_2$ und $\Theta_2 <\cdot \Theta_1$
- Falls Θ_1 und Θ_2 gleiche Priorität besitzen, dann
  $\Theta_1 \cdot> \Theta_2$ und $\Theta_2 \cdot> \Theta_1$ (Θ_1, Θ_2 linksassoziativ)
  $\Theta_1 <\cdot \Theta_2$ und $\Theta_2 <\cdot \Theta_1$ (Θ_1, Θ_2 rechtassoziativ)
- $(\doteq)$, $(<\cdot ($, $(<\cdot id$, ...

<u>Probleme mit Negation:</u>

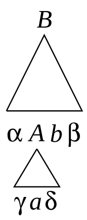
- Infix–Operator $a - b$
- Präfix–Operator $b \cdot -a$
- Man muß zuerst zwischen beiden „$-$“ unterscheiden.

Definition: $<\cdot$, $\doteq$, $\cdot>$

1. $a \doteq b$, falls gilt: $A \to \alpha a \beta b \gamma$,
   wobei $\beta = \varepsilon$ oder β besteht aus einem Nichtterminalsymbol.
2. $a <\cdot b$, falls $D \to \alpha a A \beta$ eine Produktion und $A \overset{+}{\Rightarrow} \gamma b \delta$,
   wobei $\gamma = \varepsilon$ oder γ besteht aus einem Nichtterminalsymbol.



3. $a \cdot> b$, falls $\exists\, B \to \alpha A b \beta$ und $A \overset{+}{\Rightarrow} \gamma a \delta$,
   wobei $\delta = \varepsilon$ oder δ besteht aus einem Nichtterminalsymbol.



Definition: Operator–precedence Grammatik

- G besitzt keine ε–Produktionen ($A \to \varepsilon$).
- Die Relationen $<\!\cdot$, $\doteq$, $\cdot\!>$ sind disjunkt.

Beispiel:

$E \to E + E \mid E * E \mid (E) \mid id$ ist leider keine Operator–precedence Grammatik, denn es gilt:

Eine bessere Grammatik (weil Operator–precedence–Grammatik) für dieselbe Sprache:

$E \to E + T \mid T$

$T \to T * F \mid F$

$F \to (E) \mid id$

Beweis:

Zeichen	erstes Terminalsymbol, das man ableiten kann	letztes Terminalsymbol, das man ableiten kann
E	$+, *, (, id$	$+, *,), id$
T	$*, (, id$	$*,), id$
F	$(, id$	$), id$
	(LEADING)	(TRAILING)

$\doteq$: $(\doteq)$

$<\!\cdot$: Die Rolle von a und A könnten folgende Symbole spielen

$+$ und T	$\overset{\text{es folgt}}{\leadsto}$	$+ <\!\cdot *,\ + <\!\cdot (,\ + <\!\cdot id$
$*$ und F	$\leadsto$	$* <\!\cdot (,\ * <\!\cdot id$
$($ und E	$\leadsto$	$(<\!\cdot +,\ (<\!\cdot *,\ (<\!\cdot id,\ (<\!\cdot ($

$\cdot\!>$: Ähnlich!

Berechnungsmethode von $<\!\cdot$, $\doteq$, $\cdot\!>$

$\text{LEADING}(A) = \{a \mid A \overset{*}{\Rightarrow} \gamma a \delta \text{ mit } \gamma = \varepsilon \text{ oder } \gamma \text{ ein Nichtterminal}\}$

$\text{TRAILING}(A) = \{a \mid A \overset{*}{\Rightarrow} \gamma a \delta \text{ mit } \delta = \varepsilon \text{ oder } \delta \text{ ein Nichtterminal}\}$

Prinzip

i) $a \in \text{LEADING}(A)$, falls $\exists$ Produktion $A \rightarrow \gamma a \delta$,
wobei $\gamma = \varepsilon$ oder γ ein Nichtterminal.

ii) $\exists$ Produktion $A \rightarrow B\alpha$ und $a \in \text{LEADING}(B) \Rightarrow$
$a \in \text{LEADING}(A)$

iii) Wiederhole Schritt ii) bis keine Änderungen mehr möglich sind.

Algorithmus

$\text{L}(A, a) = \text{True} \Leftrightarrow a \in \text{LEADING}(A)$

1. $\text{L}(A, a) = \text{False } \forall a$
2. Änderung nach i), für jedes $\text{L}(A, a) = \text{True}$ schreibt man (A, a) in den Keller.
3. while (Keller nicht leer)

```
Nimm (B, a) aus dem Keller
Für jede Produktion A → Bα tue
    if (L(A, a) = False) then
       L(A, a) = True
       Schreibe (A, a) in den Keller
    endif
endFür
```

endwhile

Berechnung von $\lessdot$, $\doteq$, $\gtrdot$

Für jede Produktion $A \rightarrow x_1 \ldots x_k$

for i = 1 **to** k **begin**

if $(x_i, x_{i+1}$ sind Terminalsymbole) **then** $x_i \doteq x_{i+1}$
if $(x_i, x_{i+2}$ sind Terminalsymbole) **and** $(x_{i+1}$ ist Nichtterminalsymbol) **then** $x_i \doteq x_{i+2}$
if $(x_i$ ist Terminalsymbol) **and** $(x_{i+1}$ ist Nichtterminalsymbol) **then** $\forall a \in \text{LEADING}(x_{i+1})$: $x_i \lessdot a$
if $(x_i$ ist Nichtterminalsymbol) **then** $\forall a \in \text{TRAILING}(x_i)$: $a \gtrdot x_{i+1}$

end

Vereinfachung

Für jedes Terminalsymbol definiere $f(a), g(a) \in \mathbb{Z}$ mit dem Ziel

$$
\begin{array}{lcl}
f(a) < g(b) & \Leftrightarrow & a \lessdot b \\
f(a) = g(b) & \Leftrightarrow & a \doteq b \\
f(a) > g(b) & \Leftrightarrow & a \gtrdot b
\end{array}
$$

Dies ist z.B. bei unserer Grammatik möglich:

	$+$	$*$	(	)	id	\$
f	2	4	0	4	4	0
g	1	3	5	0	5	0

z.B.

$$
\begin{array}{lll}
f(+) > g(+) & \text{, also} & + \gtrdot + \\
f(*) > g(+) & \text{, also} & * \gtrdot + \\
g(*) > f(+) & \text{, also} & + \lessdot *
\end{array}
$$

Prozedur zur Suche von f und g

Für jedes a betrachten wir zwei Knoten f_a, g_a.

Knoten gruppieren: $a \doteq b \Rightarrow f_a$ und g_b sind in derselben Gruppe.

Bemerkung: $a \doteq b \wedge a \doteq c \Rightarrow g_b$ und g_c sind in derselben Gruppe,
aber $b \doteq c$ muß nicht gelten.

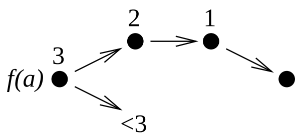
Definiere gerichtete Kanten:

$a <\cdot\, b \Rightarrow$ Es gibt eine Kante von Gruppe g_b zu Gruppe f_a.

$a \cdot\!> b \Rightarrow$ Es gibt eine Kante von Gruppe f_a zu Gruppe g_b.

1. Fall: Der Graph besitzt einen Zyklus, d.h.
   Es gibt keine Funktionen f, g, die die Relationen $<\cdot$, $\doteq$, $\cdot\!>$ definieren. ($\Rightarrow$ keine operator–precedence Grammatik)
2. Fall: Der Graph ist zyklenfrei, d.h.
   $f(a) :=$ Die Länge des längsten Pfades mit Anfangsknoten f_a.
   $g(a) :=$ Die Länge des längsten Pfades mit Anfangsknoten g_a.

Beispiel:



Bemerkung:

Auf jeder gerichteten Kante hat der Anfangsknoten einen größeren Wert als der Endknoten.

3.2 Top–down–Parsing

Beispiel:

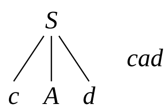
$S \rightarrow cAd$

$A \rightarrow ab \mid a$

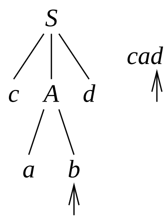
Eingabe: cad

Was macht der Top–down–Parser bei Abarbeitung der Eingabe?

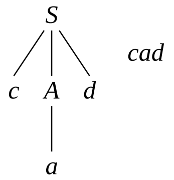
1. Schritt $S \rightarrow cAd$ (eindeutig!)



2. Schritt $A \rightarrow ab$ (nicht korrekt)



3. Schritt Wahl $A \to ab$ muß rückgängig gemacht werden. $A \to a$ (korrekt)



4. Schritt Akzeptiere

Beobachtung:

- Rückschritte wurden getätigt (Backtracking).
- Backtracking ist notwendig. (Analog: Prolog)
- Linksrekursion ist möglich $\Rightarrow$ Die Prozedur gerät u.U. in eine Endlosschleife von Linksrekursionen.

  Wie vermeidet man Linksrekursionen?

  Wandle die gegebene Grammatik zum Beispiel in die bekannte Greibach–Normalform um.
- Wie kann man Backtracking vermeiden ?

  Beispiel: **`if`** ... **`then`** ... **`else`**
  `if` ... **`then`** ...

  Oder $A \to \alpha\beta \mid \alpha\gamma$. Hier ist Backtracking notwendig, da man beim Lesen von α nicht wissen kann, ob danach β oder γ kommen wird. Dies kann man aber ändern:

  Linksfaktorisierung

  Wandle $A \to \alpha\beta \mid \alpha\gamma$ um in
  $A \to \alpha A'$
  $A' \to \beta \mid \gamma$

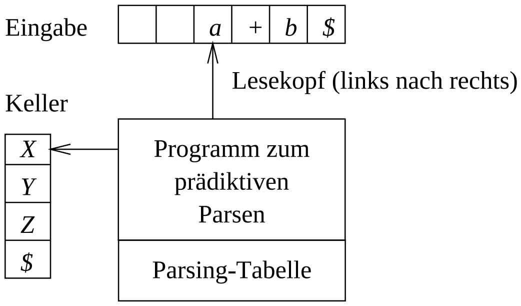
  Hier ist die Backtracking–Tiefe verkleinert worden.

  $\to$ So wenig Backtracking wie möglich machen.

3.3 Prädiktive Parser

Bemerkung:

Kein Backtracking!



Funktionsweise des Prädiktiven Parsers

a sei der Buchstabe, der auf dem Eingabeband gelesen wird,

X das oberste Symbol im Keller.

1.) $X = a = \$$ $\Rightarrow$ akzeptiere
2.) $X = a \neq \$$ $\Rightarrow$ entferne X aus dem Keller. Der Lesekopf bewegt sich um 1 Position nach rechts ($a, X \in \Sigma$) (vgl. Schritt 2) Kellerinhalt ◯ markiert.
3.) X sei Nichtterminalsymbol, suche Eintrag M$[X, a]$ in Parsing–Tabelle. Das ist dann die richtige Position zur Ableitungsbaum–Erweiterung. Es muß nicht mit Backtracking „getestet“ werden.
    1. Fall $M[X, a]$ leer: Fehlermeldung Syntaxfehler
    2. Fall $M[X, a]$ enthält Produktion, z.B. $X \to MNY$, dann ersetze X durch MNY, Lesekopf bewegt sich nicht.

Konstruktion der Parsing–Tabelle

Definition:

$$\begin{aligned} \text{FIRST}(\alpha) &= \{\, x \mid x \text{ Terminalsymbol}, \exists \beta : \alpha \overset{*}{\Rightarrow} x\beta \,\} \\ \text{FOLLOW}(A) &= \{\, x \mid x \text{ Terminalsymbol}, \exists \alpha, \beta : S \overset{*}{\Rightarrow} \alpha A x \beta \,\} \end{aligned}$$

(α beliebige Satzform, A Nichtterminalsymbol)

FIRST–Berechnung

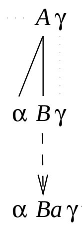
X sei das erste Zeichen von α.

1. X – Terminalsymbol, dann FIRST$(X) = \{X\}$
2. X – Nichtterminalsymbol:
   - $X \to a\alpha$, a Terminalsymbol, dann FIRST(X) = FIRST$(X) \cup$ FIRST(a)
   - $X \to \varepsilon$, FIRST(X) = FIRST$(X) \cup \{\varepsilon\}$
3. X – Nichtterminalsymbol:
   - $X \to Y_1 \ldots Y_p$, $\forall j \leq p$: Falls $Y_1 \ldots Y_p$ Nichtterminalsymbole. $\forall i < j : \varepsilon \in$ FIRST(Y_i), dann FIRST(X) = FIRST$(X) \cup$ FIRST(Y_j)

Diese Schritte werden wiederholt, bis keine Änderung mehr möglich ist.

FOLLOW–Berechnung

1. S Anfangssymbol, $\$ \in$ FOLLOW(S)
2. $A \to \alpha B \beta, \beta \neq \varepsilon$ Produktion:
   FOLLOW(B) = FOLLOW$(B) \cup$ FIRST(β)
3. $A \to \alpha B$ oder
   $A \to \alpha B \beta$ und $\varepsilon \in$ FIRST(β):
   FOLLOW(B) = FOLLOW$(B) \cup$ FOLLOW(A)



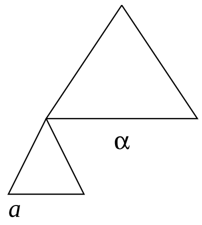
$(a \in \text{FOLLOW}(A),\ a \in \text{FOLLOW}(B))$

Schritte 1. bis 3. werden wiederholt, bis keine Änderung mehr möglich ist.

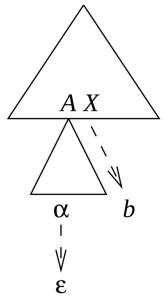
Konstruktion der Parsing–Tabelle

Für jede Produktion $A \to \alpha$ tue folgendes:

- $\forall a \in \Sigma$: Falls $a \in \text{FIRST}(\alpha)$, füge $A \to \alpha$ in $\text{M}(A, a)$ ein.



- Falls $\varepsilon \in \text{FIRST}(\alpha)$, dann füge $A \to \alpha$ in $\text{M}(A, b)$ für jedes $b \in \text{FOLLOW}(A)$ ein.



Problem:

Es kann passieren, daß $\text{M}(A, a)$ mehrere Produktionen enthält. In diesem Fall ist die Parsing–Tabelle unnutzbar, es kann kein Parser generiert werden. Damit das vermieden wird, muß die kontextfreie Grammatik eingeschränkt werden.

Definition:

Eine Grammatik heißt LL(1) $\Leftrightarrow$

$\forall A, a : \text{M}(A, a)$ enthält höchstens eine Produktion.

LL(1): Lesen der Eingabe von links nach rechts,
bilden von Linksableitungen
1 bedeutet, daß in einem Schritt höchstens ein Zeichen vom Eingabeband gelesen wird.

Satz:

G sei eine LL(1)–Grammatik, dann konstruiert der prädiktive Parser für G die Ableitungsbäume für die Worte aus L_G.

Satz:

G ist LL(1)–Grammatik $\Leftrightarrow$

$\forall A : A \to \alpha$ Produktion, $A \to \beta$ Produktion, gilt:

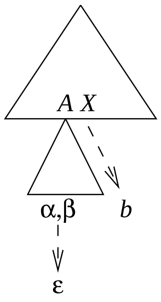
i) Es kann nicht passieren, daß gilt:
   $\alpha \stackrel{*}{\Rightarrow} \underline{a}\alpha'$, $\beta \stackrel{*}{\Rightarrow} \underline{a}\beta'$, $(a \in \Sigma)$
   (Sonst befinden sich $A \to \alpha$, $A \to \beta$ in $\mathrm{M}(A, a)$ und der Parser kann nicht deterministisch entscheiden.)

ii) Falls $\alpha \stackrel{*}{\Rightarrow} \varepsilon$, dann gilt nicht: $\beta \stackrel{*}{\Rightarrow} \varepsilon$ und umgekehrt.
   (Sonst gilt: $\forall b \in \mathrm{FOLLOW}(A)$ erhalten wir 2 Produktionen in $\mathrm{M}(A, b)$.)

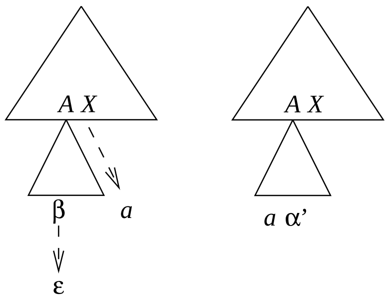
iii) Falls $\beta \stackrel{*}{\Rightarrow} \varepsilon$, $\alpha \stackrel{*}{\Rightarrow} a\alpha'$, dann gilt $a \notin \mathrm{FOLLOW}(A)$
   (Sonst hätten wir $A \to \alpha$ und $A \to \beta$ in $\mathrm{M}(A, a)$.)

Anschaulich:

ii) Hat man das Symbol $b \in \mathrm{FOLLOW}(A)$ in der Vorschau, dann kann nicht deterministisch zwischen $A \to \alpha$ und $A \to \beta$ entschieden werden.



iii) Wäre $a \in \mathrm{FOLLOW}(A)$, dann könnte zwischen $A \to \alpha$ und $A \to \beta$ entschieden werden, denn beide sind Element von $M(A, a)$.



<u>Beweis:</u>

„$\Leftarrow$“ Aus ii), iii) folgt: M kann keine Doppeleinträge enthalten.

Eine weitere Methode des Top–down–Parsing:
<u>Transitionsdiagramme</u>

$$\forall A \in V : A \to X_1 \ldots X_n$$

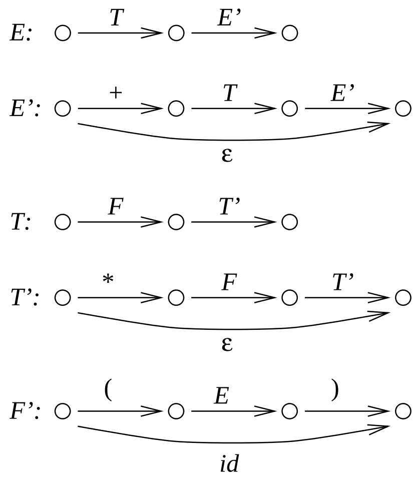
$$\underset{Quelle}{\circ} \xrightarrow{X_1} \circ \xrightarrow{X_2} \circ \cdots \circ \xrightarrow{X_n} \underset{Senke}{\circ}$$

<u>Beispiel:</u>

Grammatik:

$$\begin{aligned} E &\to TE' \\ E' &\to +TE' \mid \varepsilon \\ T &\to FT' \\ T' &\to *FT' \mid \varepsilon \\ F &\to (E) \mid id \end{aligned}$$

zugehörige Transitionsdiagramme:

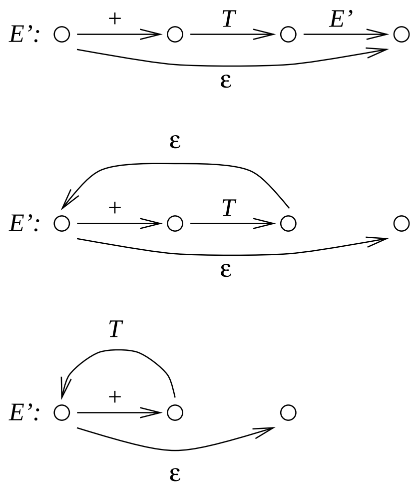


<u>Bemerkung:</u>

Top–down Parsing ist eine Wanderung durch Diagramme, daher sollten diese optimiert werden.

<u>Beispiel:</u>

E' aus voherigem Beispiel:



(Eliminierung der Rekursion)

3.4 LR(0)–Grammatiken

(LR(0) — von links nach rechts, Rechtsableitungen bildend,
keine Zeichen aus Eingabeband vor der Entscheidung geholt)

Ziel: bottom–up–parsing

Problem: Handle–Erkennung

Annahme: Wir betrachten nur kfG mit Präfix–Eigenschaft:
$w \in L_G \Leftrightarrow$ kein Präfix von w ist in L_G
Dies ist keine wesentliche Einschränkung:
L kfS $\Rightarrow L\$ = \{w\$ \,|\, w \in L\}$ ist kfS

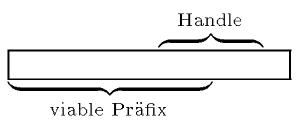
<u>Technische Definitionen zur Handle Suche</u>

Element (item): Situation

$A \to \alpha\beta$ Produktion $\Rightarrow A \to \alpha \cdot \beta$ ist Element (α, β können leer sein)

Viable Präfix (lebensfähiges Präfix)

Satzform:



Das Ende des viable Präfix darf nicht auf der rechten Seite des Handles liegen.

Definition:

Eine Situation (Element) $A \to \alpha \cdot \beta$ ist *gültig* für ein viable Präfix γ, falls es eine Rechtsableitung gibt vom Typ

$$S \overset{*}{\Rightarrow} \delta A w \Rightarrow \delta\alpha\beta w \ (\gamma = \delta\alpha \text{ und Handle} = \alpha\beta)$$

Ziel:

Für jedes viable Präfix gültige Elemente berechnen!

Berechnung der gültigen Elemente:

Man nehme einen NEA mit Zuständen: Elemente $\cup \{q_0\}$, q_0 Anfangszustand

$\delta(q_0, \varepsilon) = \{S \to \cdot\alpha \mid S \to \alpha \;\text{ Produktion}\}$

$\delta(A \to \alpha \cdot X\beta, X) = \{A \to \alpha X \cdot \beta\}$

$$\left.\begin{array}{l} A \to \alpha \cdot X\beta \text{ nach dem Lesen von } \gamma \\ S \stackrel{*}{\Rightarrow} \delta A w \Rightarrow \underbrace{\delta\alpha}_{\gamma} X\beta w \end{array}\right\} \text{Falls NEA richtig funktioniert!}$$

Aber dann:

$S \stackrel{*}{\Rightarrow} \delta A w \Rightarrow \underbrace{\delta\alpha}_{\gamma} X\beta w \quad$ für γX ist Element $A \to \alpha X \cdot \beta$ gültig

$\delta(A \to \alpha \cdot B\beta, \varepsilon) = \{B \to \cdot\eta \mid B \to \eta \text{ Produktion}\}$

Begründung:

$S \stackrel{*}{\Rightarrow} \delta A w \Rightarrow \underbrace{\delta\alpha}_{\gamma} B\beta w$, also auch:

$S \stackrel{*}{\Rightarrow} \delta A w \Rightarrow \delta\alpha B\beta w \stackrel{*}{\Rightarrow} \delta\alpha B\beta' w \Rightarrow \delta\alpha\eta\beta' w$

$B \Rightarrow \cdot\eta$ ist gültig für γ.

Dieser NEA rät nur gültige Elemente.

Problem: Deterministischer Automat gesucht

1. Lösung: Potenzmengenkonstruktion

   Aber:
   Hat der NEA z.B. auch nur 10 Zustände, so hat unter Umständen der resultierende DEA unvertretbar viele Zustände (1024).

2. Lösung

   Ermittle für ein γ:
   Nach dem Lesen von γ gerät der DEA in folgenden Zustand:
   Menge aller Zustände, die für γ gültig sind.

Beispiel:

Beispielgrammatik:

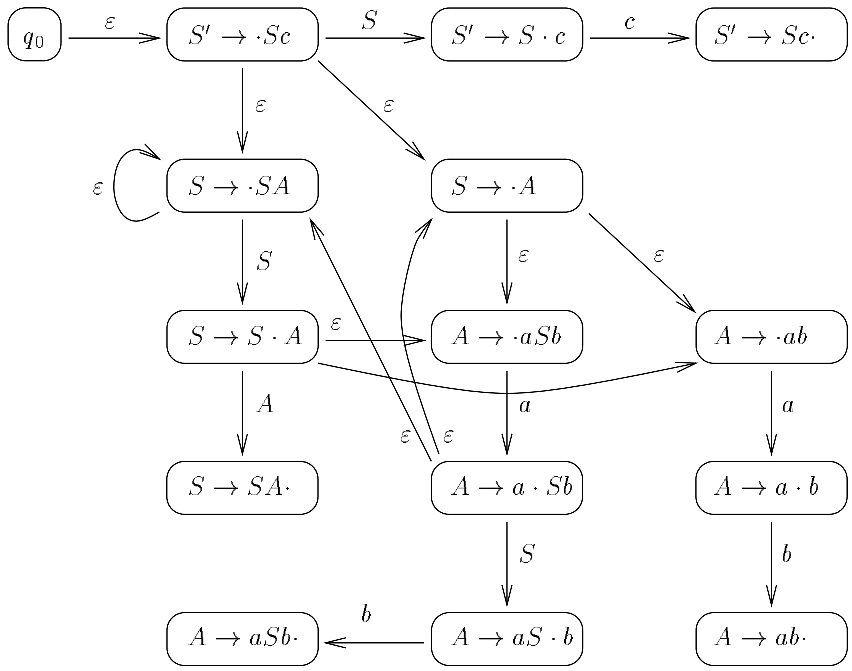
$S' \to Sc$

$S \to SA \mid A$

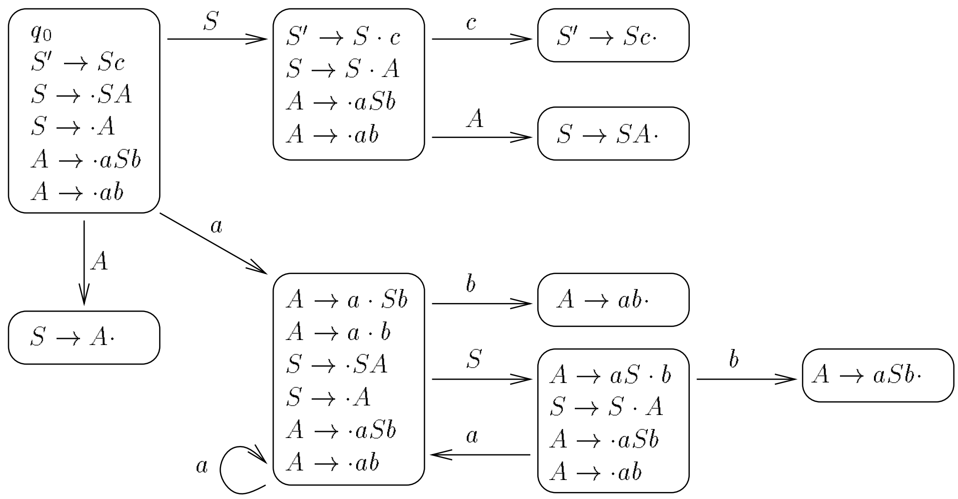
$A \to aSb \mid ab$

$\delta(A \to \alpha \cdot X\beta, X) = (A \to \alpha X \cdot \beta)$

$\delta(A \to \alpha \cdot B\beta, \varepsilon) = \{B \to \cdot\eta \mid (B \to \eta) \in P\}$



Idee: Fasse alle ε–Bewegungen zusammen, weil alle Elemente des daraus resultierenden Zustands für das entsprechende Präfix γ gültig sind (Analogie: Hüllenbildung).



Satz:

Der oben definierte NEA hat folgende Eigenschaft:

$\delta(q_0, \gamma)$ enthält $A \to \alpha \cdot \beta \Leftrightarrow A \to \alpha \cdot \beta$ ist gültig für γ.

$\uparrow$ Zustand des NEA nach Lesen von γ. q_0 ist Anfangszustand.

<u>Beweis:</u>

($\Rightarrow$) ist offensichtlich.

($\Leftarrow$)

Wir zeigen, daß jedes gültige Element erreichbar ist.

Sei $A \to \alpha \cdot \beta$ gültig für γ, also $S \overset{*}{\Rightarrow} \gamma_1 A w \Rightarrow \underbrace{\gamma_1 \alpha}_{\gamma} \beta w$

Wir zeigen, daß $\delta(q_0, \gamma_1)$ die Produktion $A \to \cdot\alpha\beta$ enthält.

Dann ist ($\Leftarrow$) gezeigt.

Induktion über die Länge der Ableitung.

Sei $S \overset{*}{\Rightarrow} \gamma_1 A w$, $S \overset{*}{\Rightarrow} \gamma_2 B x \Rightarrow \gamma_2\gamma_3 A \gamma_4 x \overset{*}{\Rightarrow} \underbrace{\gamma_2\gamma_3}_{\gamma_1} A \underbrace{yx}_{w}$.

Nach der Induktionsvoraussetzung gilt für $S \overset{*}{\Rightarrow} \gamma_2 B x \Rightarrow \gamma_2\gamma_3 A \gamma_4 x$ $[B \to \cdot\gamma_3 A \gamma_4]$ ist gültig für $\gamma_2 : \delta(q_0, \gamma_2)$ enthält Produktion.

Dann gilt: $\delta(q_0, \gamma_2\gamma_3)$ enthält $B \to \gamma_3 \cdot A\gamma_4$.

Induktionsschritt (der Länge 1): $\delta(q_0, \gamma_2\gamma_3) = \delta(q_0, \gamma_1)$ enthält $A \to \cdot\alpha\beta$, oder anders: $A \to \cdot\alpha\beta$ ist gültig für γ_1.

Definition:

KfG heißt LR(0), falls gilt:

1) Das Anfangssymbol kommt auf der rechten Seite einer Produktion nicht vor.
2) Für jedes viable Präfix γ gilt:
   Falls $A \to \alpha\cdot$ für γ gültig ist, dann ist kein anderes Element für γ gültig. (sonst reduziere–reduziere–Konflikt)
   Ausnahme: Falls $\alpha = \varepsilon$ und alle Terminalsymbole sind auf der rechten Seite von einem Punkt.

Bemerkung:

Entscheidung, ob eine Grammatik die LR(0)–Eigenschaft besitzt:

Definiere DEA, der gültige Elemente berechnet.

Teste, ob ein Zustand diesen Eigenschaften widerspricht.

3.5 LR(0)–Parser

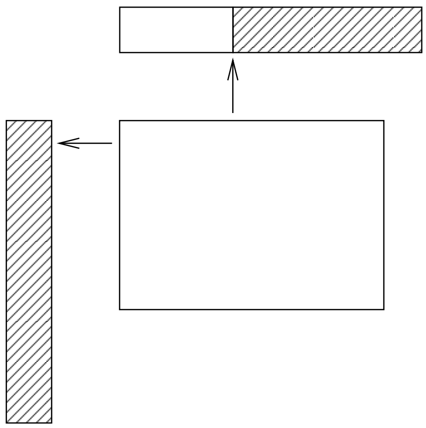
Satz:

Sei G eine LR(0)–Grammatik. Dann kann L_G durch einen deterministischen Kellerautomaten akzeptiert werden. Bei leerem Keller akzeptiert der Kellerautomat. Dieser Kellerautomat heißt LR(0)–Parser.

Definition: LR(0)–Parser

Kellerinhalt: $X_1 s_1 X_2 s_2 \ldots X_k s_k$, wobei

- $X_1 \ldots X_k$ bildet ein viable Präfix,
- $s_i \quad \forall i \leq k$ ist der Zustand des Automaten, der gültige Elemente für $X_1 \ldots X_i$ berechnet.



Inhalt des Kellers + nichtgelesener Teil der Eingabe = Rechtssatzform

Berechnung: 1. Schritt (reduce)

Fall 1: s_k enthält vollständiges Element $(A \to X_{k-i} \dots X_k)$
Im Keller gibt es den Handle $X_{k-i} \dots X_k$

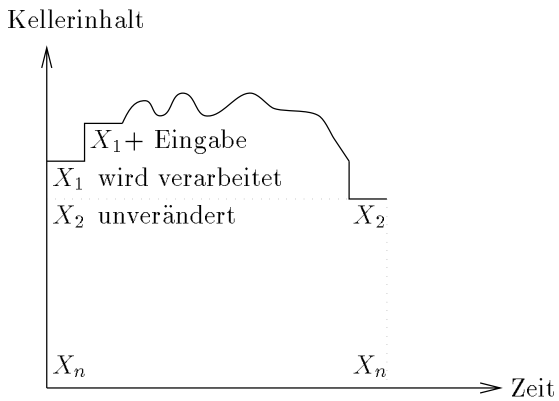
- Zuerst $X_{k-i}s_{k-i} \dots X_k s_k$ aus dem Keller entfernen
- Dann kommt A in den Keller
- Dann kommt neues gültiges Element hinein:
  $\delta(s_{k-i-1}, A)$ für $X_1 \dots X_{k-i-1}A$.

Fall 2: s_k enthält kein vollständiges Element. Dann ist im Keller kein Handle. „Fülle“ weiter auf:

- Neues Eingabesymbol wird gelesen und in den Keller geschrieben. (shift)
- $s_{k+1} = \delta(s_k, X_{k+1})$ wird berechnet und in den Keller geschrieben.

Jede deterministisch kfS L besitzt einen äquivalenten deterministischen Kellerautomaten M. Zu M konstruieren wir eine LR(0)–Grammatik G_M.

Zur Erinnerung



$(q, X_1) \rightsquigarrow (p, X_1 X_2 \dots X_n)$

Schreibweise:

$[qX_1p][pX_2p'] \ldots$

Nichtterminale Zeichen:

Q – Menge der Zustände des Kellerautomaten M.

δ – Übergangsfunktion

- $[qXp]$ $\quad p, q \in Q$, X ist Kellersymbol
- A_{qaY} $\quad q \in Q$, a Eingabesymbol, Y Kellersymbol

Neuer Beweis:

A_{qaY}, um zu gewährleisten, daß G_M eine LR(0)–Grammatik ist (Steuerung).

Produktionen:

(1) $S \to [q_0 Z_0 p]$ für jedes $p \in Q$

(2) $[qYp] \to A_{qaY}$,
falls $\delta(q, a, Y) = (p, \varepsilon)$

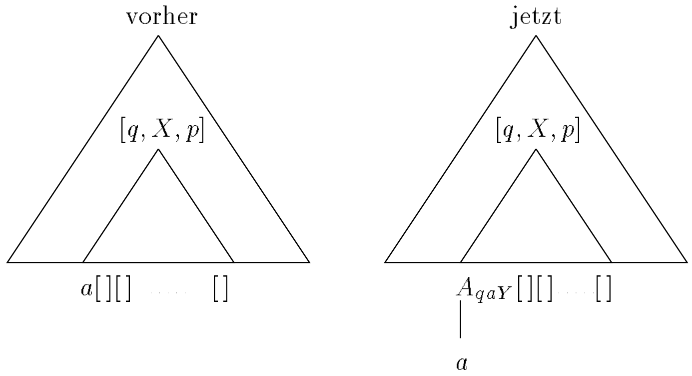
(3) $[qYq_{k+1}] \to A_{qaY}[q_1X_1q_2][q_2X_2q_3] \cdots [q_kX_kq_{k+1}]$,
falls $\delta(q, a, Y) = (q_1, X_1 \ldots X_k)$ $\forall q_1, \ldots, q_k \in Q$

(4) $A_{qaY} \to a \quad \forall q, a, Y$

Bemerkung:

Diese Grammatik erzeugt dieselbe Sprache wie die ursprüngliche Grammatik aus dem Beweis/Satz „Jeder Kellerautomat akzeptiert eine kontextfreie Sprache.“.

Skizze:



Bemerkung:

Wichtig!

Eine Folge der Schritte von M entspricht $\leftrightsquigarrow$

(der Folge der Ableitungen mit Hilfe der Produktionen $A_{qaY} \to a$ für eine Rechtsableitung)^R (Reverse, Spiegelung)

Grund: $[qXp] \to \underbrace{A_{qaY}}_{\text{später}}\ \underbrace{[qX_1p_1] \cdots [p_kX_{k+1}p]}_{\text{wird früher durch Terminalsymbole ersetzt}}$

Definition:

- Wir nehmen an, daß die Grammatik keine nutzlosen Symbole besitzt.
- Für jedes $[qXp]$ sei w_{qXp} eine Zeichenkette aus Terminalsymbolen, für die gilt: $[qXp] \stackrel{*}{\Rightarrow} w_{qXp}$
  $h(A_{qaY}) = a$
  $h([qXp]) = w_{qXp}$
- $h(XY) = h(X)h(Y)$
- N sei die Anzahl der Schritte des Kellerautomaten, die $\gamma \stackrel{*}{\Rightarrow} h(\gamma)$ entsprechen. Also
  $N(A_{qaY}) = 1$
  $N([qXp]) =$ Anzahl der Schritte für w_{qXp}
- $m(A_{qaY}) = (qaY)$
  $m([qXp]) =$ (Folge der Indizes von A für die Ableitung von $w_{qXp})^R$
  $m(XY) = m(X)m(Y)$

Lemma:

Sei γ ein viable Präfix. Dann existieren p und β mit
$(q_0, h(\gamma), Z_0) \stackrel{N(\gamma)}{\vdash} (p, \varepsilon, \beta)$ durch die Schritte in $m(\gamma)$.

<u>Beweis:</u>

$[q_0 Z_0 r] \stackrel{*}{\Rightarrow} \gamma y \stackrel{*}{\Rightarrow} \gamma y' \stackrel{*}{\Rightarrow} h(\gamma) y'$
(D.h. es gibt eine eindeutige Folge von Bewegungen, so daß
$(q_0 h(\gamma) y', Z_0) \stackrel{*}{\vdash} (p, \varepsilon, \varepsilon)$, $y' \in T^*)$
Die letzten Ableitungsschritte ($\stackrel{*}{\Rightarrow}$) entsprechen $N(\gamma)$ Schritten des Kellerautomaten. Diese Schritte sind die Schritte $m(\gamma)$.
$(q_0, h(\gamma) y', Z_0) \stackrel{N(\gamma)}{\vdash} (p, y', \beta)$
Ziel: Für jedes γ (viable Präfix), für die das vollständige Element $(\ldots \rightarrow \beta)$ gültig ist, gibt es kein anderes gültiges Element.

Lemma:

Sei I die Menge der gültigen Elemente für γ (viable Präfix).
$B \rightarrow \beta\cdot$ ist in I, dann gehört kein $A_{qaY} \rightarrow \cdot a$ zu I.

<u>Beweis:</u>

Fall 1: $B \rightarrow \beta \stackrel{\text{ist vom Typ}}{\equiv} S \rightarrow [q_0 Z_0 p]$, $\gamma = [q_0 Z_0 p]$
(Annahme: $(A_{qaY} \rightarrow \cdot a \in I) \Rightarrow S \stackrel{*}{\Rightarrow} \gamma A_{qaY}\, y$, aber γ existiert nur nach dem ersten Schritt!? $\natural$ zur Annahme!)

Fall 2: i) $B \rightarrow \beta \equiv [qYp] \rightarrow A_{qaY}[\,][\,]\cdots[\,]$, $\gamma = \gamma'\beta$
$S \stackrel{*}{\Rightarrow} \gamma' B y \Rightarrow \gamma' \beta y \underbrace{\Rightarrow}_{\text{das letzte Zeichen aus } \beta \text{ wird ersetzt.}} \gamma' \beta' y,$
β muß unmittelbar nach dem Erzeugen ersetzt werden.
$S \stackrel{*}{\Rightarrow} \gamma' \beta A_{qaY} y' \Rightarrow \gamma' \beta a y'$, dann wäre $A_{qaY} \rightarrow \cdot a$ in I $\natural$

ii) Anwendung einer anderen Produktion. – Das ist unmöglich!
Zuerst nehmen wir an: $\ldots \Rightarrow x\alpha_1 M\alpha_2 y \Rightarrow x\underbrace{\alpha_1\delta\alpha_2 y}_{\beta}$, $\alpha_1 \neq \varepsilon$

a) $M = A_{qaY}$, dann ist $\delta = a$ und β enthält Terminale $\natural$
b) $M = [qXp]$, dann $\delta = A_{qXp}[\,]\ldots$ (oder leer)
Falls $\alpha_1 \neq \varepsilon$, dann steht A_{qXp} in der Mitte von β $\natural$
Für $\alpha_1\delta_1\delta_2$ ($\beta = \alpha_1\delta_1$, $\delta = \delta_1\delta_2$) erhalten wir einen ähnlichen Widerspruch.

Weiter nehmen wir an: $\ldots \Rightarrow xM\alpha_2 Y \Rightarrow x\delta_1\underbrace{\delta_2\alpha_2 y}_{\beta}$

$\delta_2 \neq \varepsilon$, denn sonst hätten wir β schon früher als Teilwort.
(Das ist also der erste Schritt, in dem β als Teilwort vorkommt.)

a) $\delta_2 = a$ und $M = A_{qaY}$ $\natural$ (da β ohne Terminale)
b) $M = [\ldots]\ M \to \delta\, A \ldots [\,][\,]\ldots$
$\Rightarrow \delta_1 = \varepsilon$, sonst wäre ein Zeichen $[\ldots]$ das erste Zeichen von δ_2, also auch von β, β ist aber normale Produktion.
δ ist ein Präfix von β, also $\delta = \beta$ (die Zeichen von δ und β beschreiben vollständig die Länge des Restes, die Anzahl der $[\ldots]$ Ausdrücke).

Fall 3: $B \to \beta \equiv A_{pbZ} \to b$
Jetzt: γA_{qaY} – viable Präfix, $A_{qaY} \to \cdot a$
$\gamma = \gamma' b$, weil $A_{pbZ} \to b$ gültig für γ (ist Suffix)
$\gamma' b A_{qaY}$ – viable Präfix, aber es kann kein Terminalzeichen auf der linken Seite von A_{qaY} stehen ($\natural$, falls $b \neq \varepsilon$)
$b = \varepsilon$: Wir betrachten viable Präfixe γA_{qaY}, $\gamma A_{p\varepsilon Z}$.
Was passiert im Kellerautomaten?
$h(\gamma A_{qaY}) = h(\gamma)a$
$h(\gamma A_{p\varepsilon Z}) = h(\gamma)$
Wir betrachten $N(\gamma) + 1$ Schritte von M auf $h(\gamma)a$.
Die ersten $N(\gamma) + 1$ Schritte von M auf $h(\gamma)a = h(\gamma A_{p\varepsilon Z})a$ sind $m(\gamma)m(A_{p\varepsilon Z}) = m(\gamma)(p, \varepsilon, Z)$.
Andersherum:
$h(\gamma)a = h(\gamma A_{qaY})$ diese $N(\gamma) + 1$ Schritte sind in
$m(\gamma A_{qaY}) = m(\gamma)m(A_{qaY}) = m(\gamma)(q, a, Y)$
Widerspruch zur Annahme, daß M deterministisch ist!

Lemma: reduce–reduce Konflikt

Sei I die Menge der gültigen Elemente für γ, $B \to \beta\cdot \in I$, dann ist kein anderes Element $C \to \alpha\cdot \in I$.

Beweis:

Fall 1: $B \to \beta\cdot$, $C \to \alpha\cdot$ sind keine „$A_{qaY} \to a$“–Elemente.
(Weitere Fälle, z.B. $\beta = A \ldots [\,] \ldots [\,]$

$$\boxed{\quad\gamma\quad}\underbrace{\boxed{A}\overbrace{\boxed{[\,]}\boxed{\cdots}\boxed{[\,]}}^{\alpha}}_{\beta}$$ $\natural$, da α mit $[\,]$ anfängt.

$$\boxed{\gamma} \overbrace{\boxed{A}\underbrace{\boxed{[\,]}\boxed{\cdots}\boxed{[\,]}}_{\beta}}^{\alpha}$$ ♮, da β mit [] anfängt.

$$\boxed{\gamma} \overbrace{\underbrace{\boxed{A_{qaY}}\boxed{\quad}\boxed{[\cdots p]}}_{\beta}}^{\alpha}$$ Wie sehen B, C aus?

$B = [qYp]$, $C = [qYp]$ gleiche Produktion!

)

(Ähnlich für $S \to [q_0 Z_0 p]$)

Fall 2: $B \to \beta\cdot$, $C \to \alpha\cdot$ sind vom Typ „$A_{qaY} \to a\cdot$“. Dann $\alpha = \beta$ oder $\alpha = \varepsilon$ oder $\beta = \varepsilon$

a) $\alpha = \beta$: Betrachte die ersten $N(\gamma) + 1$ Schritte auf $h(\gamma)a$
$\gamma A_{q\alpha Y}$, $h(\gamma A_{q\alpha Y}) = h(\gamma)\alpha$
$\gamma A_{q'\beta' Y'}$, $h(\gamma A_{q'\beta Y}) = h(\gamma)\beta = h(\gamma)\alpha$

1. Auf $h(\gamma A_{q\alpha Y})$ sind die ersten $N(\gamma) + 1$ Schritte:
   $m(\gamma)m(A_{q\alpha Y}) = m(\gamma)(q\alpha Y)$
2. Auf $h(\gamma A_{q'\beta Y'})$ sind die ersten $N(\gamma) + 1$ Schritte:
   $m(\gamma)m(A_{q'\beta Y'}) = m(\gamma)(q'\beta\, Y')$

Der Kellerautomat muß aber deterministisch sein!

b) $\alpha = \varepsilon$: Betrachte die ersten $N(\gamma) + 1$ Schritte auf $h(\gamma)\beta$.
Wir betrachten:
$h(\gamma A_{q\varepsilon Y}) = h(\gamma)$
$h(\gamma A_{q'\beta Y'}) = m(\gamma)\beta$
$N(\gamma) + 1$ Schritte auf $h(\gamma)\beta$. Nach Lemma gilt:
Für $\gamma A_{q\varepsilon Y} : m(\gamma)(q\varepsilon Y)$
Für $\gamma A_{q'\beta Y'} : m(\gamma)(q'\beta Y')$
Wir haben also wieder zwei Berechnungen für dieselbe Eingabe, aber der Kellerautomat soll deterministisch sein!

c) $\alpha = \beta = \varepsilon$ Ähnlich.

Fall 3: $B \to \beta\cdot$ ist vom Typ $A_{qaY} \to a\cdot$.
$C \to \alpha\cdot$ ist nicht vom Typ $A_{qaY} \to a\cdot$.

$$\boxed{\gamma \qquad \underbrace{\qquad\quad \boxed{\overbrace{a}^{\beta}}}_{\alpha}}$$

Aber α hat nur Nichtterminalzeichen auf der rechten Seite $\Rightarrow$ ♮

Korollar:

Die konstruierte Grammatik ist LR(0).

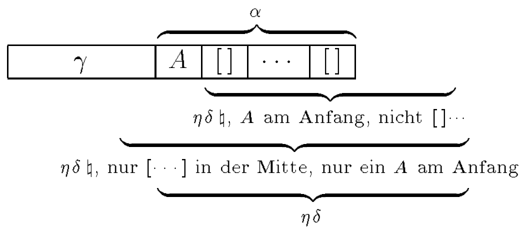
Beweis:

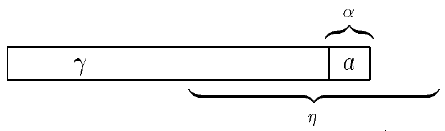
Sei $B \to \alpha\cdot$ gültig für γ.

- Kein $C \to \beta \cdots$ ist gültig für γ ($C \to \beta\cdot \not\equiv B \to \alpha\cdot$)
- Kein $A_{qaY} \to \cdot a$ ist gültig für γ. (Lemma 1)

Einzige weitere Möglichkeiten:

$C \to \eta \cdot \delta$ gültig für γ. Dann ist
$\eta = A \ldots [\,] \ldots [\,]$



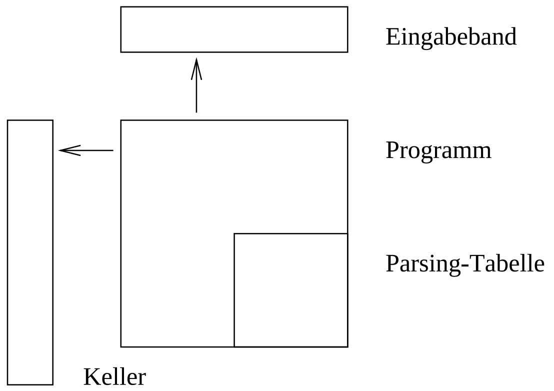


, aber η enthält keine Terminale!

$C \to \beta \cdot \equiv S \to \cdot[q_0 Z_0 p]$, $S \Rightarrow \ldots \Rightarrow \gamma[q_0 Z_0 p]$
ist aber nicht möglich, denn $[q_0 Z_0 p]$ verschwindet nach dem zweiten Schritt.
$S \Rightarrow [q_0 Z_0 p] \Rightarrow \underbrace{\ldots}_{\text{kein } [q_0 Z_0 p] \text{ enthalten}}$

3.6 LR–Parser

Definition: LR–Parser



- Im Keller: $s_0 X_1 s_1 X_2 s \cdots X_m s_m$
  (s_0 Zustand, Kellerinhalt: viable Präfix)
- Konfiguration: $(\underbrace{s_0 X_1 s_1 X_2 s \cdots X_m s_m}_{\text{Keller}},\ \underbrace{a_i a_{i+1} \cdots a_n \$}_{\text{nicht gelesener Teil vom Eingabeband}}\ )$
- Operationen
  AKTION(s_m, a) (Look–up in Parsing–Tabelle) mit s_m Kellersymbol, a Eingabesymbol

  1. shift s :
     Ein Zeichen vom Eingabeband holen, Zeichen s zusätzlich eintragen.
  2. Reduziere durch $A \to \beta$
  3. Akzeptiere

4. Fehler

Neue Konfiguration:

zu 1.) $(s_0 X_1 s_1 X_2 s_2 \cdots X_m s_m a_i s, a_{i+1} \cdots a_n \$)$
$s = \text{GOTO}(s_m, a_i)$, ein Schritt des Automaten, Übergang

zu 2.) $(s_0 X_1 s_1 X_2 s_2 \cdots X_{m-r} s_{m-r} A s, a_i \cdots a_n \$)$
$s = \text{GOTO}(s_{m-r}, A)$, $r =$ Länge von β

3.7 Simple–LR–Parsing–Tabelle (SLR)

Vorteil:

Wie LR(0), aber wir nutzen die gesamte vorhandene Information aus:

- Falls $A \to \alpha\cdot$ zu s_M gehört, dann
ACTION(s_M,a)= „reduziere durch $A \to \alpha$, falls a $\in$ FOLLOW(A)"
(FOLLOW läßt sich im voraus berechnen.)
- Falls $A \to \alpha \cdot a\beta$ in s_M ist, dann ACTION(s_M,a)= shift

Es geht nicht um frühere Fehlererkennung!

Definition:

Die SLR–Parsing–Tabelle enthält keine Konflikte.

(ACTION(s_M,a) ist eindeutig definiert, aber ACTION ist abhängig vom gelesenen Zeichen a.)

Beispiel:

$E' \to E,\ E \to E + T \,|\, T,\ T \to T * F \,|\, F,\ F \to (E) \,|\, id$

Man kann zeigen, daß diese Grammatik keine LR(0)–Grammatik ist.

Betrachte folgenden Zustand: $I_1 = \{E' \to E\cdot,\ E \to E \cdot +T\}$

Wie sieht ACTION hier aus?

ACTION$(I_1, \$)$ = „akzeptiere"

ACTION$(I_1, +)$ = Shift

Achtung: In FOLLOW(E') darf „+" nicht vorkommen, da sonst Konflikte entstehen.

SLR reicht nicht immer.

Bsp:

$S \to L = R \,|\, R$	$- * R$ kann man als Inhalt der Zelle R verstehen.
$L \to *R \,|\, id$	– Grammatik ist eindeutig.
$R \to L$	

Zustand $I_2 = \{S \to L\cdot = R,\ R \to L\cdot\}$

- „=" $\in$ FOLLOW(R) (weil $S \Rightarrow L = R \Rightarrow *R = R$)
- ACTION$(I_2, =)$ = Shift
ACTION$(I_2, =)$ = reduziere, weil $= \in$ FOLLOW(R)

Konflikt! Idee: Ein Look–ahead–Zeichen in Betracht nehmen.

3.8 Kanonische LR–Parsing–Tabelle

Definition: LR(1)–Element

[LR(0)–Element, Eingabesymbol] ist LR(1)–Element.

Ein Element [A $\rightarrow \alpha \cdot \beta, a$] heißt *gültig* für ein viable Präfix γ, falls es eine Rechtsableitung gibt mit:

$S \overset{*}{\Rightarrow} \delta A w \Rightarrow \delta\alpha\beta w$ ($\gamma = \delta\alpha$), wobei a das erste Terminal aus w ist (oder $w = \varepsilon$, dann a = \$).

Beispiel:

$S \rightarrow BB,\ B \rightarrow aB \,|\, b$, also $L = a^*ba^*b$

$S \overset{*}{\Rightarrow} aaBab \Rightarrow aaaBab$, $[B \rightarrow a \cdot B, a]$ ist gültig für γ.

$[B \rightarrow a \cdot Ba, \$]$ ist gültig für Baa.

$S \Rightarrow BB \Rightarrow BaB \Rightarrow \underbrace{Baa}_{\gamma}B$

Ist $[B \rightarrow a \cdot B, a]$ gültig für Baa? Angenommen ja, dann

$S \overset{*}{\Rightarrow} BaBa \Rightarrow BaaBa \Rightarrow a \ldots aba \ldots aba \ldots \notin L\natural$. Also ungültig.

Wie berechnet man nun gültige LR(1)–Elemente?

Sei $[A \rightarrow \alpha \cdot B\beta, a]$ gültiges LR(1)–Element für γ, das ist

$S \overset{*}{\Rightarrow} \delta Aax \Rightarrow \delta\alpha B\beta ax \overset{B\rightarrow\eta}{\Rightarrow} \delta\alpha\eta\beta ax$ – $B \rightarrow \cdot\eta$

Ganz allgemein:

$S \overset{*}{\Rightarrow} \delta Aax \Rightarrow \delta\alpha B\beta ax \overset{*}{\Rightarrow} \delta\alpha B\beta' ax \Rightarrow \underbrace{\delta\alpha}_{\gamma}\eta\beta' ax$, dann $[B \rightarrow \cdot\eta, y]$,

wobei y das erste Zeichen aus $\beta' ax$ ist. (γ enthält nur Terminale.)

Was gilt für γ?

Fall 1: $\beta' = \varepsilon$, dann $\gamma = a$

Fall 2: $\beta' \neq \varepsilon$, dann

$y \in \text{FIRST}(\beta' ax) = \text{FIRST}(\beta' a) \in \underbrace{\text{FIRST}(\beta a)}_{\text{schon bekannt}}$

$[A \rightarrow \alpha \cdot X\beta, a]$ ist gültig für γ. Dann ist

$S \overset{*}{\Rightarrow} \delta Aax \Rightarrow \underbrace{\delta a X}_{\gamma'}\beta ax$. Also ist $[A \rightarrow \alpha \cdot X\beta, a]$ gültig für γ'.

Prozedur zur Berechnung der LR(1)–Elemente

PROCEDURE CLOSURE(I);

REPEAT

$\forall$ Elemente $[A \rightarrow \alpha \cdot B\beta, a] \in I$, Produktionen $B \rightarrow \gamma$,

$b \in \text{FIRST}(\beta a)$ und $[B \rightarrow \cdot\gamma, b] \notin I : I = I \cup [B \rightarrow \cdot\gamma, b]$

UNTIL Keine Änderung mehr möglich.

Gebe I aus.

CLOSURE ist vergleichbar mit der Hüllenbildung aus Compilerbau.

PROCEDURE GOTO(I, X) (,X ist Zeichen aus dem Vokabular.)

$I = \{[A \rightarrow \alpha X \cdot \beta, a] | [A \rightarrow \alpha \cdot X\beta, a] \in I\}$

Gebe I aus.

Hauptteil:

$C :=$ CLOSURE($\{[S' \to \cdot S] | S' \to S$ in Grammatik$\}$)

für jedes $I \in C$, X Zeichen mit GOTO$(I, X) \neq \varnothing$

$C := C \cup \{$ CLOSURE(GOTO(I, X))$\}$

UNTIL Keine Änderung mehr in C.

Danach steht der Kellerautomat LR(1).

Kanonische Parsing–Tabelle (LR(1))

Im Keller steht I (der Zustand = Menge von Elementen), das Eingabesymbol ist $a \in \Sigma$.

- Falls $[A \to \alpha \cdot a\beta, b] \in I$, führe „Shift" aus, $a \in \Sigma$:
  $a \to$ Keller, neuer Zustand im Keller: CLOSURE(GOTO(I, a)).
- Falls $[A \to \alpha\cdot, a] \in I$, dann ACTION$(I, a)$ = reduziere $A \to \alpha$
- Falls $[S' \to S\cdot, \$] \in I$, dann ACTION$(I, S)$ = akzeptiere.

Was ist nun eine LR(1)–Grammatik?

- Grammatik, in der keine Konflikte bei ACTION() in der Parsing–Tabelle auftreten.
  Folgende Konflikte sind möglich:
  - $\left.\begin{array}{l}[A \to \alpha\cdot, b] \\ [A \to \beta\cdot, b]\end{array}\right\}$ reduce–reduce Konflikt
  - $\left.\begin{array}{l}[A \to \alpha \cdot a\beta, c] \\ [B \to \beta\cdot, a]\end{array}\right\}$ shift–reduce Konflikt

Bemerkung:

- Jede SLR–Grammatik ist LR(1).
- Nicht jede LR(1)–Grammatik ist SLR. (SLR $\subsetneq$ LR(1))
- Für praktische Zwecke sind LR(1) Parser stark genug.
  nicht alle Grammatiken von dkfS sind LR(1).
- Die LR(1)–Parsing–Tabelle wird manchmal sehr groß.
  $\Rightarrow$ Motivation für neuen Grammatiktyp der zwischen SLR und LR(1) liegt.

$[\cdot\cdot, \$]$			Fehler shift		

$\left.\begin{array}{l}[S \to \alpha \cdot \beta, a] \\ [B \to \alpha \cdot \beta c, \$]\end{array}\right\}$ LR(1)

3.9 Look–Ahead–LR–Parsing (LALR)

Vorteile:

- kleine Parsing–Tabelle
- erfolgreiche Methode
- Größe der Parsing–Tabelle für SLR und LALR sind gleich.

Beispiel:

Betrachte Beispiel auf dem 13. Übungsblatt

Berechnung auf $c \dots cdc \dots cd$

Kellerinhalt	Operation
$I_0cI_3cI_3 \dots cI_3$	shift c's bis d gelesen
$I_0cI_3cI_3 \dots cI_3dI_4$	Look–ahead in I_4 betrachten. Falls nicht c oder d, dann Syntaxfehler (Bsp. \$) $\Rightarrow$ gelesen wird c, also red. $C \to d$.
$I_0cI_3cI_3 \dots c \underbrace{I_3CI_8}_{entkellern}$	red. $C \to cC$ (richtiger Look–ahead).
$\vdots$	
$I_0cI_3cI_8$	
$I_0cI_2cI_6cI_6 \dots I_6dI_7 \dots$	und wieder Keller entleeren.

Änderung: Verschmelze I_4 und I_7 zu einem Zustand

$I_{47} = \{[C \to d\cdot, c, d, \$]\}$.

Klar: I_4 meldet bei Vorschau \$ einen Fehler. I_{47} meldet keinen Fehler!

Bemerkung: Eine Fehlermeldung kriegen wir nach der Modifikation etwas später.

Definition: Kern

Als *Kern* eines LR(1)–Elementes verstehen wir dessen erste Komponente. Sei S eine Menge von LR(1)–Elementen, dann ist

$$\mathrm{Kern}(S) = \{\mathrm{Kern}(E) \mid E \in S\}.$$

LALR–Parsingtabelle

$\forall$ Paare S_1, S_2 – Zustände: $\mathrm{Kern}(S_1) = \mathrm{Kern}(S_2) \Rightarrow$ verschmelze S_1, S_2 zu $S_1 \cup S_2$.

Was passiert mit den Übergängen (Pfeilen)?

Bemerkung: Die GOTO()–Funktion hängt nur vom Kern ab, der look–ahead wird nur „mitgeschleppt“:

$$\begin{array}{ccccc}
\boxed{[A \to \alpha \cdot X\beta, \$]} & & \boxed{[A \to \alpha \cdot X\beta, b]} & & \text{LALR} \\
\big\downarrow X & & \big\downarrow X & & X \\
\boxed{[A \to \alpha X \cdot \beta, \$]} & & \boxed{[A \to \alpha X \cdot \beta, b]} & & \text{LALR}
\end{array}$$

Ist die so entstandene neue Tabelle konfliktfrei oder nicht?

Sei gegeben: $[A \to \alpha\cdot, a] \in S_1$,
$[B \to \beta \cdot a\gamma, b] \in S_2$
$[B \to \beta \cdot a\gamma, c] \in S_1$ für irgendwelche c.

Dann ist ein Konflikt zwischen $[A \to \alpha, a]$ in S_1! Shift–reduce–Konflikt.
Ebenso:

$S' \to S$, $S \to aAd \mid bBd \mid aBe \mid bAe$, $A \to C$, $B \to C$

$L = \{acd, ace, bcd, bce\}$

Betrachte viable Präfixe $\quad ac : S_1 = \{[A \to C\cdot, d], [B \to C\cdot, e]\}$
$\qquad bc : S_2 = \{[A \to C\cdot, e], [B \to C\cdot, d]\}$

S_1 und S_2 sind gültige LR(1)–Elemente $aA \cdot d \to \underbrace{ac}_{\gamma} \cdot d$

S_1 und S_2 sind konfliktfrei, $S_1 \cup S_2$ aber nicht: red–red–Konflikt

$\Rightarrow$ nicht jede LR(1)–Grammatik ist LALR(1)–Grammatik.

Mengentheoretisch: $\quad \text{SLR} \subsetneqq \text{LALR}(1) \subsetneqq \text{LR}(1)$

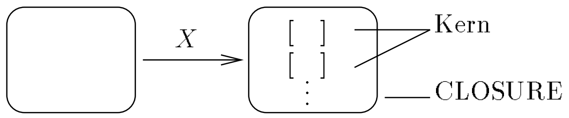
1. $\text{LALR}(1) \subsetneqq \text{LR}(1)$ ist offensichtlich, folgt aus Konstruktion
2. $\text{SLR} \subsetneqq \text{LALR}(1)$:

Lemma:

Für jedes S – Zustand für LR(1) – ist Kern(S) ein Zustand für SLR.

<u>Beweis:</u>

– folgt aus Konstruktion:



Konstruktion der LALR–Parsing–Tabelle

$\to$ durch Modifikation der LR(1)–Tabelle. Dazu muß zuerst die größere LR(1)–Tabelle erzeugt werden – nicht effizient.

$\to$ bessere Technik (sparsamer im Platzbedarf) – später

<u>Mehrdeutige Grammatiken</u>

Beispiel: Arithmetisch Ausdrücke: $E \to E + E \mid E * E \mid (E) \mid id$

— diese Grammatik ist intuitiv, die LR(0)–Grammatik für die Sprache nicht.

1. Fall:

I_1

$E \to E + E\cdot$
$E \to E \cdot +E$
$E \to E \cdot *E$

$E + E \mid *$
Konflikt!

2. Fall:

I_2

$E \to E + E$

$E + E \mid +$

Operationen sind
– rechtsassoziativ oder
– linksassoziativ oder
– nicht assoziativ.

Lösung: Jeder Operator hat Priorität.

Allgemein:

Jedes Terminalzeichen besitzt Priorität.

Jede Produktion besitzt Priorität:

Normalerweise:

Priorität der Produktion = Priorität des letzten Terminals

$X \to E + E * E$

Implementierung der LR–Parsing–Tabelle:

Zustände	ACTION	GOTO
	$abcd\ldots \in \Sigma$	$XYZ \in V$
1		
2		
$\vdots$		

Geg. Kellerinhalt:	$Y s_0 X_1 s_1 \ldots X_k s_k$	und $A \to X_1 \ldots X_k$
Dann ist	$Y s_0 A$„GOTO(s_0, A) “	neuer Kellerinhalt.

ACTION–Teil der Parsing–Tabelle

- viele Zeilen sind identisch, für jeden Zustand haben wir pro Zeile einen Zeiger.

GOTO–Teil der Parsing–Tabelle

- für ein Zeichen gibt es wenig Einträge in einer Spalte, jede Spalte ist als Liste implementiert.

Alternative Methode:

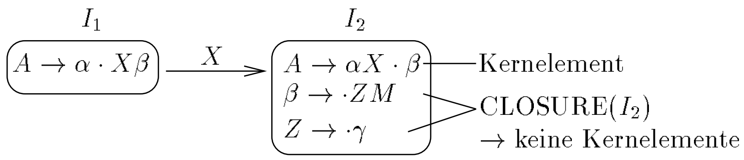
Berechnung der LALR–Parsing–Tabelle:

Zustände müssen berechnet werden.

Definition: Kernelemente

Kernelemente sind Elemente, die nicht durch CLOSURE–Operation berechnet werden.

Beispiel:



Es reicht, die Kernelemente für einen Zustand zu bestimmen.

Die Kernelemente können so aussehen:

– $[S' \to \cdot S, \$]$ ist immer ein Kernelement.

– $[A \to \alpha \cdot \beta, a]$, wobei $\alpha \neq \varepsilon$

– Jedes Element, das kein Kernelement ist, fängt mit $\cdot$ an.

Wie kann man bei dieser Darstellung die Pfeile im Graph berechnen?

GOTO: Shift auf a – $[A \to \alpha \cdot B\beta, b]$, wobei $B \stackrel{*}{\Rightarrow} ax$ ($\Rightarrow a \in \mathrm{FIRST}(B)$)

Dann kann man Shiften.

Reduktion:

Jede Produktion, die für eine Reduktion verwendet wird, hat einen Punkt auf der rechten Seite, ist also Kernelement.

Ausnahme: $A \to \varepsilon$, dann gibt es

$[B \to \beta \cdot C\gamma, b]$ im Kern, so daß $C \stackrel{*}{\Rightarrow} A\delta$ und $a \in \mathrm{FIRST}(\delta\gamma b)$

Verschiedene Look-ahead–Zeichen

1. Spontane Look–ahead–Zeichen: $A \to B \cdot CD$ im Kern, $C \stackrel{*}{\Rightarrow} E\alpha$ und $E \to X\beta$, $b \in \mathrm{FIRST}(\alpha D)$, dann $[E \to X \cdot \beta, b]$ und b heißt spontan (hängt nicht von Element $A \to B \cdot CD$ ab).
2. Propagierende Look–ahead–Zeichen: $\alpha D \stackrel{*}{\Rightarrow} \varepsilon$ (im voraus berechenbar). Dann $[E \to X \cdot \beta, a]$, a propagierender Look-ahead. $A \to \varepsilon \mid (X)$

3.10 LR(k)–Grammatiken

Definition:

Eine Grammatik heißt LR(k), falls folgendes gilt:

- $S_0 \stackrel{*}{\Rightarrow} p_1 X p_3 \Rightarrow p_1 p_2 p_3$ und
- $S_0 \stackrel{*}{\Rightarrow} p'_1 X p'_3 \Rightarrow p'_1 p'_2 p'_3$ und
- die ersten $(p_1 p_2) + k$ Zeichen aus $p_1 p_2 p_3$
  $=$ die ersten $(p_1 p_2) + k$ Zeichen aus $p'_1 p'_2 p'_3$
  $\Rightarrow p_1 = p'_1$, $X = X'$, $p_2 = p'_2$

Satz:

$\forall k$ gibt es $(k+1)$–Grammatiken, die nicht LR(k) sind.

4 Eigenschaften von deterministisch kontextfreien Sprachen

Definition: Normalform

Ein deterministischer Kellerautomat ist in Normalform, falls

- M jede Eingabe vollständig liest
- M mit Hilfe der akzeptierenden Zustände akzeptiert
- $\delta(q, a, X) = (p, \gamma)$, dann $\gamma = \varepsilon \vee \gamma = X \vee \gamma = YX$

Satz:

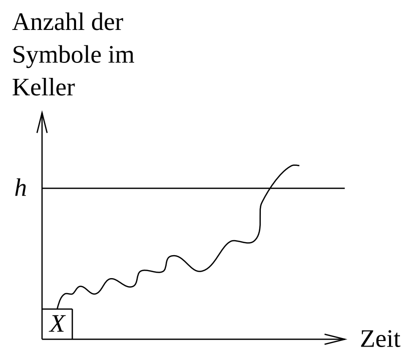
Zu jedem deterministischen Kellerautomaten M existiert ein äquivalenter deterministischer Kellerautomat M' in Normalform.

<u>Beweis:</u>

1. Schritt:
   Ersetze $A \to X_1 \cdots X_k$
   durch $A \to X_1 P_1$, $P_1 \to X_2 P_2$, ..., $P_{k-2} \to X_{k-1} P_{k-1}$, $P_{k-1} \to X_k$.
   (Analogie: Chomsky–Normalform)
2. Schritt:
   Das oberste Kellersymbol wird in die Zustandsmenge eingebaut.

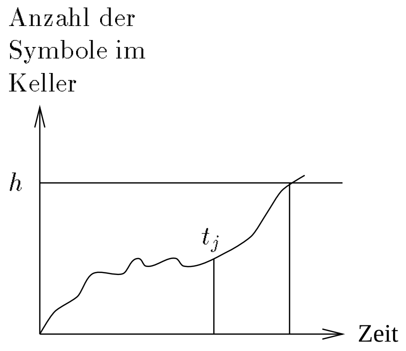
Keller	Zustand		Keller	Zustand
$X_1 \cdots X_k$	q	$\rightsquigarrow$	$X_2 \cdots X_k$	(q, X_1)

   $\delta((q, X_1), a, X_2) = ((p, Y), X_2)$
3. Schritt:
   Vorzeitiges Stoppen verhindern, d.h.
   $\delta(q, \varepsilon, U) \cup \delta(q, a, U) = \varnothing \quad (a \in \Sigma)$
   Idee: Neuer Zustand beim Lesen von a:
   $\delta(q, a, U) = (q_{neu}, U)$ und $\forall b, X : \delta(q_{neu}, b, X) = (q_{neu}, X)$
4. Schritt:
   Unendliche Berechnungen verhindern
   Solche Berechnungen müssen unendlich viele ε–Bewegungen enthalten; wir fangen bei (q, ε, X) an:



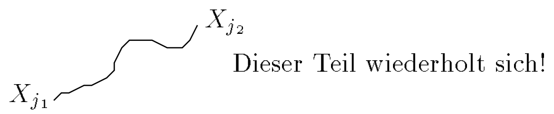
Fall 1: $h > \#(\text{Zustände}) \cdot \#(\text{Kellersymbole})$

Sei $z := \#(\text{Zustände})$ und sei $s := \#(\text{Kellersymbole})$. Für alle j sei t_{j-1} der letzte Schritt, in dem der Keller $j-2$ Symbole über X enthält:



(q, X_j) sind Zustand und oberstes Kellersymbol nach Schritt t_j. Dann gilt: $\exists j_1, j_2 : (q_{j_1}, X_{j_1}) = (q_{j_2}, X_{j_2})$

Graphisch:



Fall 2: $h \leq z \cdot b$ mit $b = \#(\text{Kellersymbole})$

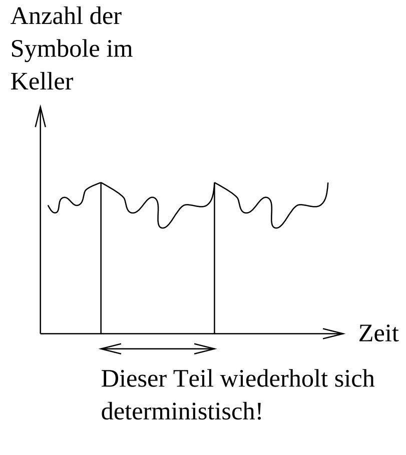
Bezeichne M die Anzahl der möglichen Konfigurationen, also:

$$M := z \cdot (b + b^2 + \cdots + b^{z \cdot b})$$

Es gibt zwei Fälle

a) höchstens M ε–Bewegungen sind durchgeführt worden.

b) mehr als M ε–Bewegungen sind durchgeführt worden $\Rightarrow$ mindestens eine der Konfigurationen wiederholt sich:



Korollar:

Man kann im voraus berechnen, welche (q, ε, X) zu einer unendlichen Reihe von ε–Bewegungen führen.

<u>Beweis:</u>

Konstruktiv, durch Modifikation des Automaten.

In Zustand q mit Kellersymbol X:

Bevor ein ε–Schritt durchgeführt wird, wird überprüft, ob eine unendliche Berechnung mit ε–Schritten in dieser Konfiguration (q, ε, X) anfängt. Falls ja, gehe in speziellen Zustand, der das Ende der Berechnung bedeutet. Falls nein, simuliere weiter.

Satz:

Die Menge der dkfS ist gegen Komplement abgeschlossen.

Beweis:

Vermutung:

det. KA M		neuer det. KA M'
akz. Zustand	$\rightsquigarrow$	nicht akz. Zustand
nicht akz. Zustand	$\rightsquigarrow$	akz. Zustand

Problem: Berechnungen, die unendlich viele ε–Schritte machen.

Annahme: M ist in Normalform.

Dann gibt es, wie oben gesehen, keine unendlichen ε–Bewegungen.

Weiteres Problem: Was passiert am Ende?

Beispiel: Nach dem Lesen der Eingabe gäbe es folgende Zustände

M :	$\underline{q_1}q_2\underline{q_3}$	$\Rightarrow M$ akzeptiert Eingabe $[q_1,\ q_3$ akz.$]$
M' :	$q_1\underline{q_2}q_3$	$\Rightarrow M'$ akzeptiert Eingabe $[q_2$ akz.$]$ (Widerspruch!)

Letzte Modifikation:

Sei Q die Zustandsmenge von M.

$Q \to Q \times \{1, 2, 3\}$, wobei $q \mapsto (q, 1), (q, 2), (q, 3)$ mit folgender Bedeutung:

$(q, 1)$	nach dem Lesen des letzten gelesenen Zeichens wurde kein akzeptierender Endzustand erreicht.
$(q, 2)$	Gegenteil
$(q, 3)$	aus $(q, 2)$ vor dem Lesen des nächsten Zeichens

Akzeptierende Zustände sind $(q, 3) \in Q'$.

Korollar:

Nicht alle kfS sind dkfS.

Beweis:

Nicht alle kfS sind gegen Komplement abgeschlossen.

Satz:

Die dkfS sind abgeschlossen gegen

- inverser Homomorphismus
- Durchschnitt mit regulären Sprachen

Die dkfS sind nicht abgeschlossen gegen

1. Durchschnitt
2. Vereinigung
3. Homomorphismen
4. Konkatenation
5. Kleene–Abschluß

Beweis:

1. $L_1 = \{a^n b^n c^m \mid n, m \in \mathbb{N}\}$, $L_2 = \{a^n b^m c^m \mid n, m \in \mathbb{N}\}$ sind deterministisch kontextfreie Sprachen.

   $L_1 \cap L_2 = \{a^n b^n c^n \mid n \in \mathbb{N}\}$ ist nicht deterministisch kontextfrei (Pumping–Lemma), also sind die dkfS nicht abgeschlossen bzgl. Durchschnitt.
2. Nach deMorgan gilt: $L_1 \cap L_2 = \overline{\overline{L_1} \cup \overline{L_2}}$

   Wären die dkfS abgeschlossen bzgl. Vereinigung, so wären sie auch abgeschlossen bzgl. Durchschnitt (Widerspruch zu 1.).

   (Hinweis: Wegen der Abgeschlossenheit bzgl. Komplementbildung bei dkfS, spielt diese keine Rolle bei obiger Umformung.)
3. Seien L_1, L_2 zwei deterministisch kontextfreie Sprachen mit $L_1 \cup L_2$ ist nicht deterministisch kontextfrei (existieren nach 2.). Sei L_3 wie folgt definiert:

   $$L_3 = \{\#\}L_1 \cup \{\$\}L_2 \quad (\#, \$ \text{ „Umschalter“})$$

   L_3 ist wieder eine deterministisch kontextfreie Sprache. Nun läßt sich folgender Homomorphismus h definieren:

   $$h(x) = \begin{cases} \varepsilon & \text{wenn } x = \# \text{ oder } x = \$ \\ x & \text{sonst} \end{cases}$$

   $h(L_3) = L_1 \cup L_2$, daher ist $h(L_3)$ nicht deterministisch kontextfrei, also sind die dkfS nicht abgeschlossen bzgl. Homomorphismus.
4. $L_4 := \{\#\}L_1 \cup L_2$ ist dkfS, $\#^*$ ist reguläre Sprache.

   $L_5 := \#^* L_4 = \#^+ L_1 \cup \#^* L_2$.

   Wäre L_5 deterministisch kontextfrei, dann wäre auch

   $L_5 \cap \underbrace{\#a^*b^*c^*}_{regS} = \{\#\}L_1 \cup \{\#\}L_2$ deterministisch kontextfrei!

   Widerspruch, da $\#L_1 \cup \#L_2$ nicht deterministisch kontextfrei ist.
5. $L_6 := \{\#\} \cup L_4$ ist dkfS. Ist L_6^* deterministisch kontextfrei?

   $L_6^* \cap \#a^*b^*c^* = \{\#\} \cup \{\#\}L_1 \cup \{\#\}L_2$ ist nicht deterministisch kontextfrei!

   $\Rightarrow L_6^*$ ist nicht deterministisch kontextfrei.

5 Entscheidbarkeitsfragen für kontextfreie Sprachen

Bemerkung:

Bekannt aus Theo.Inf: Halteproblem H für TM ist unentscheidbar.

Definition: Postsche Korrespondenzproblem (PKP)

Gegeben:	$((x_1, y_1), (x_2, y_2), ..., (x_k, y_k))$ mit (x_i, y_i) Worte über Σ und $\lvert\Sigma\rvert < \infty$
Frage:	Gibt es Indizes $i_1, ..., i_n$ (n beliebig), so daß gilt $x_{i_1}x_{i_2}...x_{i_n} = y_{i_1}y_{i_2}...y_{i_n}$

Modifiziertes Korrespondenzproblem(MPKP): verlange $i_1 = 1$

Definition: Reduzierbarkeit

X,Y seien Sprachen. X ist reduzierbar auf Y (X$\leq$Y), falls eine total rekursive Funktion f, mit $f : \Sigma^* \to \Sigma^*$, existiert, so daß $\forall w \in X$ gilt:

$$w \in X \Leftrightarrow f(w) \in Y$$

Bemerkung:

Ist Y entscheidbar, dann ist X entscheidbar.

Ist X unentscheidbar, dann ist Y unentscheidbar.

zu zeigen also: H $\leq$ MPKP $\leq$ PKP

Lemma:

Es gilt: MPKP$\leq$PKP

Beweis:

Definiere eine Funktion f, so daß gilt:

$$\underbrace{((x_1, y_1), (x_2, y_2), ..., (x_k, y_k))}_{\text{Lösung in MPKP}} = \underbrace{((x'_0, y'_0), (x'_1, y'_1), (x'_2, y'_2), ..., (x'_k, y'_k))}_{\text{Lösung in PKP}}$$

Definiere

$$\begin{aligned} \forall i \neq 0, i \neq k+1 : x_i = a_1 \dots a_s &\Rightarrow x'_i = a_1\#a_2\# \dots a_s\# \\ y_i = b_1 \dots b_r &\Rightarrow y'_i = \#b_1\#b_2 \dots \#b_r \\ x'_0 = \#x'_1, y'_0 &= y'_1 \\ x'_{k+1} = \$, y'_{k+1} &= \#\$ \end{aligned}$$

Sei folgende Lösung für MPKP gegeben: $x_1x_{i_2} \dots x_{i_n} = y_1y_{i_2} \dots y_{i_n}$

Dann ergibt die Funktion f

$$\begin{aligned} x'_0x'_{i_2} \dots x'_{i_n}x'_{k+1} &= \#a_1\#a_2\# \dots \#a_n\#\$ \\ y'_0y'_{i_2} \dots y'_{i_n}y'_{k+1} &= \#a_1\#a_2\# \dots \#a_n\#\$ \quad \Rightarrow \text{Lösung in PKP} \end{aligned}$$

umgekehrt: $x'_{i_1} \dots x'_{i_n} = y'_{i_1} \dots y'_{i_n}$:

- $i_1 = 0$, da $x_i, y_i, \forall i > 0 : x_i \neq y_i$ (Buchstabe oder # und umgekehrt)
  $\Rightarrow x'_0x'_{i_2} \dots x'_{i_n} = y'_0y'_{i_2} \dots y_{i_n}$

- $i_n = k+1$, wegen \$
- $i_j = 0$ für j $\neq 1 \Rightarrow$ es gibt benachbarte ## im ersten Wort, das ist zweiten Wort nicht möglich.
- Man kann beim ersten Auftreten von x'_{k+1}, y'_{k+1} die Worte beenden. Nach dem Weglassen der # gilt dann:

  $x_1 x_{i_2} \ldots x_{i_{n-1}} = y_1 y_{i_2} \ldots y_{i_{n-1}}$

Satz:

Es gilt: H $\leq$ MPKP

Beweis: (Idee)

F wandelt eine Folge von Konfigurationen der Turingmaschine, die simuliert wird ($H = \{< M > x \mid$ M hält nicht auf x$\}$), in eine Lösung für MPKP um. Wir bauen zwei Zeichenketten:

Konfiguration 1 # Konfiguration 2 ... letzte Konfiguration

Konfiguration 1 # Konfiguration 2 ...

$\leftrightsquigarrow (b, q, a)\#b, q \leftrightsquigarrow \quad x_i = q \;\; y_i = qa$
$\leftrightsquigarrow (b, q, a) \quad x_i = q \;\; y_i = bq$

Daraus folgt dann, daß MPKP und PKP unentscheidbare Probleme sind. Dies macht man sich bei folgenden Entscheidungsfragen zunutze.

Satz:

Das Schnittproblem für kfS ist unentscheidbar:

Eingabe:	G_1, G_2 kfG
Frage:	Ist $L_{G_1} \cap L_{G_2} = \emptyset$?

Beweis: PKP $\leq$ Schnittproblem ($\Rightarrow$ Schnittproblem unentscheidbar)

$(x_1, y_1) \ldots (x_k, y_k)$. Wir konstruieren G_1 und G_2.

$G_1 : S_1 \to a_1 x_1 \mid a_2 x_2 \mid \ldots \mid a_k x_k \mid a_1 S_1 x_1 \mid a_2 S_1 x_2 \mid \ldots \mid a_k S_1 x_k$
$G_2 : S_2 \to a_1 y_1 \mid a_2 y_2 \mid \ldots \mid a_k y_k \mid a_1 S_2 y_1 \mid a_2 S_2 y_2 \mid \ldots \mid a_k S_2 y_k$

Betrachte folgende Sätze: $\to$
In $G_1 : S_1 \Rightarrow a_{i_1} S_1 x_{i_1} \Rightarrow a_{i_1} a_{i_2} S_1 x_{i_2} x_{i_1} \overset{*}{\Rightarrow} a_{i_1} \ldots a_{i_n} x_{i_n} x_{i_1}$
In $G_2 : S_2 \Rightarrow a_{i_1} S_2 y_{i_1} \Rightarrow a_{i_1} a_{i_2} S_2 y_{i_2} y_{i_1} \overset{*}{\Rightarrow} a_{i_1} \ldots a_{i_n} y_{i_n} y_{i_1}$
Sei $w \in L_1 \cap L_2$. Dann gilt

$$\left.\begin{array}{l} w = a_{i_1} \ldots a_{i_n} x_{i_n} \ldots x_{i_1} \\ w = a_{i_1} \ldots a_{i_n} y_{i_n} \ldots y_{i_1} \end{array}\right\} \Rightarrow \forall i : x_i = y_i$$

Also $x_{i_n} \ldots x_{i_1} = y_{i_n} \ldots y_{i_1}$,
insbesondere $((x_{i_n}, y_{i_n}), \ldots, (x_{i_1}, y_{i_1})) \in$ PKP
Also gilt $w \in$ PKP $\Leftrightarrow w \in L_1 \cap L_2$.

Satz:

Eingabe:	G kfG
Frage:	Ist G mehrdeutig? ist unentscheidbar.

Beweis: wieder PKP $\leq$ Frage

Wir nehmen G_1, G_2 aus vorherigem Problem.
Daraus definiere $G := G_1 \cup G_2 \cup \{S \to S_1 \mid S_2\}$
Sei $w = a_{i_1} \dots a_{i_s} x_{i_s} \dots x_{i_1}$. Läßt sich w auf zwei Arten ableiten:

$$\left.\begin{array}{l} S \Rightarrow S_1 \Rightarrow a_{i_1} S_1 x_{i_1} \stackrel{*}{\Rightarrow} a_{i_1} \dots a_{i_s} x_{i_s} \dots x_{i_1} \\ S \Rightarrow S_2 \Rightarrow a_{i_1} S_2 x_{i_1} \stackrel{*}{\Rightarrow} a_{i_1} \dots a_{i_s} x_{i_s} \dots x_{i_1} \end{array}\right\} \Rightarrow \text{mehrdeutig}$$

Insbesondere gilt: PKP hat Lösung $\Leftrightarrow$ G mehrdeutig.

Bemerkung:

Rechenwege einer Turingmaschine

$w\#w\#\dots$: für kfG schwer.

$w_1\#w_2^R\#w_3\#w_4^R\#\dots$

$\uparrow$ 1. Konfiguration $\qquad w_i$: Konfiguration vor i. Schritt

Lemma:

B_M : Menge der Rechenwege einer Turingmaschine M
B_M : Durchschnitt von 2 kontextfreien Sprachen

Beweis: (Begründung)

Konstruiere zwei Maschinen M_1 und M_1.
M_1 untersucht, ob w_1 und $w_2 \dots$ übereinstimmen.
M_2 untersucht, ob w_2 und $w_3 \dots$ übereinstimmen.

Lemma:

B_M kontextfrei $\Leftrightarrow$ L_M endlich.

Beweis:

($\Leftarrow$) L_M endlich, dann auch B_M, also kontextfrei.

($\Rightarrow$): Ogden Lemma

Sei L_M unendlich. Sei n die Konstante aus Ogden's Lemma

$\gamma = w_1\#w_2^R\#w_3\#\dots\ |w_2| > n$, wir markieren $\#w_2^R\#$.

Sei $\gamma = uvwxy$;

vx enthält mindestens einen markierten Buchstaben

vwx enthält höchstens n markierte Buchstaben

und $uv^iwx^iy \in B_M$.

- wird ein Teil von w_1 gepumpt $\Rightarrow$ kein Teil von w_3 wird gepumpt
- wird ein Teil von w_3 gepumpt $\Rightarrow$ kein Teil von w_1 wird gepumpt
- wird w_3 gepumpt $\Rightarrow$ M wäre nichtdeterministisch.

$\Rightarrow$ w_1 wird gepumpt:

- v^i : das vergrößerte w_1, x^i : das vergrößerte w_1, w_2^R oder beide.
- w_3 bleibt unverändert.

Aber die Länge von 1,2,3 Konfigurationen müssen fast gleich sein.

Satz:

Eingabe:	G_1, G_2 kfG
Frage:	Ist $L_{G_1} \cap L_{G_2}$ kfS? ist unentscheidbar.

Beweis:

Man nehme G_1, G_2, so daß $L_{G_1} \cap L_{G_2} = B_M$ ist.

$$\begin{array}{ccc} \text{Ist } B_M \text{ kontextfrei?} & \Leftrightarrow & \text{Ist } L_{G_1} \cap L_{G_2} \text{ kontextfrei?} \\ \Updownarrow & & \Updownarrow \\ \text{Ist } L_M \text{ endlich?} & & \text{Ist } L_M \text{ endlich?} \end{array}$$

Diese Frage ist für TM's bekanntlich unentscheidbar.

Lemma:

$\forall$ TM M ist das Komplement von B_M kontextfrei.

Beweis: (Begründung)

Eine Folge w gehört zu $\overline{B_M}$, falls

- w ist nicht in der Form $w_1\#w_2\#w_\# \ldots$
- das Komplement einer regulären Sprache ist auch regulär.
- w ist wohlgeformt aber $\exists\, i : w_{i+1}^R$ ist keine direkte Nachfolgekonfiguration von w_i durch nichtdeterministischen Kellerautomaten.

Satz:

Eingabe:	G kfG
Frage:	Ist $\overline{L_G}$ kfS? ist unentscheidbar.

Beweis:

Annahme: ... ist entscheidbar.

M sei TM. $\overline{B_M}$ ist kontextfrei. Gemäß Annahme kann man entscheiden, ob $\overline{\overline{B_M}} = B_M$ kontextfrei ist.

Es geht nur, wenn L_M endlich ist! $\natural$

Satz:

Eingabe:	G_1, G_2 kfG
Frage:	Ist $L_{G_1} = L_{G_2}$ kfS? ist unentscheidbar.

(ohne Beweis)

6 Anhang

6.1 Zusammenfassung: Abschlußeigenschaften

	regS	kfS	dkfS
$\cap$	ja	nein	nein
$\cup$	ja	ja	nein
$*$	ja	ja	nein
Komplement	ja	nein	ja
Substitution	ja	ja	nein
Homomorphismus	ja	ja	nein
inv. Homomorph.	ja	ja	ja
$\cap$ regS	ja	ja	ja
Kleene–Abschluß	ja	ja	nein

6.2 Zusammenfassung: Entscheidbarkeitsfragen

	regS	kfS	dkfS
$L_M = \emptyset$?	P	P	
$L_M = \infty$?	P	P	
$L_M = \Sigma^*$?	P	n.r	
$L_G = L_{G'}$?	P	n.r.	?
$\overline{L_G} < \infty$?	P	n.r.	
G eindeutig?	immer	n.r.	
$L_1 \cap L_2$ kfS?	–	n.r.	
G mehrdeutig?	–	n.r.	
$\overline{L_G}$ kfS?	–	n.r.	
$L_1 \cap L_2 = \emptyset$?	–	n.r.	

6.3 Literatur

[1] Hopcroft, J.E., Ullman, J.D. *Einführung in die Automatentheorie, Formale Sprachen, Komplexitätstheorie.* Addison–Wesley, 1992.

[2] Aho, A.V., Sethi, R., Ullman, J.D. *Compilerbau, Teil 1.* Addison–Wesley, 1995.

[3] Wegener, I. *Theoretische Informatik.* Teubner, 1993.

[4] Schöning, U. *Theoretische Informatik kurzgefasst.* BI–Wissenschaftsverlag, 1992.