Das Pumping–Lemma (uvwxy–Theorem)

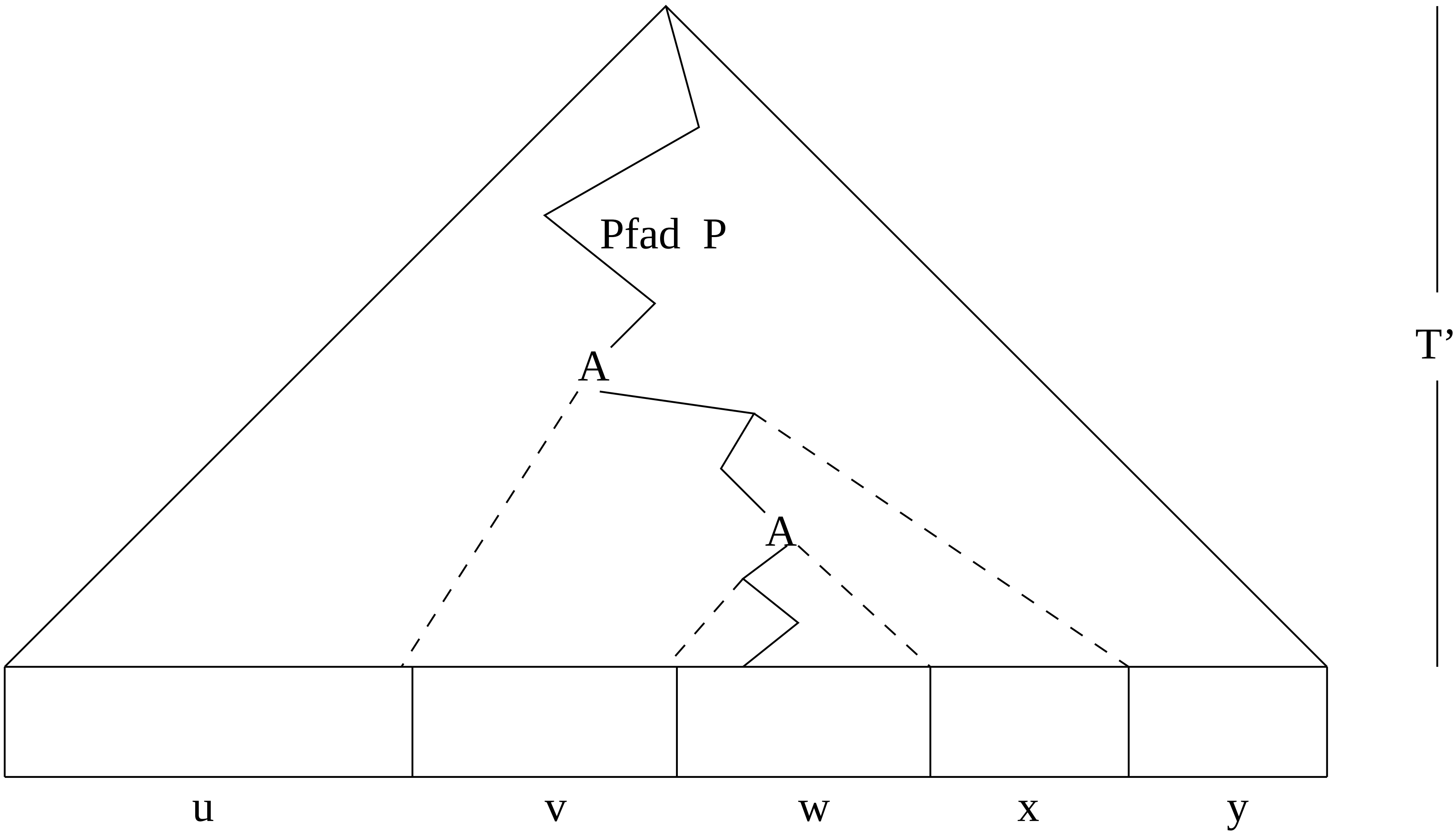
Satz: Für jede kontextfreie Sprache $L \subseteq \Sigma^*$ gilt:

$$\exists n \geq 1,$$
$$\forall z \in L \text{ mit } |z| \geq n,$$
$$\exists u, v, w, x, y \in \Sigma^* \text{ mit } z = uvwxy, 1 \leq |vx| \leq |vwx| \leq n, \quad .$$
$$\forall i \geq 0 \quad :$$
$$uv^iwx^iy \in L$$

Es folgt:

- der formale Beweis
- plus eine Veranschaulichung

- Benutze eine kontextfreie Grammatik G für $L \setminus \{\varepsilon\}$ in Chomsky Normalform und wähle $k := |V|$ (Anzahl der Variablen) und $n := 2^k$.
- Zu einem Wort $z = a_1 \cdots a_s \in L$ mit $a_i \in \Sigma$ und $s \geq n$ betrachte den Syntaxbaum T mit Beschriftung z und den binären Teilbaum T', der von den inneren Knoten von T induziert wird.
- Da T' (wie auch T) $s \geq 2^k$ Blätter hat, folgt $h :=$ (Höhe von T') $\geq k$.
- Betrachte einen Pfad P der Länge $h \geq k$ (also mit $h+1 \geq k+1$ Knoten) in T'. Gemäß Schubfachprinzip sind zwei der $k+1$ letzten Knoten von P mit derselben Variable, sagen wir A, markiert.
- Die Ableitung von z aus dem Startsymbol S hat demnach die folgende Form:
$$S \Rightarrow_G^* \overbrace{uAy \Rightarrow_G^* uvAxy}^{zyklischer\ Teil} \Rightarrow_G^* uvwxy = z$$
für geeignet gewählte Wörter u, v, w, x, y mit $1 \leq |vx| \leq |vwx| \leq n$.
- Durch i–fache Iteration (oder Weglassen) des zyklischen Ableitungsteils sehen wir, dass $uv^iwx^iy \in L$ für alle $i \geq 0$.



Baum T mit Beschriftung z = uvwxy

Anwendung: Nachweis der Nichtkontextfreiheit

Folgerung: Falls für eine Sprache $L \subseteq \Sigma^*$ die Bedingung

$$\forall n \geq 1,$$
$$\exists z \in L \text{ mit } |z| \geq n,$$
$$\forall u, v, w, x, y \in \Sigma^* \text{ mit } z = uvwxy, 1 \leq |vx| \leq |vwx| \leq n,$$
$$\exists i \geq 0 \ :$$
$$uv^iwx^iy \notin L$$

erfüllt ist, dann ist L nicht kontextfrei.

Beispiel für eine nicht–kontexfreie Sprache

Die folgende Sprache $L \subseteq \{a,b\}^*$ ist nicht kontexfrei:

$$L = \{a^m b^m c^m \mid m \geq 1\}$$

Begründung:

1. Zu beliebig vorgegebenem $n \geq 1$ wähle $z = a^n b^n c^n$.
   Offensichtlich gilt $z \in L$ und $|z| \geq n$.

2. Zu beliebig vorgegebener Zerlegung $z = uvwxy$ mit $1 \leq |vx| \leq |vwx| \leq n$ wähle $i = 0$. Beachte, dass vx wegen $|vwx| \leq n$ das Zeichen a oder das Zeichen c nicht enthält.

Nun gilt

$$z' = uv^0wx^0y = uwy \notin L \ ,$$

da die notwendige Bedingung

$$|z'|_a = |z'|_b = |z'|_c$$

nicht erfüllt sein kann.

Der Fall unärer Sprachen

Vereinfachte Form des Pumping Lemmas: Für jede kontextfreie Sprache $L \subseteq \{0\}^*$ gilt:

$$\exists n \geq 1,$$
$$\forall m \geq n \text{ mit } 0^m \in L$$
$$\exists 1 \leq l \leq n \quad .$$
$$\forall i \geq 0 \ :$$
$$0^{m-l}0^{il} \in L$$

denn: Für die Zerlegung $0^m = uvwxy$ aus dem Pumping Lemma wähle $l := |vx|$ so dass $vx = 0^l$.

Der Fall unärer Sprachen (fortgesetzt)

Satz: Jede kontextfreie Sprache über einem einelementigen Alphabet ist sogar regulär.

Folgerung: Die folgenden Sprachen sind nicht kontextfrei:

$$
\begin{aligned}
L &= \{0^p \mid p \text{ ist Primzahl}\} \\
L &= \{0^m \mid m \text{ ist Quadratzahl}\}
\end{aligned}
$$

Beweis des Satzes

- Wähle n als die betreffende Konstante aus dem Pumping Lemma.
- $l \geq 1$ heißt „Pumpindex“ **gdw** $\exists m \geq n, \forall i \geq 0 : 0^{m-l}0^{il} \in L$.
- $I \subseteq \{1, \ldots, n\}$ bezeichne die Menge aller Pumpindizes.
- Die Zahl $q := n! \geq n$ ist dann ein Vielfaches aller Pumpindizes.
- r heißt „Pumprest“ **gdw** $\exists m \geq q : 0^m \in L$ und $r = m \bmod q$.
  m_r bezeichne das kleinste m mit dieser Eigenschaft.
- $R \subseteq \{0, \ldots, q-1\}$ bezeichne die Menge aller Pumpreste.
- Die Sprache

$$L' := \overbrace{\{x \in L \mid |x| < q\}}^{endlich} \cup \bigcup_{r \in R} \overbrace{\{0^{m_r + iq} \mid i \geq 0\}}^{0^{m_r}(0^q)^*}$$

  ist regulär und es gilt $L' = L$ (Details an der Tafel).

Der CYK–Algorithmus

Das spezielle Wortproblem für eine kontextfreie Sprache L (mit $\varepsilon \notin L$):

Eingabe: $w = a_1 \cdots a_n$ mit $a_i \in \Sigma$

Frage: $w \in L$?

Der Algorithmus von Cocke, Younger und Kasami benutzt eine kontextfreie Grammatik $G = (V, \Sigma, P, S)$ in Chomsky Normalform für L und die Methode des dynamischen Programmierens zur Berechnung einer Tabelle T:

1. Für $1 \leq i, j \leq n$ berechne
$$T[i,j] := \{A \in V \mid A \Rightarrow_G^* a_i a_{i+1} \cdots a_{i+j-1}\} \ .$$
2. Akzeptiere w **gdw** $S \in T[1,n]$.

Wegen
$$S \in T[1,n] \textbf{ gdw } S \Rightarrow_G^* a_1 a_2 \cdots a_n = w$$
akzeptiert der Algorithmus exakt die Eingaben aus L.

Berechnung der Tabelle

- Das Teilwort

$$w_{i,j} := a_i a_{i+1} \cdots a_{i+j-1}$$

  ist das ab Position i beginnende Teilwort von w der Länge j.

- $T[i, j]$ soll die Menge aller Variablen sein, aus denen das Teilwort $w_{i,j}$ ableitbar ist.

- Die Ableitung eines Teilwortes a_i der Länge 1 aus einer Variablen A ist nur über eine Regel der Form $A \rightarrow a_i$ möglich ist.

- Die Ableitung eines Teilwortes $w_{i,j} = a_i \cdots a_{i+j-1}$ für $j \geq 2$ hat wegen der binären Syntaxbäume eine spezielle Form (s. folgende Illustration).

Berechnung der Tabelle (fortgesetzt)

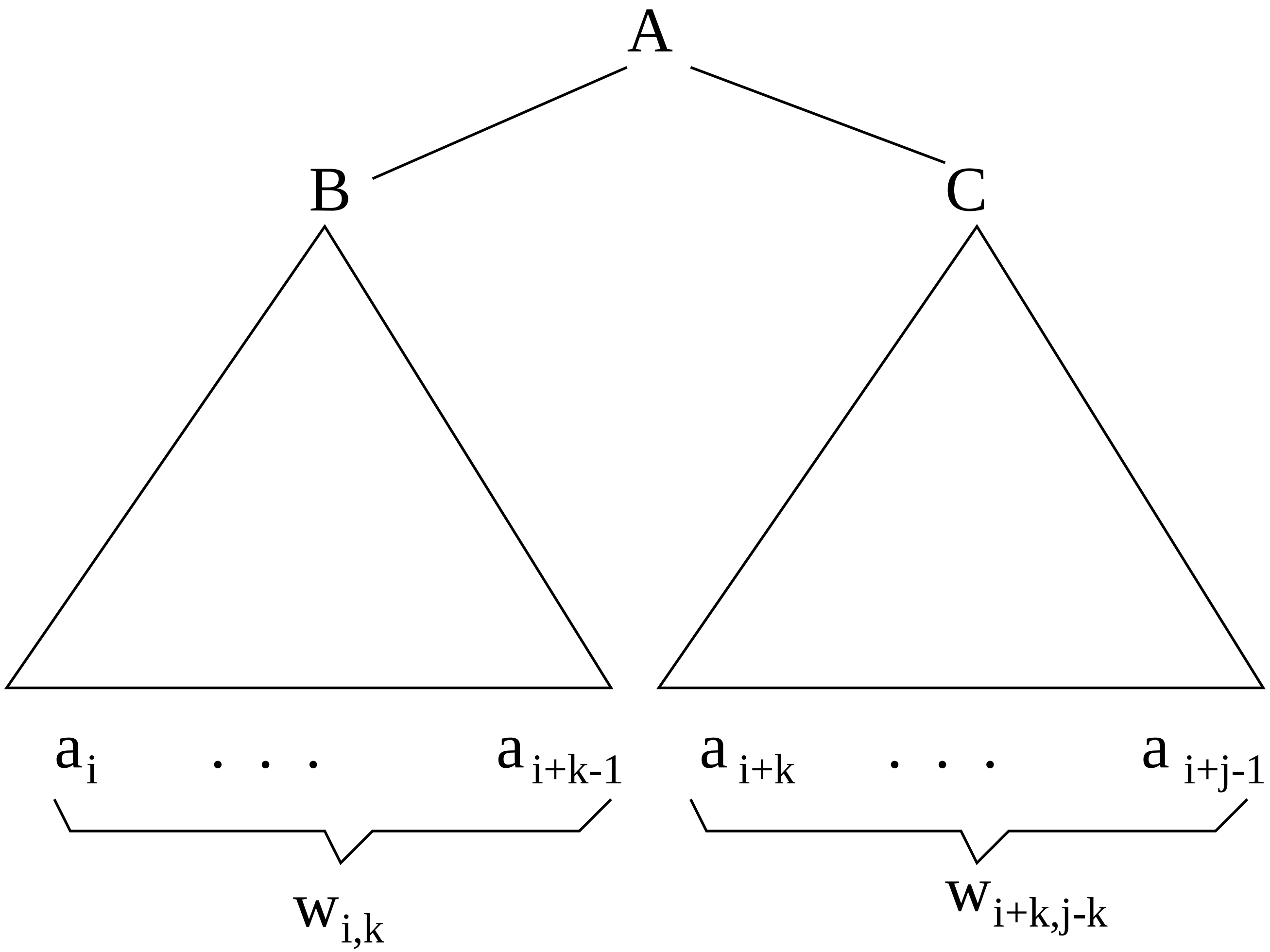


Abbildung 1: Syntaxbaum zur Ableitung $A \Rightarrow_G^* w_{i,j}$.

Berechnung der Tabelle (fortgesetzt)

Tabelle T kann spaltenweise nach folgendem Schema berechnet werden:

$$
\begin{aligned}
T[i,1] &= \{X \in V| \quad X \rightarrow a_i \in P\} \\
T[i,j] &= \{X \in V| \quad \exists 1 \leq k \leq j-1, \exists X \rightarrow BC \in P : \\
&\qquad\qquad B \Rightarrow_G^* w_{i,k} \;,\; C \Rightarrow_G^* w_{i+k,j-k}\} \\
&= \{X \in V| \quad \exists 1 \leq k \leq j-1, \exists X \rightarrow BC \in P : \\
&\qquad\qquad B \in T[i,k] \;,\; C \in T[i+k, j-k]\}
\end{aligned}
$$

berechnet werden.

Beachte, dass die Tabelleneinträge $T[i,k]$ und $T[i+k, j-k]$ zum Zeitpunkt der Berechnung von $T[i,j]$ schon bekannt sind.

Beispiel

Die Sprache $L = \{a^n b^n c^m | \ n, m \geq 1\}$ wird durch folgende kontextfreie Grammatik mit Startvariable S generiert:

$$S \rightarrow AB \ , \ A \rightarrow ab \mid aAb \ , \ B \rightarrow c \mid cB$$

Umformen in Chomsky Normalform ergibt:

$$S \rightarrow AB \ , \ A \rightarrow CD \mid CF \ , \ B \rightarrow c \mid EB \ , \ C \rightarrow a \ , \ D \rightarrow b \ , \ E \rightarrow c \ , \ F \rightarrow AD$$

Beispiel (fortgesetzt)

$S \to AB\ ,\ A \to CD \mid CF\ ,\ B \to c \mid EB\ ,\ C \to a\ ,\ D \to b\ ,\ E \to c\ ,\ F \to AD$

Eingabe $aaabbbcc$ führt zu folgender Tabelle:

$i_\downarrow \mid j \to$	1	2	3	4	5	6	7	8
a	C					A	S	S
a	C			A	F			-
a	C	A	F				-	-
b	D					-	-	-
b	D				-	-	-	-
b	D			-	-	-	-	-
c	E,B	B	-	-	-	-	-	-
c	E,B	-	-	-	-	-	-	-

Statischer versus dynamischer Speicher

Statischer Speicher: ein im „Programm“ festgelegter Speicherblock konstanter Größe; kann nicht dynamisch (zur Laufzeit) und damit auch nicht in Abhängigkeit von der Länge der Eingabe wachsen.

Dynamischer Speicher: kann dynamisch (zur Laufzeit) wachsen und daher auch von der Länge der Eingabe abhängig sein.

- Endliche Automaten verfügen (in Form ihrer Zustandsmenge) nur über einen statischen Speicher.
- Ihre beschränkte Potenz (zum Beispiel die Unfähigkeit Sprachen wie $\{a^n b^n | \ n \geq 1\}$ zu erkennen) ist diesem Umstand geschuldet.

Kellerspeicher

Eine eingeschränkte aber durchaus wirkungsvolle Form des dynamischen Speichers ist der Kellerspeicher (auch einfach „Keller“ genannt);

- In einem Keller können Symbole „gestapelt“ werden.
- Nur oben auf dem Stapel darf ein Symbol draufgelegt (Schreib– bzw. Push–Operation) oder entfernt werden (Lösch– bzw. Pop–Operation).
- Der Stapel darf dynamisch (theoretisch unbegrenzt) wachsen.

Ein Lob der „Hochstapelei“

Das Erkennen der nicht–regulären Sprache

$$\{a^n b^n \mid n \geq 1\}$$

ist unter Verwendung eines Kellerspeichers babyleicht:

Push–Phase: Solange a's gelesen werden, lege diese auf den Stapel

Pop–Phase: Beim Lesen des ersten b's schalte von der Push– auf die Pop–Phase um: pro gelesenem b entferne von dem Stapel ein a.

Akzeptieren/Verwerfen: Akzeptiere **gdw** die Eingabe die Form a^+b^+ hat (was mit der „endlichen Kontrolle“ so nebenbei getestet wird) und der Stapel nach Verarbeitung der Eingabe leer ist.

Beispiel

Bei Eingabe

$$aaaabbbb$$

verändert sich der Stapel wie folgt:

Eingabe	a	a	a	a	b	b	b	b
				a				
			a	a	a			
		a	a	a	a	a		
Stapel	a	a	a	a	a	a	a	

Kellerautomat (informelle Definition)

Ein (nichtdeterministischer) Kellerautomat (englisch: pushdown automaton), kurz PDA ist ein um einen Kellerspeicher erweiterter NFA.

Die formale Definition wird zudem folgende technische Details berücksichtigen:

- Neben dem Eingabealphabet gibt es ein (zum Eingabealphabet nicht notwendig disjunktes) Kelleralphabet (inklusive einem „untersten Kellerzeichen“).
- Ein PDA darf „ε–Transitionen“ vollziehen, in denen kein weiteres Eingabesymbol verarbeitet wird.
- Es gibt nur einen Startzustand (wegen der ε–Transitionen kein echtes „handicap“).
- Die Eingabe wird „mit leerem Keller“ (wie in dem eben gezeigten illustrierenden Beipiel) akzeptiert. (Endzustände sind daher überflüssig.)

Kellerautomat (formale Definition)

$\mathcal{P}_e(U)$ bezeichnet die Menge aller endlichen Teilmengen einer (evtl. unendlichen) Menge U.

Ein PDA M besteht aus den folgenden Komponenten:

- Z, die Zustandsmenge (eine endliche Menge)
- Σ, das Eingabealphabet (ebenfalls endlich)
- Γ, das Kelleralphabet (ebenfalls endlich)
- $\delta : Z \times (\Sigma \cup \{\varepsilon\}) \times \Gamma \rightarrow \mathcal{P}_e(Z \times \Gamma^*)$, die Überführungsfunktion
- $z_0 \in Z$, der Startzustand
- $\# \in \Gamma$, das unterste Kellerzeichen

Konvention: Wir werden des öfteren (ohne dies explizit immer wieder zu erwähnen und hauptsächlich zur Vermeidung lästiger Fallunterscheidungen) mit a ein Element von $\Sigma \cup \{\varepsilon\}$ bezeichnen.

Konventionen

- Der String $B_1 \cdots B_k$ wird im Grenzfall $k = 0$ mit dem leeren Wort identifiziert.
- Der Kellerinhalt wird als String notiert, wobei das „links“ im String „oben“ im Keller bedeutet.

Beachte, dass die Ersetzung von A durch $B_1 \cdots B_k$ im Keller die folgenden Grenzfälle mit einschließt:

Pop–Operation Im Falle $k = 0$ wird A aus dem Keller entfernt.

Ersetzung Im Falle $k = 1$ wird A durch B_1 ersetzt.

Falls $k \geq 1$ ist im Anschluss B_1 das oberste Kellersymbol.

Arbeitsweise des PDA

- Anfangs befindet sich M im Starzustand z_0 und der Keller enthält nur das unterste Kellersymbol $\#$.
- Falls $(z', B_1 \cdots B_k) \in \delta(z, \varepsilon, A)$, dann darf M im Zustand z und bei oberstem Kellersymbol A eine ε–Transition vollziehen, wobei ein Wechsel auf den Zustand z' erfolgt und im Keller das oberste Symbol A durch $B_1 \cdots B_k$ ersetzt wird.
- Falls $(z', B_1 \cdots B_k) \in \delta(z, a, A)$ mit $a \in \Sigma$, dann darf M im Zustand z, bei oberstem Kellersymbol A und beim Lesen des Eingabesymbols a auf den Zustand z' wechseln und im Keller das oberste Symbol A durch $B_1 \cdots B_k$ ersetzen. Anschließend rückt der Lesekopf des Eingabebandes auf das nächste Symbol.
- Die Eingabe wird akzeptiert **gdw** M durch geeignete Wahl der jeweils möglichen Rechenschritte erreichen kann, dass der Keller nach Verarbeitung des letzten Zeichens der Eingabe (+ evtl. weiteren ε–Transitionen) leer ist.

Exkurs: Konfiguration (Momentaufnahme)

In eine Konfiguration für einen (evtl. nichtdeterministischen) Prozess packen wir alle Informationen hinein, die nötig sind, um

- den Prozess zu suspendieren (temporär zu stoppen)
- und ihn später wieder weiterlaufen zu lassen.

Beispiele:

- Informationen die Betriebssysteme von laufenden Prozessen protokollieren (s. Grundvorlesung zur Informatik oder Vorlesung über Betriebssysteme)
- Schachspiel mit Konfiguration bestehend aus der aktuellen Schachstellung und einem Zusatzbit, welches angibt, ob WEISS oder SCHWARZ am Zug ist (+ weitere Details wegen komplizierter Remis–Regeln)

Konfigurationen eines PDA

Die Konfiguration eines PDA besteht aus

- dem aktuellen Zustand $z \in Z$,
- dem noch nicht verarbeiteten Teil $v \in \Sigma^*$ der Eingabe,
- dem aktuellen Kellerinhalt $\alpha \in \Gamma^*$.

Notation: $(z, v, \alpha) \in Z \times \Sigma^* \times \Gamma^*$.

Anfangskonfiguration bei Eingabe w: $(z_0, w, \#)$

Akzeptierende Endkonfiguration: $(z, \varepsilon, \varepsilon)$ für jedes $z \in Z$

`Macht das Sinn ?? Erläutere !!`

Folgekonfigurationen

Eine „Rechnung“ eines PDA lässt sich als Folge von Konfigurationen beschreiben.

Definition

1. (z', v', α') heißt unmittelbare Folgekonfiguration von (z, v, α) **gdw** (z', v', α') kaus (z, v, α) durch einen „Rechenschritt“ (einmalige Verwendung der Überführungsfunktion) resultieren kann.
   Notation: $(z, v, \alpha) \vdash (z', v', \alpha')$.

2. (z', v', α') heißt Folgekonfiguration von (z, v, α) **gdw** (z', v', α') aus (z, v, α) durch eine (evtl. leere) Folge von Rechenschritten resultieren kann.
   Notation: $(z, v, \alpha) \vdash^* (z', v', \alpha')$.
   Formal ist „$\vdash^*$” die reflexive–transitive Hülle von „$\vdash$“.

Sprache eines PDA

Die folgende Definition der vom PDA M erkannten Sprache $N(M)$ entspricht unserer Vereinbarung über das Akzeptieren mit leerem Keller:

$$N(M) := \{w \in \Sigma^* |\ \exists z \in Z : (z_0, w, \#) \vdash^* (z, \varepsilon, \varepsilon)\}$$

In Worten: Wort w gehört zur Sprache $N(M)$ **gdw** M durch Verarbeitung der Eingabe w aus der Anfangskonfiguration in eine akzeptierende Endkonfiguration gelangen kann.

Beispiel

Die Sprache

$$L = \{a_1 a_2 \cdots a_n \$ a_n \cdots a_2 a_1 |\ a_i \in \{a, b\}\}$$

kann durch einen PDA erkannt werden, der

- in einer „Push–Phase“ $a_1, \ldots, a_n$ in den Keller legt (a_n befindet sich dann oben),
- beim Lesen des Trennzeichens „\$“ von der „Push–“ in die „Pop-Phase“ umschaltet,
- in der Pop-Phase beim Lesen von a_i und oberstem Kellersymbol a_i (Match!) das Symbol a_i vom Keller entfernt,
- nach Abarbeitung der Eingabe und bei oberstem Kellersymbol „#“, das Symbol # vom Keller entfernt (um diesen vollständig zu leeren).

Beispiel (fortgesetzt)

Auf diese Weise werden auch Eingaben $w \notin L$ entlarvt. Beim Auftreten von „Störungen“ wie

- Eingabe w hat nicht die Form $w = u\$v$ mit $u, v \in \{a, b\}^*$,
- Auftreten eine „Mismatches“ in der Pop–Phase (gelesenes Zeichen stimmt nicht überein mit dem obersten Kellerzeichen),

wird der Keller (zur Strafe) nicht geleert.

Beachte, dass der Fall des Mismatches die Fälle

$$w = u\$v \ , |u| \neq |v|$$

mit einschließt.

Bei der folgenden Spezifikation des hier nur informell skizzierten PDAs schreiben wir (der besseren Lesbarkeit zuliebe)

$$zaA \to z'x \text{ statt } (z', x) \in \delta(z, a, A) \ .$$

Beispiel (fortgesetzt)

M hat die folgenden Komponenten:

- Zustandsmenge $\{z_0, z_1\}$
- Eingabealphabet $\{a, b, \$\}$
- Kelleralphabet $\{a, b, \#\}$
- Startzustand z_0
- unterstes Kellerzeichen $\#$

Intuition:

- Während der Push–Phase ist M im Zustand z_0.
- Während der Pop–Phase ist M im Zustand z_1.

Beispiel (fortgesetzt)

Die folgenden Transitionen realisieren die Push–Phase:

$$z_0a\# \rightarrow z_0a\# \quad , \quad z_0aa \rightarrow z_0aa \quad , \quad z_0ab \rightarrow z_0ab$$
$$z_0b\# \rightarrow z_0b\# \quad , \quad z_0ba \rightarrow z_0ba \quad , \quad z_0bb \rightarrow z_0bb$$

Das Umschalten von Push auf Pop wird realisiert durch

$$z_0\$\# \rightarrow z_1\# \ , \ z_0\$a \rightarrow z_1a \ , \ z_0\$b \rightarrow z_1b \ .$$

Die folgenden Transitionen realisieren die Pop–Phase:

$$z_1aa \rightarrow z_1\varepsilon \ , \ z_1bb \rightarrow z_1\varepsilon \ , \ z_1\varepsilon\# \rightarrow z_1\varepsilon$$

Auf der Eingabe $ba\$ab$ ergibt sich die Rechnung:

$$(z_0, ba\$ab, \#) \ \vdash \ (z_0, a\$ab, b\#) \ \vdash \ (z_0, \$ab, ab\#) \ \vdash$$
$$(z_1, ab, ab\#) \ \vdash \ (z_1, b, b\#) \ \vdash \ (z_1, \varepsilon, \#) \ \vdash \ (z_1, \varepsilon, \varepsilon)$$

Beispiel (fortgesetzt)

Unser PDA für die Sprache

$$L = \{a_1 a_2 \cdots a_n \$ a_n \cdots a_2 a_1 | \ a_i \in \{a, b\}\}$$

ist deterministisch:

- Jede Konfiguration hat nur eine unmittelbare Folgekonfiguration.

Beispiel (fortgesetzt)

Die Sprache

$$L' = \{a_1 a_2 \cdots a_n a_n \cdots a_2 a_1 |\ a_i \in \{a, b\}\}$$

(L abzüglich des Trennzeichens) kann von einem PDA M' erkannt werden, der ein nicht–deterministischer Verwandter von M ist:

- Ersetze die Transitionen von M beim Lesen von Trennzeichen \$ durch

$$z_0 aa \rightarrow z_1 \varepsilon\ ,\ \ z_0 bb \rightarrow z_1 \varepsilon\ ,\ \ z_0 \varepsilon\# \rightarrow z_1 \varepsilon\ \ .$$

M' hat (sofern gelesenes Zeichen mit dem obersten Kellerzeichen übereinstimmt) die Wahl in der Push–Phase zu verweilen oder (mit Hilfe der neuen Transitionen) in die Pop–Phase umzuschalten. Bei Eingaben aus L' wird die akzeptierende Rechnung genau zum richtigen Zeitpunkt diese Umschaltung vornehmen (Raten der Nahtstelle an der Mitte des Eingabewortes).

- Man kann zeigen, dass kein deterministischer PDA für L' existiert.

Von der kontextfreien Grammatik zum PDA

Satz: Jede kontextfreie Grammatik G kann in einen äqivalenten PDA M transformiert werden.

Beweis: Gegeben $G = (V, \Sigma, P, S)$, wähle die Komponenten von M wie folgt:

- Zustandsmenge $\{z\}$ (**ein** Zustand)
- Eingabealphabet Σ
- Kelleralphabet $V \cup \Sigma$
- Überführungsfunktion δ (durch Transitionen weiter unten spezifiziert)
- Startzustand z
- unterstes Kellerzeichen S

Intuition: Zu Eingabe $w \in \Sigma^*$ versucht M eine Linksableitung $S \Rightarrow_G^* w$ herzustellen. Dabei soll zu jedem Zeitpunkt im Keller eine Satzform stehen, aus welcher der noch nicht gelesene Teil der Eingabe ableitbar ist (sofern $w \in L$).

Anfangs richtig: gesamte Eingabe w noch nicht gelesen, S im Keller.

Beweis (fortgesetzt)

Überführungsfunktion δ wird entworfen wie folgt:

- Für jede Regel $A \rightarrow \alpha$ kreiere die ε–Transition $z\varepsilon A \rightarrow z\alpha$. (Intuitiv: wenn in einer Linksableitung A durch α ersetzt wird, dann ersetzt M im Keller A durch α.)
- Kreiere ferner für alle $a \in \Sigma$ die Transitionen $zaa \rightarrow z\varepsilon$.

Beobachtung am Rande: Falls G in Greibach Normalform ist, dann hat jede Regel die Form

$$A \rightarrow \alpha \text{ mit } \alpha = a\alpha' \text{ und } a \in \Sigma \ .$$

Dann können wir anstelle der ε–Transition $z\varepsilon A \rightarrow z\alpha$ auch die Transition

$$zaA \rightarrow z\alpha'$$

kreieren und der resultierende PDA (ohne ε–Transitionen!) arbeitet in Realzeit.

Die folgenden Aussagen zu einem Wort $w \in \Sigma^*$ sind induktiv beweisbar (s. Tafel):

1. **Voraussetzung:** In G ist mit einer Linksableitung die Satzform $u\beta$ erzeugbar, wobei $w = uv$ und β — sofern nicht leer — mit einer Variablen beginnt.
   Dann gilt: M kann in die Konfiguration (z, v, β) gelangen.

2. **Voraussetzung:** M kann in die Konfiguration (z, v, β) gelangen, wobei $w = uv$ und der Kellerinhalt β — sofern nicht leer — als oberstes Zeichen eine Variable aufweist.
   Dann gilt: In G ist mit einer Linksableitung die Satzform $u\beta$ erzeugbar.

Die erste Aussage (angewendet auf $u = w$ und $v = \beta = \varepsilon$) liefert (lm=leftmost)

$$w \in L(G) \Rightarrow S \Rightarrow^*_{lm} w \Rightarrow (z, w, S) \vdash (z, \varepsilon, \varepsilon) \Rightarrow w \in N(M) \ .$$

Die zweite Aussage (angewendet auf $u = w$ und $v = \beta = \varepsilon$) liefert

$$w \in N(M) \Rightarrow (z, w, S) \vdash (z, \varepsilon, \varepsilon) \Rightarrow S \Rightarrow^*_{lm} w \Rightarrow w \in L(G) \ .$$

Vom PDA zur kontextfreien Grammatik

Satz: Jeder PDA M kann in eine äquivalente kontextfreie Grammatik G transformiert werden.

Beweisstruktur:

- Bringe M zunächst in eine „Normalform" (ohne die zugehörige Sprache abzuändern).
- Entwerfe eine kontextfreie Grammatik G, deren Linksableitungen imstande sind, M zu „simulieren".
- Weise die **Korrektheit der Simulation** mit Hilfe einer **Invarianzbedingung** nach.
- Folgere, dass $L(G) = N(M)$.

„Normalform“ des PDA

Transitionen der Form

$$zaA \rightarrow z'B_1 \cdots B_k \text{ mit } k > 2$$

könnten mit Hilfe neuer Zustände $z_1, \ldots, z_{k-2}$ simuliert werden durch die Transitionen

$$zaA \rightarrow z_1 B_{k-1} B_k \ , \ z_1 \varepsilon B_{k-1} \rightarrow z_2 B_{k-2} B_{k-1} \ , \ \ldots \ , \ z_{k-2} \varepsilon B_2 \rightarrow z' B_1 B_2 \ .$$

Daher können wir **annehmen**, dass der gegebene PDA M nur Transitionen der Form

$$z \rightarrow z'B_1 \cdots B_k \text{ mit } k \in \{0, 1, 2\}$$

verwendet.

Entwurf einer Grammatik (Tripelkonstruktion)

Gegeben der (normalisierte) PDA $M = (Z, \Sigma, \Gamma, \delta, z_0, \#)$, wähle die Komponenten von G wie folgt:

- Variablenmenge $V = \{S\} \cup (Z \times \Gamma \times Z)$
- Terminalalphabet Σ
- Produktionensystem P (durch Regeln weiter unten spezifiziert)
- Startsymbol S

Wegen der Variablen der Form $(z, A, z') \in Z \times \Gamma \times Z$ spricht man von der „Tripelkonstruktion“.

Intuition: Interpretiere das Tripel (z, A, z') als „Wette“ auf folgendes Ereignis: nachdem M, gestartet im Zustand z mit Kellerinhalt A, den Keller geleert hat, befindet er sich im Zustand z'.

- Nimm für jedes $z \in Z$ die Regel $S \rightarrow (z_0, \#, z)$ in P auf.
- Für jede Transition von M der Form $zaA \rightarrow z'\varepsilon$ nimm die Regel $(z, A, z') \rightarrow a$ in P auf.
- Für jede Transition von M der Form $zaA \rightarrow z_1B$ und jeden Zustand $z' \in Z$, nimm die Regel der Form $(z, A, z') \rightarrow a(z_1, B, z')$ in P auf.
- Für jede Transition von M der Form $zaA \rightarrow z_1BC$ und jede Wahl von $z', z_2 \in Z$, nimm die Regel $(z, A, z') \rightarrow a(z_1, B, z_2)(z_2, C, z')$ in P auf.

Die folgende Äquivalenz ist induktiv beweisbar (s. Tafel):

$$(z, A, z') \Rightarrow_{lm}^{*} x \textbf{ gdw } (z, x, A) \vdash^{*} (z', \varepsilon, \varepsilon)$$

$L(G) = N(M)$ ergibt sich jetzt wie folgt:

$$\begin{aligned} w \in N(M) \quad &\Leftrightarrow \quad \exists z \in Z : (z_0, w, \#) \vdash^{*} (z, \varepsilon, \varepsilon) \\ &\Leftrightarrow \quad \exists z \in Z : S \Rightarrow (z_0, \#, z) \Rightarrow_{lm}^{*} w \\ &\Leftrightarrow \quad w \in L(G) \end{aligned}$$

Folgerungen

1. Die Klasse der von PDAs erkennbaren Sprachen stimmt mit der Klasse der kontextfreien Sprachen überein.

2. Jede kontextfreie Sprache kann von einem PDA erkannt werden, der nur einen einzigen Zustand besitzt und ohne ε–Transitionen auskommt (und daher in Realzeit arbeitet).