TECHNISCHE UNIVERSITÄT DRESDEN

FORMALE SYSTEME

17. Vorlesung: Deterministische Sprachen / Entscheidungsprobleme

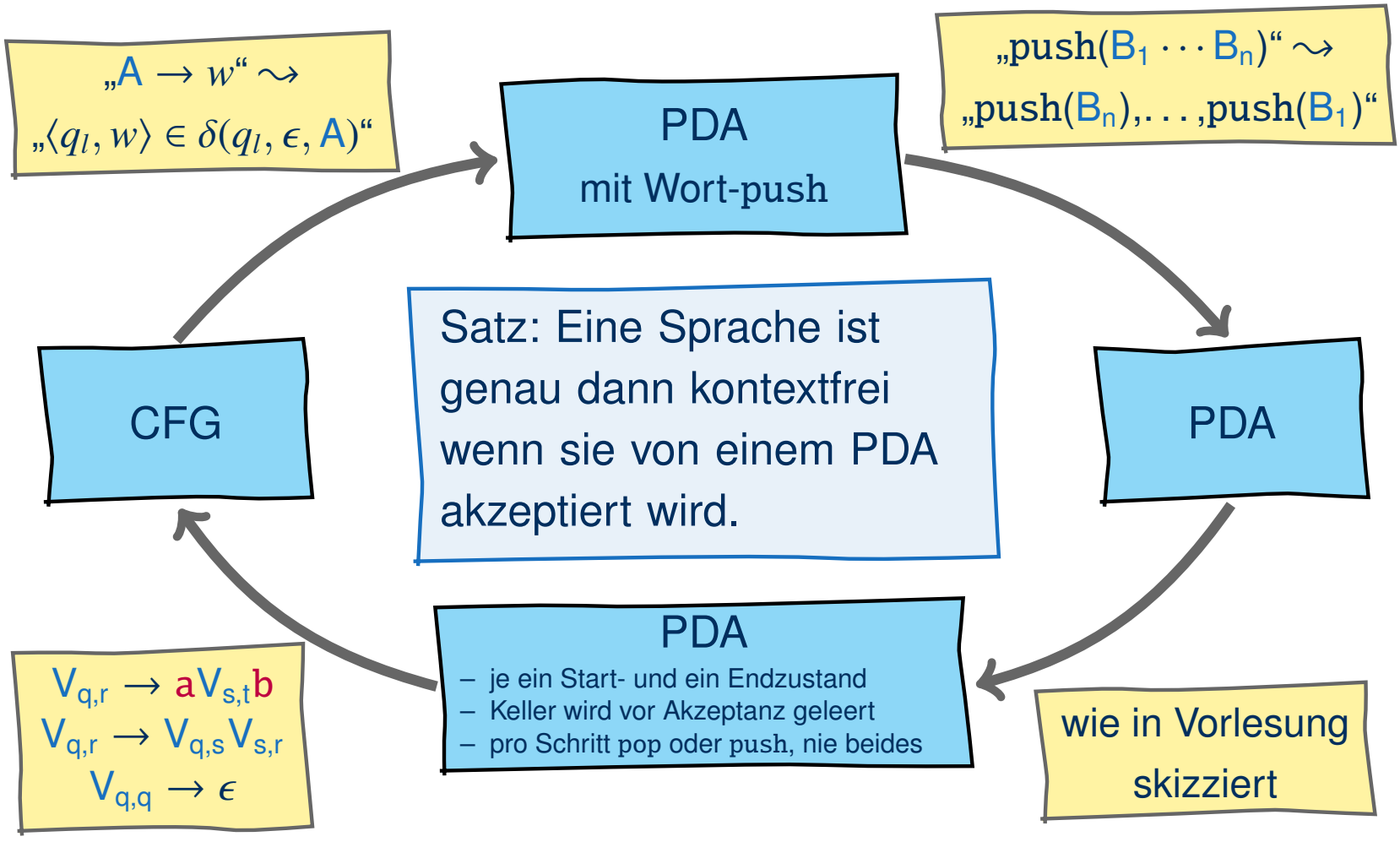
Markus Krötzsch

Lehrstuhl Wissensbasierte Systeme

TU Dresden, 12. Dezember 2016

Rückblick

PDA ≙ CFG ≙ Typ 2


„$A \to w$“ ⇝
„$\langle q_l, w\rangle \in \delta(q_l, \epsilon, A)$“
PDA
mit Wort-push
„push($B_1 \cdots B_n$)“ ⇝
„push(B_n),...,push(B_1)“
CFG
Satz: Eine Sprache ist genau dann kontextfrei wenn sie von einem PDA akzeptiert wird.
PDA
$V_{q,r} \to aV_{s,t}b$
$V_{q,r} \to V_{q,s}V_{s,r}$
$V_{q,q} \to \epsilon$
PDA
– je ein Start- und ein Endzustand
– Keller wird vor Akzeptanz geleert
– pro Schritt pop oder push, nie beides
wie in Vorlesung skizziert

Deterministische Kellerautomaten

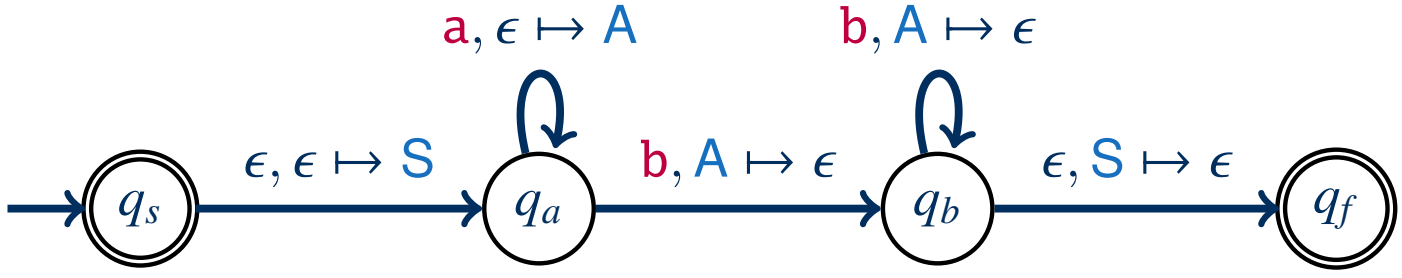
Ein deterministischer Kellerautomat (international: „DPDA“) $\mathcal{M}$ ist ein Tupel $\mathcal{M} = \langle Q, \Sigma, \Gamma, \delta, q_0, F \rangle$ bestehend aus Zustandsmenge Q, Eingabealphabet Σ, Kelleralphabet Γ, Startzustand $q_0 \in Q$, Endzustände $F \subseteq Q$ und partieller Übergangsfunktion

$$Q \times \Sigma_\epsilon \times \Gamma_\epsilon \to Q \times \Gamma_\epsilon,$$

so dass für alle $q \in Q$, $\mathrm{a} \in \Sigma$ und $\mathrm{A} \in \Gamma$ jeweils nur eines der folgenden definiert ist:

$$\delta(q, \mathrm{a}, \mathrm{A}) \qquad \delta(q, \mathrm{a}, \epsilon) \qquad \delta(q, \epsilon, \mathrm{A}) \qquad \delta(q, \epsilon, \epsilon)$$

Beispiel: $\mathrm{a}^i\mathrm{b}^i$



Deterministisch vs. nichtdeterminitisch

Typ-2-Sprachen

- Erkannt durch PDAs
- Nicht unter Komplement abgeschlossen
- Echte Obermenge der det. Typ-2-Sprachen, z.B. $\{a^i b^j c^k \mid i \neq j \text{ oder } j \neq k\}$
- Wortproblem in $O(|w|^3)$ (CYK-Algorithmus)
- Generiert durch CFGs

Deterministische Typ-2-Sprachen

- Erkannt durch DPDAs
- Unter Komplement abgeschlossen
- Echte Obermenge der regulären Sprachen, z.B. $\{a^i b^i \mid i \geq 0\}$
- Wortproblem in $O(|w|)$ (DPDA-Abarbeitung)
- Generiert durch deterministische CFGs (kein Vorlesungsstoff)

Deterministische Typ-2-Sprachen

Rückblick: Ableitungsbäume

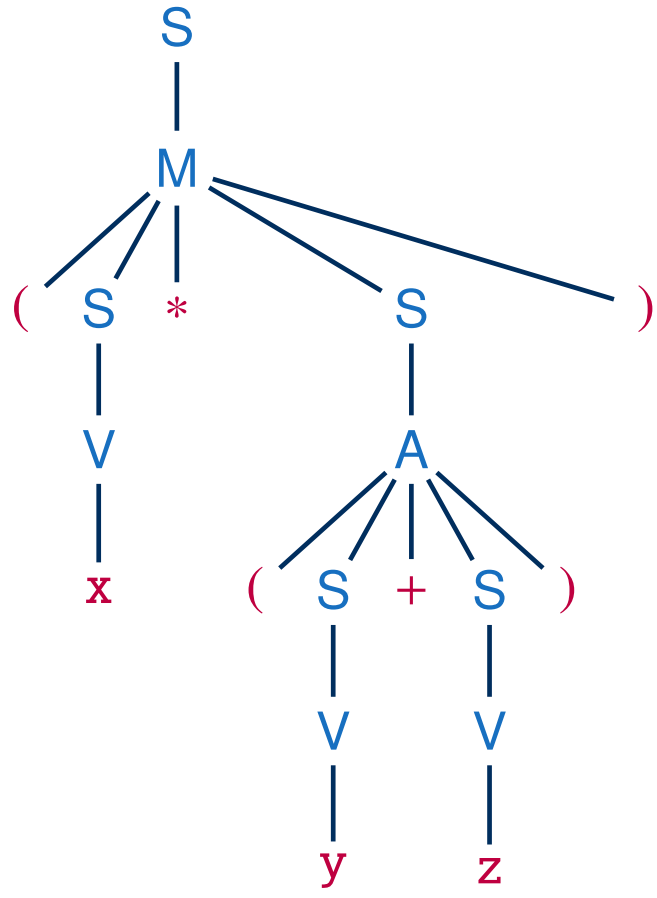
Rückblick:
Die Interpretation von Wörtern kontextfreier Sprachen basiert zumeist auf dem Syntaxbaum.

Beispiel:

$S \to A \mid M \mid V \quad A \to (S{+}S)$

$M \to (S{*}S) \quad V \to \mathtt{x} \mid \mathtt{y} \mid \mathtt{z}$

Wort: $(\mathtt{x}{*}(\mathtt{y}{+}\mathtt{z}))$



Mehrdeutige Grammatiken

Wir wissen: Ein Ableitungsbaum kann zumeist durch verschiedene Ableitungen erzeugt werden

Mehrdeutige Grammatiken

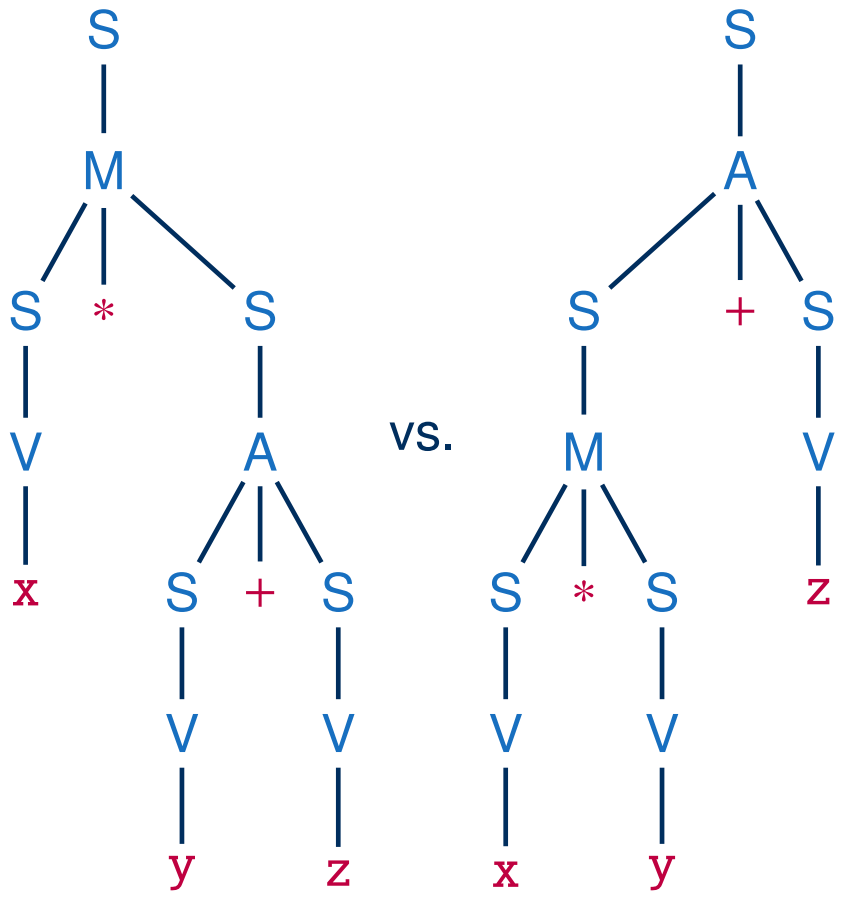
Wir wissen: Ein Ableitungsbaum kann zumeist durch verschiedene Ableitungen erzeugt werden

Es gilt aber auch: Ein Wort kann mehrere verschiedene Ableitungsbäume haben

Mehrdeutige Grammatiken

Wir wissen: Ein Ableitungsbaum kann zumeist durch verschiedene Ableitungen erzeugt werden

Es gilt aber auch: Ein Wort kann mehrere verschiedene Ableitungsbäume haben



Beispiel:

$S \to A \mid M \mid V \quad A \to S{+}S$

$M \to S{*}S \quad V \to \mathtt{x} \mid \mathtt{y} \mid \mathtt{z}$

Mehrdeutige Grammatiken (2)

Eine Grammatik G ist mehrdeutig wenn es ein Wort $w \in \mathbf{L}(G)$ gibt, das mehrere Syntaxbäume zulässt (äquivalent: wenn es für w mehrere unterschiedliche Linksableitungen gibt).

Mehrdeutige Grammatiken (2)

Eine Grammatik G ist mehrdeutig wenn es ein Wort $w \in \mathbf{L}(G)$ gibt, das mehrere Syntaxbäume zulässt (äquivalent: wenn es für w mehrere unterschiedliche Linksableitungen gibt).

Mehrdeutige Grammatiken in der Praxis:

- unerwünscht in technischen Sprachdefinitionen (Programmiersprachen, Datenformate)
- relevant in der Sprachverarbeitung (um die Mehrdeutigkeit natürlicher Sprache abzubilden)

Mehrdeutige Grammatiken (2)

Eine Grammatik G ist mehrdeutig wenn es ein Wort $w \in \mathbf{L}(G)$ gibt, das mehrere Syntaxbäume zulässt (äquivalent: wenn es für w mehrere unterschiedliche Linksableitungen gibt).

Mehrdeutige Grammatiken in der Praxis:

- unerwünscht in technischen Sprachdefinitionen (Programmiersprachen, Datenformate)
- relevant in der Sprachverarbeitung (um die Mehrdeutigkeit natürlicher Sprache abzubilden)

Für viele mehrdeutige Grammatiken kann man eindeutige Grammatiken finden, welche die gleiche Sprache generieren, z.B.:

S → A | M A → A+A | M M → M∗M | V V → x | y | z

Mehrdeutige Grammatiken (2)

Eine Grammatik G ist mehrdeutig wenn es ein Wort $w \in \mathbf{L}(G)$ gibt, das mehrere Syntaxbäume zulässt (äquivalent: wenn es für w mehrere unterschiedliche Linksableitungen gibt).

Mehrdeutige Grammatiken in der Praxis:

- unerwünscht in technischen Sprachdefinitionen (Programmiersprachen, Datenformate)
- relevant in der Sprachverarbeitung (um die Mehrdeutigkeit natürlicher Sprache abzubilden)

Für viele mehrdeutige Grammatiken kann man eindeutige Grammatiken finden, welche die gleiche Sprache generieren, z.B.:

$$S \to A \mid M \qquad A \to A{+}A \mid M \qquad M \to M{*}M \mid V \qquad V \to \mathtt{x} \mid \mathtt{y} \mid \mathtt{z}$$

Aber: es gibt inhärent mehrdeutige Sprachen, die nur mehrdeutige Grammatiken haben, z.B.: $\{\mathtt{a}^i\mathtt{b}^i\mathtt{c}^k \mid i,k \geq 0\} \cup \{\mathtt{a}^i\mathtt{b}^k\mathtt{c}^k \mid i,k \geq 0\}$

Mehrdeutig vs. Deterministisch

Fakten:

- Deterministische Sprachen haben immer auch eindeutige Grammatiken (unter anderen)
- Aber: eindeutige Grammatiken können nicht-deterministische Typ-2-Sprachen beschreiben

$$\text{deterministisch} \subsetneq \text{eindeutig} \subsetneq \text{Typ 2}$$

Mehrdeutig vs. Deterministisch

Fakten:

- Deterministische Sprachen haben immer auch eindeutige Grammatiken (unter anderen)
- Aber: eindeutige Grammatiken können nicht-deterministische Typ-2-Sprachen beschreiben

$$\text{deterministisch} \subsetneq \text{eindeutig} \subsetneq \text{Typ 2}$$

Unterschiedliche Motivationen:

- eindeutige Sprachdefinitionen: eindeutige Syntaxbäume, eindeutige Interpretation
- deterministische Sprachen: effizienteres Parsing möglich

viele Programmiersprachen verwenden deterministische (und eindeutige) Grammatiken

Deterministische CFGs

Man kann die Klasse der deterministischen Typ-2-Sprachen auch durch Grammatiken beschreiben:

- Dies führt zu deterministischen kontextfreien Grammatiken (DCFGs)
- Die eigentliche Definition ist relativ technisch
- DPDAs erkennen im Prinzip dieselben Sprachen wie DCFGs

  sofern man sich auf Sprachen beschränkt, bei denen jedes Wort als letztes Zeichen ein spezielles Schlusssymbol verwendet, um das Ende zu markieren!

Deterministische CFGs

Man kann die Klasse der deterministischen Typ-2-Sprachen auch durch Grammatiken beschreiben:

- Dies führt zu deterministischen kontextfreien Grammatiken (DCFGs)
- Die eigentliche Definition ist relativ technisch
- DPDAs erkennen im Prinzip dieselben Sprachen wie DCFGs

  sofern man sich auf Sprachen beschränkt, bei denen jedes Wort als letztes Zeichen ein spezielles Schlusssymbol verwendet, um das Ende zu markieren!

Gute Nachricht: Man kann effektiv entscheiden, ob eine gegebene CFG deterministisch ist.

Deterministische CFGs

Man kann die Klasse der deterministischen Typ-2-Sprachen auch durch Grammatiken beschreiben:

- Dies führt zu deterministischen kontextfreien Grammatiken (DCFGs)
- Die eigentliche Definition ist relativ technisch
- DPDAs erkennen im Prinzip dieselben Sprachen wie DCFGs

  sofern man sich auf Sprachen beschränkt, bei denen jedes Wort als letztes Zeichen ein spezielles Schlusssymbol verwendet, um das Ende zu markieren!

Gute Nachricht: Man kann effektiv entscheiden, ob eine gegebene CFG deterministisch ist.

Schlechte Nachricht: Praktische Grammatiken erfüllen die strengen Bedingungen oft nicht, auch wenn sie eine deterministisch kontextfreie Sprache beschreiben

Nach vorne blicken

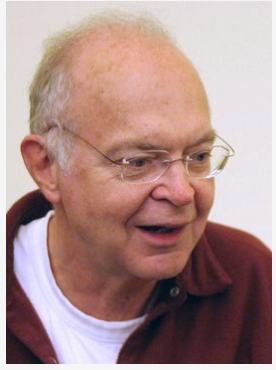
Deterministische Typ-2-Sprachen sind praktisch sehr relevant:

- Programmiersprachen sind meist im Kern deterministisch
- Man kann sie sehr effizient parsen (kein CYK)

Nach vorne blicken

Deterministische Typ-2-Sprachen sind praktisch sehr relevant:

- Programmiersprachen sind meist im Kern deterministisch
- Man kann sie sehr effizient parsen (kein CYK)



Don Knuth, 2005
CC-By-SA 2.5
(c) J. Appelbaum

In der Praxis hilft eine Verallgemeinerung von DCFGs: Grammatiken mit endlicher Vorschau (Lookahead).

Idee:

- Wort wird von links nach rechts gelesen
- Grammatikregeln werden rückwärts angewendet, um Teile des gelesenen Worts zu reduzieren
- Die Wahl der Grammatikregel hängt nur vom schon gelesenen Wort und von bis zu k weiteren Symbolen ab (Vorschau)

Grammatiken, die das erlauben, sind vom Typ **LR**(k), wobei **LR**(0) (keine Vorschau) DCFGs sind.

Abschlusseigenschaften (1)

Wir wissen bereits, dass deterministische Typ-2-Sprachen unter Komplement abgeschlossen sind.

Abschlusseigenschaften (1)

Wir wissen bereits, dass deterministische Typ-2-Sprachen unter Komplement abgeschlossen sind.

Für Schnitte gilt das allerdings nicht:

Satz: Deterministische Typ-2-Sprachen sind nicht unter Schnitten abgeschlossen.

Abschlusseigenschaften (1)

Wir wissen bereits, dass deterministische Typ-2-Sprachen unter Komplement abgeschlossen sind.

Für Schnitte gilt das allerdings nicht:

> Satz: Deterministische Typ-2-Sprachen sind nicht unter Schnitten abgeschlossen.

Beweis: Der Beweis für Typ-2-Sprachen funktioniert auch hier. Die Sprachen

$$\mathbf{L_1} = \{a^i b^i c^k \mid i \geq 0, k \geq 0\}$$
$$\mathbf{L_2} = \{a^i b^k c^k \mid i \geq 0, k \geq 0\}.$$

sind deterministisch kontextfrei (Übung: Geben Sie entsprechende DPDAs an). Ihr Schnitt $\mathbf{L_1} \cap \mathbf{L_2} = \{a^i b^i c^i \mid i \geq 0\}$ ist dagegen nicht einmal kontextfrei. □

Abschlusseigenschaften (2)

Der Nichtabschluss unter Schnitten hat weitere Konsequenzen:

Satz: Deterministische Typ-2-Sprachen sind nicht unter Vereinigung abgeschlossen.

Abschlusseigenschaften (2)

Der Nichtabschluss unter Schnitten hat weitere Konsequenzen:

Satz: Deterministische Typ-2-Sprachen sind nicht unter Vereinigung abgeschlossen.

Beweis: Angenommen sie wären unter Vereinigung abgeschlossen, dann wären sie auch unter Schnitten abgeschlossen, da sie bereits unter Komplement abgeschlossen sind (De Morgan). Widerspruch. □

Abschlusseigenschaften (3)

Bei anderen Operationen sieht es nicht besser aus:

Satz: Deterministische Typ-2-Sprachen sind nicht unter Konkatenation oder Kleene-Stern abgeschlossen.

Abschlusseigenschaften (3)

Bei anderen Operationen sieht es nicht besser aus:

Satz: Deterministische Typ-2-Sprachen sind nicht unter Konkatenation oder Kleene-Stern abgeschlossen.

Beweisidee: Vereinigungen kann man deterministisch machen, indem man einer der Alternativen ein Markierungszeichen $\mathtt{X}$ vorschaltet, das ansonsten nie am Anfang des Wortes auftauchen darf. Falls man die Sprache dann aber an die (deterministische Sprache) $\mathtt{X}^*$ anhängt, ist die Markierung nicht mehr als Entscheidungshilfe nutzbar. Die Idee beim Stern ist ähnlich. □

Zusammenfassung: Deterministische Typ-2-Sprachen sind abgeschlossen unter Komplement, aber nicht unter Vereinigung, Schnitt, Konkatenation oder Stern.

Entscheidungsprobleme auf kontextfreien Sprachen

Rückblick Entscheidungsprobleme

Für reguläre Sprachen haben wir eine Reihe von Problemstellungen kennengelernt:

- Leerheitsproblem: Ist die beschriebene Sprache $\emptyset$?
- Inklusionsproblem: Ist eine beschriebene Sprache Teilmenge einer anderen?
- Äquivalenzproblem: Wird durch zwei Beschreibungen die selbe Sprache gegeben?
- Endlichkeitsproblem: Ist die beschriebene Sprache endlich?
- Universalitätsproblem: Ist die beschriebene Sprache Σ^*?

Dabei könnten Sprachen durch verschiedene Beschreibungen gegeben sein (Automaten, Grammatiken, . . .)

Zudem gibt es freilich das Wortproblem
(für [D]CFGs bereits besprochen)

Meistens unentscheidbar

Viele interessante Fragen sind leider im Allgemeinen nicht durch Algorithmen lösbar:

> Satz: Inklusion, Äquivalenz und Universalität von CFGs ist unentscheidbar.

(ohne Beweis, da wir noch gar nicht über Entscheidbarkeit gesprochen haben . . .)

Meistens unentscheidbar

Viele interessante Fragen sind leider im Allgemeinen nicht durch Algorithmen lösbar:

> Satz: Inklusion, Äquivalenz und Universalität von CFGs ist unentscheidbar.

(ohne Beweis, da wir noch gar nicht über Entscheidbarkeit gesprochen haben . . .)

Einiges ist aber doch machbar:

> Satz: Leerheit und Endlichkeit einer CFG sind entscheidbar.

Diese Ergebnisse gelten ebenso, wenn PDAs statt CFGs gegeben sind, da wir diese ja in CFGs umwandeln können.

Leerheit entscheiden

Satz: Die Leerheit einer CFG ist entscheidbar.

Beweis: Man markiert Variablen mit folgender Prozedur:

- Markiere alle Variablen, welche direkt in ein Wort aus Terminalzeichen umgeschrieben werden können
- Markiere rekursiv alle Variablen, welche in ein Wort aus Terminalzeichen und markierten Variablen umgeschrieben werden können

Die Sprache ist genau dann nicht leer wenn bei diesem Verfahren das Startsymbol markiert wird. □

Endlichkeit entscheiden

Satz: Endlichkeit der Sprache $\mathbf{L}(G)$ einer CFG G ist entscheidbar.

Endlichkeit entscheiden

Satz: Endlichkeit der Sprache $\mathbf{L}(G)$ einer CFG G ist entscheidbar.

Beweis: Sei n die Zahl aus dem Pumpinglemma (also $2^{|V|}$).

Endlichkeit entscheiden

Satz: Endlichkeit der Sprache $\mathbf{L}(G)$ einer CFG G ist entscheidbar.

Beweis: Sei n die Zahl aus dem Pumpinglemma (also $2^{|V|}$).

- Wenn es ein Wort $z \in \mathbf{L}(G)$ mit $n \leq |z| < 2n$ gibt, dann ist $\mathbf{L}(G)$ unendlich (da man das Pumpinglemma auf z anwenden kann).

Endlichkeit entscheiden

Satz: Endlichkeit der Sprache $\mathbf{L}(G)$ einer CFG G ist entscheidbar.

Beweis: Sei n die Zahl aus dem Pumpinglemma (also $2^{|V|}$).

- Wenn es ein Wort $z \in \mathbf{L}(G)$ mit $n \leq |z| < 2n$ gibt, dann ist $\mathbf{L}(G)$ unendlich (da man das Pumpinglemma auf z anwenden kann).
- Wenn $\mathbf{L}(G)$ unendlich ist, dann gibt es ein Wort $z \in \mathbf{L}(G)$ mit $n \leq |z| < 2n$

Endlichkeit entscheiden

Satz: Endlichkeit der Sprache $\mathbf{L}(G)$ einer CFG G ist entscheidbar.

Beweis: Sei n die Zahl aus dem Pumpinglemma (also $2^{|V|}$).

- Wenn es ein Wort $z \in \mathbf{L}(G)$ mit $n \leq |z| < 2n$ gibt, dann ist $\mathbf{L}(G)$ unendlich (da man das Pumpinglemma auf z anwenden kann).
- Wenn $\mathbf{L}(G)$ unendlich ist, dann gibt es ein Wort $z \in \mathbf{L}(G)$ mit $n \leq |z| < 2n$ (Beweis: Es muss Wörter mit mehr als n Zeichen geben.

Endlichkeit entscheiden

Satz: Endlichkeit der Sprache $\mathbf{L}(G)$ einer CFG G ist entscheidbar.

Beweis: Sei n die Zahl aus dem Pumpinglemma (also $2^{|V|}$).

- Wenn es ein Wort $z \in \mathbf{L}(G)$ mit $n \leq |z| < 2n$ gibt, dann ist $\mathbf{L}(G)$ unendlich (da man das Pumpinglemma auf z anwenden kann).
- Wenn $\mathbf{L}(G)$ unendlich ist, dann gibt es ein Wort $z \in \mathbf{L}(G)$ mit $n \leq |z| < 2n$ (Beweis: Es muss Wörter mit mehr als n Zeichen geben. Sei z ein kürzestes Wort dieser Art.

Endlichkeit entscheiden

Satz: Endlichkeit der Sprache $\mathbf{L}(G)$ einer CFG G ist entscheidbar.

Beweis: Sei n die Zahl aus dem Pumpinglemma (also $2^{|V|}$).

- Wenn es ein Wort $z \in \mathbf{L}(G)$ mit $n \leq |z| < 2n$ gibt, dann ist $\mathbf{L}(G)$ unendlich (da man das Pumpinglemma auf z anwenden kann).
- Wenn $\mathbf{L}(G)$ unendlich ist, dann gibt es ein Wort $z \in \mathbf{L}(G)$ mit $n \leq |z| < 2n$ (Beweis: Es muss Wörter mit mehr als n Zeichen geben. Sei z ein kürzestes Wort dieser Art. Laut Pumpinglemma ist $z = uvwxy$ mit $|vx| < n$ und $uv^0wx^0y = uwy \in \mathbf{L}(G)$.

Endlichkeit entscheiden

Satz: Endlichkeit der Sprache $\mathbf{L}(G)$ einer CFG G ist entscheidbar.

Beweis: Sei n die Zahl aus dem Pumpinglemma (also $2^{|V|}$).

- Wenn es ein Wort $z \in \mathbf{L}(G)$ mit $n \leq |z| < 2n$ gibt, dann ist $\mathbf{L}(G)$ unendlich (da man das Pumpinglemma auf z anwenden kann).
- Wenn $\mathbf{L}(G)$ unendlich ist, dann gibt es ein Wort $z \in \mathbf{L}(G)$ mit $n \leq |z| < 2n$ (Beweis: Es muss Wörter mit mehr als n Zeichen geben. Sei z ein kürzestes Wort dieser Art. Laut Pumpinglemma ist $z = uvwxy$ mit $|vx| < n$ und $uv^0wx^0y = uwy \in \mathbf{L}(G)$. Da uwy kürzer ist als z muss gelten $|uwy| < n$.

Endlichkeit entscheiden

Satz: Endlichkeit der Sprache $\mathbf{L}(G)$ einer CFG G ist entscheidbar.

Beweis: Sei n die Zahl aus dem Pumpinglemma (also $2^{|V|}$).

- Wenn es ein Wort $z \in \mathbf{L}(G)$ mit $n \leq |z| < 2n$ gibt, dann ist $\mathbf{L}(G)$ unendlich (da man das Pumpinglemma auf z anwenden kann).
- Wenn $\mathbf{L}(G)$ unendlich ist, dann gibt es ein Wort $z \in \mathbf{L}(G)$ mit $n \leq |z| < 2n$ (Beweis: Es muss Wörter mit mehr als n Zeichen geben. Sei z ein kürzestes Wort dieser Art. Laut Pumpinglemma ist $z = uvwxy$ mit $|vx| < n$ und $uv^0wx^0y = uwy \in \mathbf{L}(G)$. Da uwy kürzer ist als z muss gelten $|uwy| < n$. Daraus folgt $|z| = |uwy| + |vx| < 2n$.)

Endlichkeit entscheiden

Satz: Endlichkeit der Sprache $\mathbf{L}(G)$ einer CFG G ist entscheidbar.

Beweis: Sei n die Zahl aus dem Pumpinglemma (also $2^{|V|}$).

- Wenn es ein Wort $z \in \mathbf{L}(G)$ mit $n \leq |z| < 2n$ gibt, dann ist $\mathbf{L}(G)$ unendlich (da man das Pumpinglemma auf z anwenden kann).
- Wenn $\mathbf{L}(G)$ unendlich ist, dann gibt es ein Wort $z \in \mathbf{L}(G)$ mit $n \leq |z| < 2n$ (Beweis: Es muss Wörter mit mehr als n Zeichen geben. Sei z ein kürzestes Wort dieser Art. Laut Pumpinglemma ist $z = uvwxy$ mit $|vx| < n$ und $uv^0wx^0y = uwy \in \mathbf{L}(G)$. Da uwy kürzer ist als z muss gelten $|uwy| < n$. Daraus folgt $|z| = |uwy| + |vx| < 2n$.)

Das heißt, wir müssen nur testen, ob es so ein Wort $z \in \mathbf{L}(G)$ mit $n \leq |z| < 2n$ gibt. Das kann man (Brute Force) für alle Wörter dieser Länge tun (da das Wortproblem lösbar ist). □

(Es gibt effizientere Verfahren, aber dieses ist das einfachste für den Beweis.)

Alles unentscheidbar

Viele weitere interessante Fragen sind leider ebenfalls unentscheidbar:

- Regularität: Ist die durch eine CFG gegebene Sprache regulär?
- Mehrdeutigkeit: Ist eine gegebene CFG mehrdeutig oder nicht?
- Determinisierbarkeit: Ist die durch eine CFG gegebene Sprache deterministisch?[1]
- Schnittproblem: Haben zwei gegebene Sprachen gemeinsame Wörter?

[1] Aber, wie zuvor angemerkt: man kann entscheiden, ob eine gegebene CFG bereits deterministisch ist (wenn sie es nicht ist, dann bedeutet das aber nicht, dass es keine äquivalente DCFG geben könnte).

Entscheidungsprobleme für DPDAs

Die Situation ist etwas besser bei DPDAs:

Leerheit	entscheidbar (wie bei CFGs)
Endlichkeit	entscheidbar (wie bei CFGs)

Entscheidungsprobleme für DPDAs

Die Situation ist etwas besser bei DPDAs:

Leerheit	entscheidbar (wie bei CFGs)
Endlichkeit	entscheidbar (wie bei CFGs)
Universalität	

Entscheidungsprobleme für DPDAs

Die Situation ist etwas besser bei DPDAs:

Leerheit	entscheidbar (wie bei CFGs)
Endlichkeit	entscheidbar (wie bei CFGs)
Universalität	entscheidbar (entspricht Leerheit des Komplements)

Entscheidungsprobleme für DPDAs

Die Situation ist etwas besser bei DPDAs:

Leerheit	entscheidbar (wie bei CFGs)
Endlichkeit	entscheidbar (wie bei CFGs)
Universalität	entscheidbar (entspricht Leerheit des Komplements)
Regularität	entscheidbar (Stearns: A Regularity Test for Pushdown Machines, 1967)

Entscheidungsprobleme für DPDAs

Die Situation ist etwas besser bei DPDAs:

Leerheit	entscheidbar (wie bei CFGs)
Endlichkeit	entscheidbar (wie bei CFGs)
Universalität	entscheidbar (entspricht Leerheit des Komplements)
Regularität	entscheidbar (Stearns: A Regularity Test for Pushdown Machines, 1967)
Inklusion	unentscheidbar (Ginsburg & Greibach: Deterministic context-free languages, 1966)

Entscheidungsprobleme für DPDAs

Die Situation ist etwas besser bei DPDAs:

Leerheit	entscheidbar (wie bei CFGs)
Endlichkeit	entscheidbar (wie bei CFGs)
Universalität	entscheidbar (entspricht Leerheit des Komplements)
Regularität	entscheidbar (Stearns: A Regularity Test for Pushdown Machines, 1967)
Inklusion	unentscheidbar (Ginsburg & Greibach: Deterministic context-free languages, 1966)
Schnitt	

Entscheidungsprobleme für DPDAs

Die Situation ist etwas besser bei DPDAs:

Leerheit	entscheidbar (wie bei CFGs)
Endlichkeit	entscheidbar (wie bei CFGs)
Universalität	entscheidbar (entspricht Leerheit des Komplements)
Regularität	entscheidbar (Stearns: A Regularity Test for Pushdown Machines, 1967)
Inklusion	unentscheidbar (Ginsburg & Greibach: Deterministic context-free languages, 1966)
Schnitt	unentscheidbar (wie Inklusion, da wir Komplemente haben)

Entscheidungsprobleme für DPDAs

Die Situation ist etwas besser bei DPDAs:

Leerheit	entscheidbar (wie bei CFGs)
Endlichkeit	entscheidbar (wie bei CFGs)
Universalität	entscheidbar (entspricht Leerheit des Komplements)
Regularität	entscheidbar (Stearns: A Regularity Test for Pushdown Machines, 1967)
Inklusion	unentscheidbar (Ginsburg & Greibach: Deterministic context-free languages, 1966)
Schnitt	unentscheidbar (wie Inklusion, da wir Komplemente haben)
Äquivalenz	

Entscheidungsprobleme für DPDAs

Die Situation ist etwas besser bei DPDAs:

Leerheit	entscheidbar (wie bei CFGs)
Endlichkeit	entscheidbar (wie bei CFGs)
Universalität	entscheidbar (entspricht Leerheit des Komplements)
Regularität	entscheidbar (Stearns: A Regularity Test for Pushdown Machines, 1967)
Inklusion	unentscheidbar (Ginsburg & Greibach: Deterministic context-free languages, 1966)
Schnitt	unentscheidbar (wie Inklusion, da wir Komplemente haben)
Äquivalenz	entscheidbar! (Sénizergues: L(A)=L(B)? decidability results from complete formal systems, 2001; komplexes Verfahren ohne Komplexitätsschranken; Ergebnis bekannt seit 1997)

(Mehrdeutigkeit und Determinisierbarkeit sind bei DPDAs trivial)

Übersicht

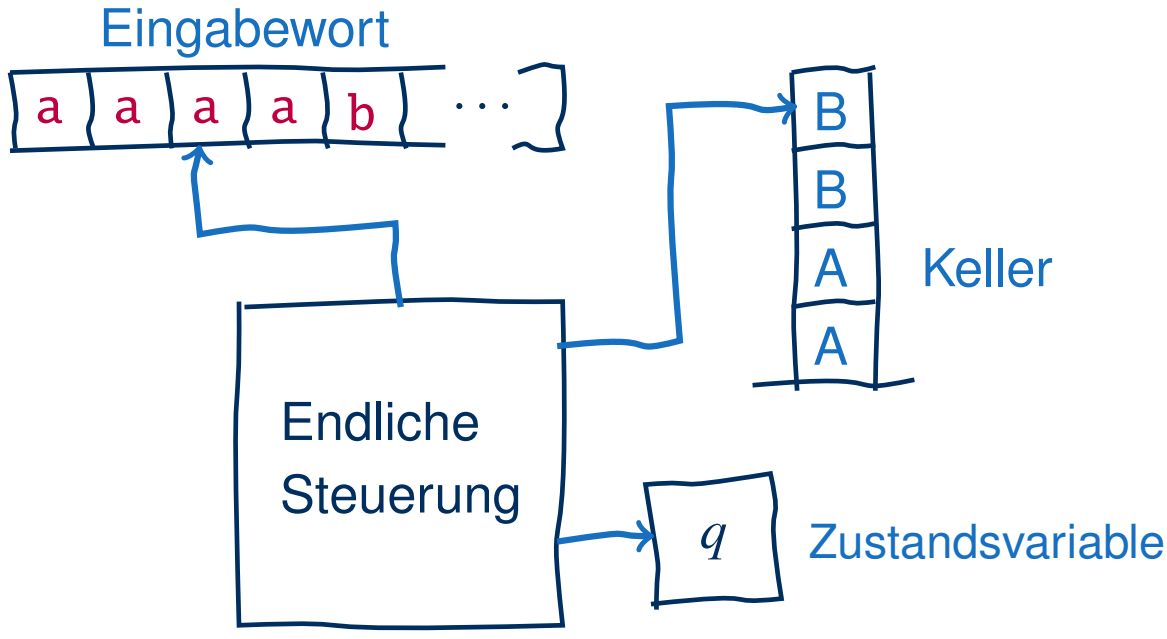
	CFG	**DPDA**
Wortproblem	in $O(\|w\|^3)$	in $O(\|w\|)$
Leerheit	entscheidbar	entscheidbar
Endlichkeit	entscheidbar	entscheidbar
Universalität	unentscheidbar	entscheidbar
Inklusion	unentscheidbar	unentscheidbar
Schnitt	unentscheidbar	unentscheidbar
Äquivalenz	unentscheidbar	entscheidbar
Regularität	unentscheidbar	entscheidbar
Mehrdeutigkeit	unentscheidbar	trivial
Determinisierbarkeit	unentscheidbar	trivial

Rechnen mit Typ 1 und Typ 0

Wortprobleme jenseits von Typ 2

Wir haben gesehen:

- endliche Automaten erkennen Typ-3-Sprachen
- endliche Automaten + Kellerspeicher erkennen Typ-2-Sprachen



Für Typ 1 und Typ 0 benötigen wir mehr als das – aber was?

Berechnungsmodelle nach Kellerautomaten?

Beobachtung: Auch jenseits von Typ 2 kann man Wortprobleme algorithmisch lösen.

Beispiel: Die Sprache $\{a^i b^i c^i \mid i \geq 0\}$ ist nicht kontextfrei, wird also von keinem PDA erkannt. Dennoch wäre es nicht sehr schwer, ein Programm in einer beliebigen Programmiersprache zu schreiben, welches feststellt, ob eine Eingabe diese Form hat.

Berechnungsmodelle nach Kellerautomaten?

Beobachtung: Auch jenseits von Typ 2 kann man Wortprobleme algorithmisch lösen.

Beispiel: Die Sprache $\{a^i b^i c^i \mid i \geq 0\}$ ist nicht kontextfrei, wird also von keinem PDA erkannt. Dennoch wäre es nicht sehr schwer, ein Programm in einer beliebigen Programmiersprache zu schreiben, welches feststellt, ob eine Eingabe diese Form hat.

Aber: Praktische Programmiersprachen eignen sich schlecht als allgemeine Berechnungsmodelle, da sie viel zu kompliziert sind.

$\rightsquigarrow$ Wir wollen lieber unser Automatenmodell erweitern

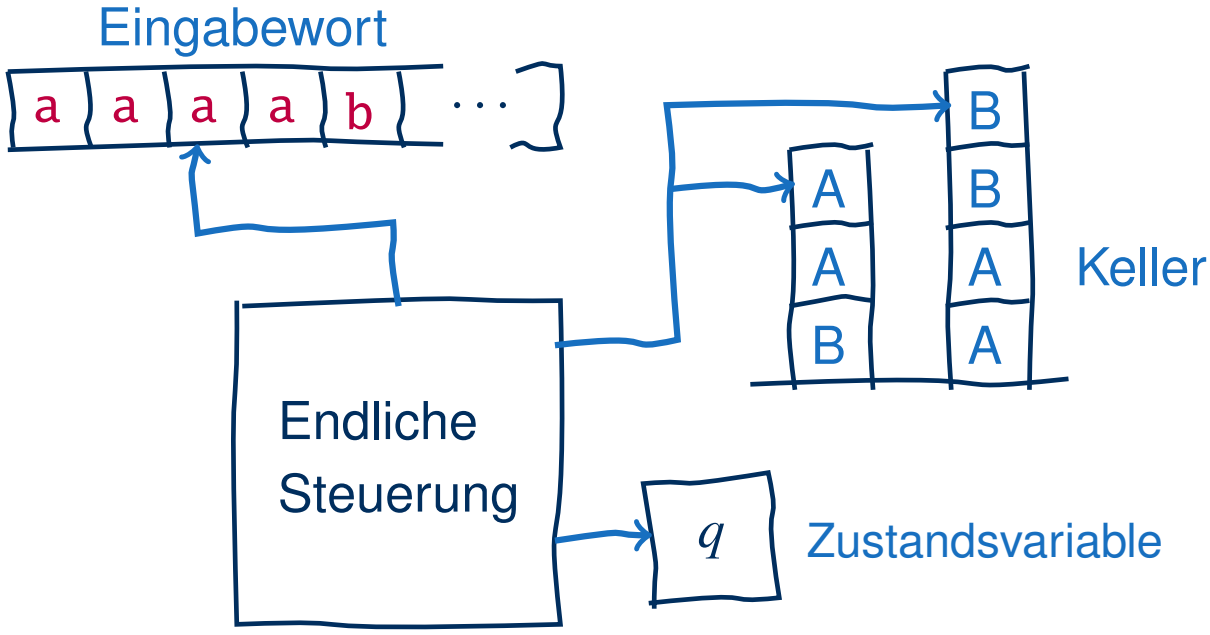
Jenseits PDAs (1)

Die Haupteinschränkung von Kellerautomaten war das eingeschränkte Speichermodell. Wir könnte man das erweitern?

Jenseits PDAs (1): Mehr Stapel

Die Haupteinschränkung von Kellerautomaten war das eingeschränkte Speichermodell. Wir könnte man das erweitern?

- Man könnte statt eines Stapelspeichers zwei (oder mehr) Stapel verwenden
- Automatenübergänge werden zum Zugriff auf weitere Stapel entsprechend erweitert



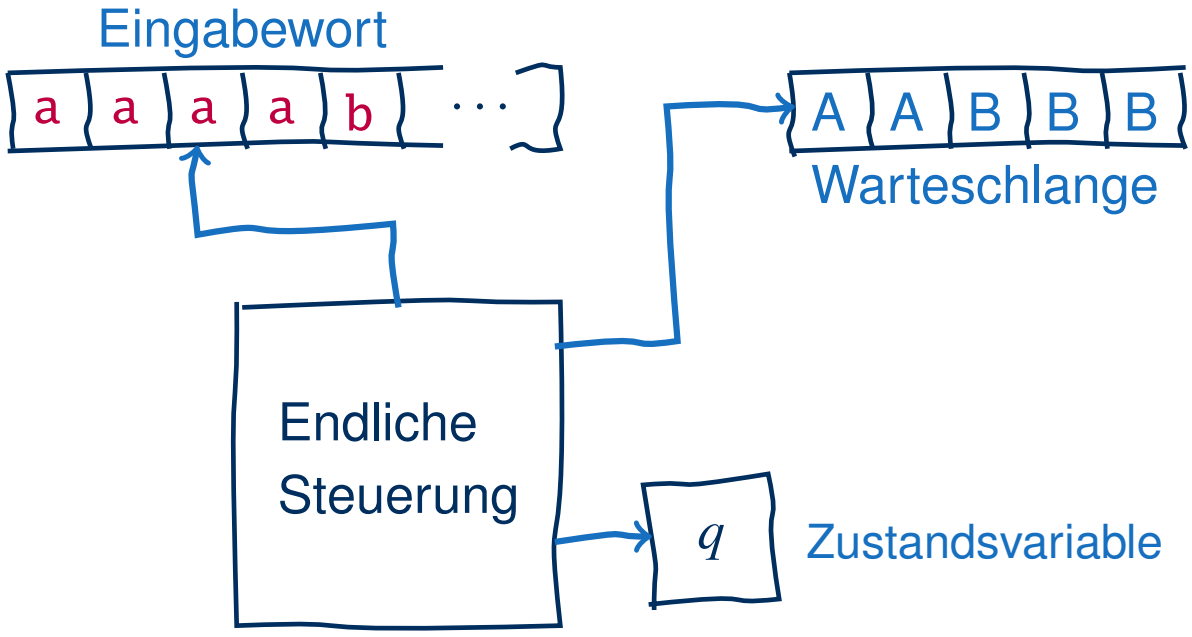
Jenseits PDAs (2)

Die Haupteinschränkung von Kellerautomaten war das eingeschränkte Speichermodell. Wir könnte man das erweitern?

Jenseits PDAs (2): „Warteschlangenautomaten“

Die Haupteinschränkung von Kellerautomaten war das eingeschränkte Speichermodell. Wir könnte man das erweitern?

- Man könnte statt eines Stapelspeichers eine Warteschlange (Queue) verwenden $\rightsquigarrow$ first-in/first-out (FIFO)
- Definition fast genau wie bei PDAs, aber mit `enqueue`/`dequeue` statt `push`/`pop`



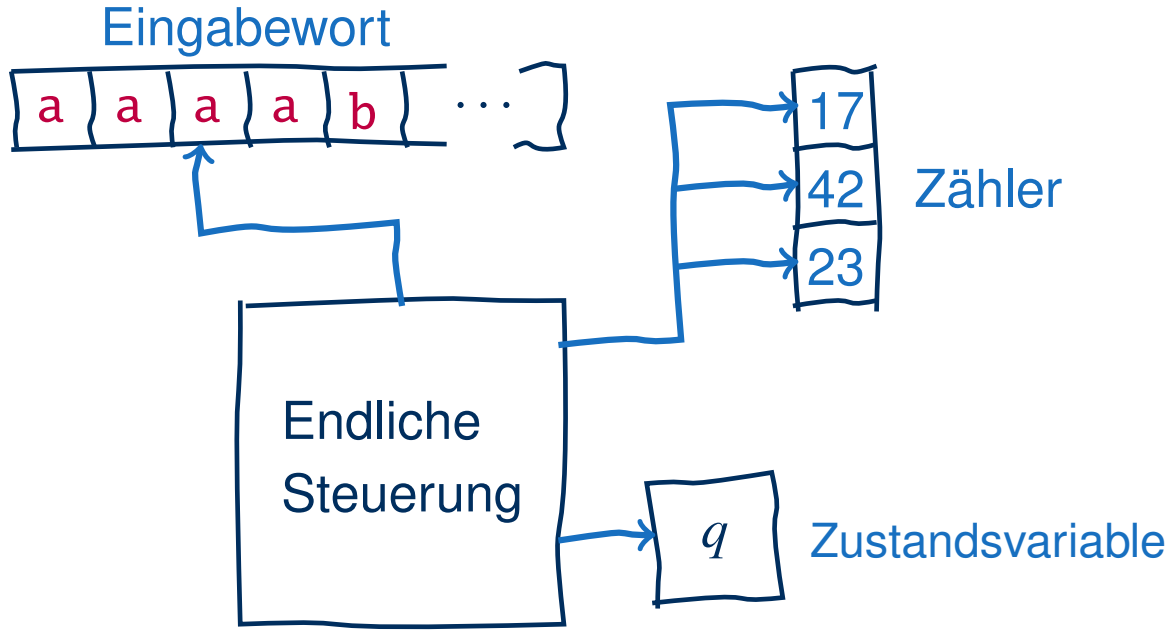
Jenseits PDAs (3)

Die Haupteinschränkung von Kellerautomaten war das eingeschränkte Speichermodell. Wir könnte man das erweitern?

Jenseits PDAs (3): Zählerautomaten

Die Haupteinschränkung von Kellerautomaten war das eingeschränkte Speichermodell. Wir könnte man das erweitern?

- Man könnte statt eines Stapelspeichers (endlich viele) Speicherplätze für natürliche Zahlen einführen
- Automatenübergänge könnten einzelne Variablen inkrementieren, dekrementieren, auf Gleichheit mit 0 testen, ...



Jenseits PDAs (4): Programme statt Automaten

Eventuell könnte man auch vom Automatenmodell abweichen und stattdessen eine einfache Programmiersprache definieren.

Jenseits PDAs (4): Programme statt Automaten

Eventuell könnte man auch vom Automatenmodell abweichen und stattdessen eine einfache Programmiersprache definieren.

Einfache Ausdrucksmittel:

- Variablen, die Zahlen speichern können
- Wertezuweisungen, die Variablen das Ergebnis eines Ausdrucks (z.B. aus $+$, $-$, Variablen, Zahlen) zuweisen
- Schleifen der Form **while** $x \neq 0$ **do:** ...

$\rightsquigarrow$ Sogenannte WHILE-Programme

Jenseits PDAs (4): Programme statt Automaten

Eventuell könnte man auch vom Automatenmodell abweichen und stattdessen eine einfache Programmiersprache definieren.

Einfache Ausdrucksmittel:

- Variablen, die Zahlen speichern können
- Wertezuweisungen, die Variablen das Ergebnis eines Ausdrucks (z.B. aus $+$, $-$, Variablen, Zahlen) zuweisen
- Schleifen der Form **while** $x \neq 0$ **do:** ...

$\rightsquigarrow$ Sogenannte WHILE-Programme

Statt **while** könnte man auch **if** und **goto** einführen
$\rightsquigarrow$ Sogenannte GOTO-Programme

Viele mögliche Wege

Bisher gesammelte Ideen:

- PDAs mit zwei Stapeln
- PDAs mit einer noch größeren Zahl an Stapeln
- Warteschlangenautomaten
- Zählerautomaten
- WHILE-Programme
- GOTO-Programme

Man kann jedes dieser Berechnungsmodelle formal definieren . . .

Viele mögliche Wege

Bisher gesammelte Ideen:

- PDAs mit zwei Stapeln
- PDAs mit einer noch größeren Zahl an Stapeln
- Warteschlangenautomaten
- Zählerautomaten
- WHILE-Programme
- GOTO-Programme

Man kann jedes dieser Berechnungsmodelle formal definieren . . .

Es ergeben sich daher viele Fragen . . .

Welche Sprachklasse können diese Modelle jeweils erkennen?

Viele mögliche Wege

Bisher gesammelte Ideen:

- PDAs mit zwei Stapeln
- PDAs mit einer noch größeren Zahl an Stapeln
- Warteschlangenautomaten
- Zählerautomaten
- WHILE-Programme
- GOTO-Programme

Man kann jedes dieser Berechnungsmodelle formal definieren . . .

Es ergeben sich daher viele Fragen . . .

Welche Sprachklasse können diese Modelle jeweils erkennen?

. . . aber immer wieder die gleiche Antwort

Viele mögliche Wege

Bisher gesammelte Ideen:

- PDAs mit zwei Stapeln
- PDAs mit einer noch größeren Zahl an Stapeln
- Warteschlangenautomaten
- Zählerautomaten
- WHILE-Programme
- GOTO-Programme

Man kann jedes dieser Berechnungsmodelle formal definieren . . .

Es ergeben sich daher viele Fragen . . .

Welche Sprachklasse können diese Modelle jeweils erkennen?

. . . aber immer wieder die gleiche Antwort:

Genau die Typ-0-Sprachen.

Zusammenfassung und Ausblick

Deterministische Typ-2-Sprachen sind praktisch wichtig, da effizient parsebar

Viele Fragestellungen für Typ-2-Sprachen sind unentscheidbar, wobei deterministische Sprachen noch etwas mehr erlauben

Zahlreiche naheliegende Erweiterungen von PDAs führen alle zur gleichen Ausdrucksstärke (Typ 0)

Offene Fragen:

- Welches Berechnungsmodell sollen wir nun verwenden?
- Wenn alle Modelle Typ 0 liefern, was ist dann mit Typ 1?
- Unterscheiden sich Typ 0 und Typ 1 überhaupt?