Script generated by TTT

Title: Nipkow: Theo (03.06.2019)

Date: Mon Jun 03 14:15:16 CEST 2019

Duration: 236:12 min

Pages: 96

4.9 Äquivalenz von PDAs und CFGs

Satz 4.57 (CFG → PDA)

Zu jeder CFG G kann man einen PDA M konstruieren, der mit leerem Keller akzeptiert, so dass $L_\epsilon(M) = L(G)$.

Konstruktion:
Zuerst bringen wir alle Produktionen von G in die Form

$$A \to bB_1 \dots B_k$$

wobei $b \in \Sigma \cup \{\epsilon\}$.

Methode: Für jedes $a \in \Sigma$

1. füge ein neues A_a zu V hinzu,
2. ersetze a rechts in P durch A_a (außer am Kopfende),
3. füge eine neue Produkion $A_a \to a$ hinzu.

Alle Produktionen in $G = (V, \Sigma, P, S)$ haben jetzt die Form

$$A \to bB_1 \dots B_k$$

Der PDA wird wie folgt definiert:

$$M \quad := \quad (\{q\},\ \Sigma,\ V,\ q,\ S,\ \delta)$$

wobei

$$(A \to b\beta) \in P \implies \delta(q, b, A) \ni (q, \beta)$$

Satz 4.60 (PDA → CFG)

Zu jedem PDA $M = (Q, \Sigma, \Gamma, q_0, Z_0, \delta)$, der mit leerem Keller akzeptiert, kann man eine CFG G konstruieren mit $L(G) = L_\epsilon(M)$.

Konstruktion: $G := (V, \Sigma, P, S)$ mit

$V := Q \times \Gamma \times Q \cup \{S\}$ wobei wir die Tripel mit $[.,.,.]$ notieren

und P die folgenden Produktionen enthält:

Satz 4.60 (PDA → CFG)

Zu jedem PDA $M = (Q, \Sigma, \Gamma, q_0, Z_0, \delta)$, der mit leerem Keller akzeptiert, kann man eine CFG G konstruieren mit $L(G) = L_\epsilon(M)$.

Konstruktion: $G := (V, \Sigma, P, S)$ mit

$V := Q \times \Gamma \times Q \cup \{S\}$ wobei wir die Tripel mit $[.,.,.]$ notieren

und P die folgenden Produktionen enthält:

- $S \to [q_0, Z_0, q]$ für alle $q \in Q$

Satz 4.60 (PDA → CFG)

Zu jedem PDA $M = (Q, \Sigma, \Gamma, q_0, Z_0, \delta)$, der mit leerem Keller akzeptiert, kann man eine CFG G konstruieren mit $L(G) = L_\epsilon(M)$.

Konstruktion: $G := (V, \Sigma, P, S)$ mit

$V := Q \times \Gamma \times Q \cup \{S\}$ wobei wir die Tripel mit $[.,.,.]$ notieren

und P die folgenden Produktionen enthält:

- $S \to [q_0, Z_0, q]$ für alle $q \in Q$
- Für alle $\delta(q, b, Z) \ni (r_0, Z_1 \dots Z_k)$ und für alle $r_1, \dots, r_k \in Q$:

$$[q, Z, r_k] \to b[r_0, Z_1, r_1][r_1, Z_2, r_2] \dots [r_{k-1}, Z_k, r_k]$$

Satz 4.60 (PDA → CFG)

Zu jedem PDA $M = (Q, \Sigma, \Gamma, q_0, Z_0, \delta)$, der mit leerem Keller akzeptiert, kann man eine CFG G konstruieren mit $L(G) = L_\epsilon(M)$.

Konstruktion: $G := (V, \Sigma, P, S)$ mit

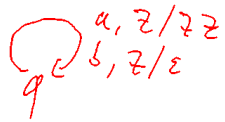
$V := Q \times \Gamma \times Q \cup \{S\}$ wobei wir die Tripel mit $[.,.,.]$ notieren

und P die folgenden Produktionen enthält:

- $S \to [q_0, Z_0, q]$ für alle $q \in Q$
- Für alle $\delta(q, b, Z) \ni (r_0, Z_1 \dots Z_k)$ und für alle $r_1, \dots, r_k \in Q$:

$$[q, Z, r_k] \to b[r_0, Z_1, r_1][r_1, Z_2, r_2] \dots [r_{k-1}, Z_k, r_k]$$

Idee: $[q, Z, r_k] \to_G^* w$ gdw $(q, w, Z) \to_M^* (r_k, \epsilon, \epsilon)$



$S \to [q, Z,$

Satz 4.60 (PDA → CFG)

Zu jedem PDA $M = (Q, \Sigma, \Gamma, q_0, Z_0, \delta)$, der mit leerem Keller akzeptiert, kann man eine CFG G konstruieren mit $L(G) = L_\epsilon(M)$.

Konstruktion: $G := (V, \Sigma, P, S)$ mit

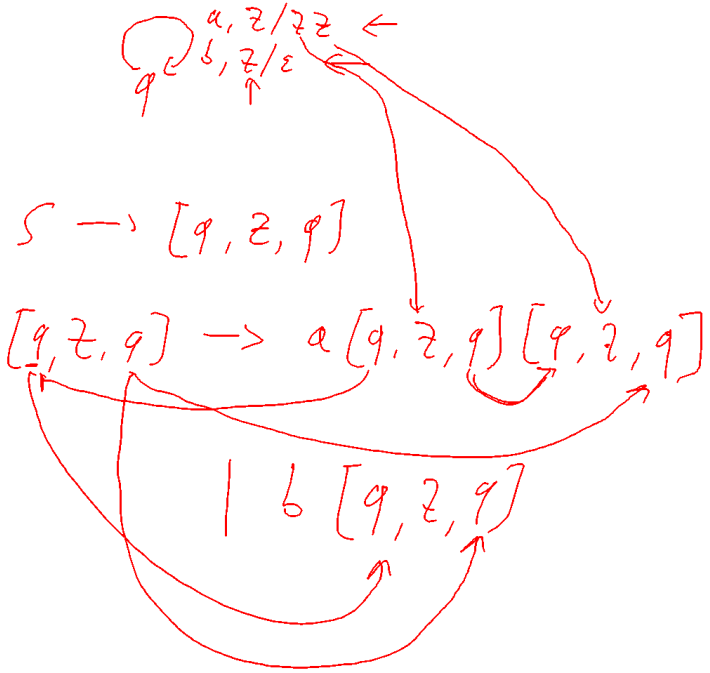
$V := Q \times \Gamma \times Q \cup \{S\}$ wobei wir die Tripel mit $[.,.,.]$ notieren

und P die folgenden Produktionen enthält:

- $S \to [q_0, Z_0, q]$ für alle $q \in Q$
- Für alle $\delta(q, b, Z) \ni (r_0, Z_1 \dots Z_k)$ und für alle $r_1, \dots, r_k \in Q$:

$$[q, Z, r_k] \to b[r_0, Z_1, r_1][r_1, Z_2, r_2] \dots [r_{k-1}, Z_k, r_k]$$

Idee: $[q, Z, r_k] \to_G^* w$ gdw $(q, w, Z) \to_M^* (r_k, \epsilon, \epsilon)$



Satz 4.60 (PDA → CFG)

Zu jedem PDA $M = (Q, \Sigma, \Gamma, q_0, Z_0, \delta)$, der mit leerem Keller akzeptiert, kann man eine CFG G konstruieren mit $L(G) = L_\epsilon(M)$.

Konstruktion: $G := (V, \Sigma, P, S)$ mit

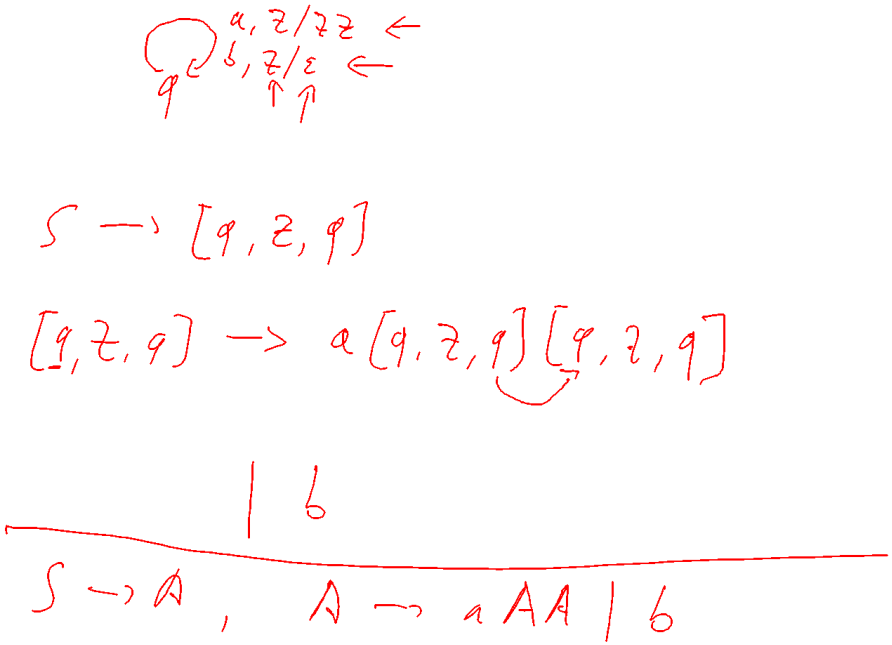
$V := Q \times \Gamma \times Q \cup \{S\}$ wobei wir die Tripel mit $[.,.,.]$ notieren

und P die folgenden Produktionen enthält:

- $S \to [q_0, Z_0, q]$ für alle $q \in Q$
- Für alle $\delta(q, b, Z) \ni (r_0, Z_1 \dots Z_k)$ und für alle $r_1, \dots, r_k \in Q$:

$$[q, Z, r_k] \to b[r_0, Z_1, r_1][r_1, Z_2, r_2] \dots [r_{k-1}, Z_k, r_k]$$

Idee: $[q, Z, r_k] \to_G^* w$ gdw $(q, w, Z) \to_M^* (r_k, \epsilon, \epsilon)$



Satz 4.60 (PDA $\to$ CFG)

Zu jedem PDA $M = (Q, \Sigma, \Gamma, q_0, Z_0, \delta)$, der mit leerem Keller akzeptiert, kann man eine CFG G konstruieren mit $L(G) = L_\epsilon(M)$.

Konstruktion: $G := (V, \Sigma, P, S)$ mit

$V := Q \times \Gamma \times Q \cup \{S\}$ wobei wir die Tripel mit $[.,.,.]$ notieren

und P die folgenden Produktionen enthält:

- $S \to [q_0, Z_0, q]$ für alle $q \in Q$
- Für alle $\delta(q, b, Z) \ni (r_0, Z_1 \dots Z_k)$ und für alle $r_1, \dots, r_k \in Q$:

$$[q, Z, r_k] \to b[r_0, Z_1, r_1][r_1, Z_2, r_2] \dots [r_{k-1}, Z_k, r_k]$$

Idee: $[q, Z, r_k] \to_G^* w$ gdw $(q, w, Z) \to_M^* (r_k, \epsilon, \epsilon)$

Satz 4.60 (PDA $\to$ CFG)

Zu jedem PDA $M = (Q, \Sigma, \Gamma, q_0, Z_0, \delta)$, der mit leerem Keller akzeptiert, kann man eine CFG G konstruieren mit $L(G) = L_\epsilon(M)$.

Konstruktion: $G := (V, \Sigma, P, S)$ mit

$V := Q \times \Gamma \times Q \cup \{S\}$ wobei wir die Tripel mit $[.,.,.]$ notieren

und P die folgenden Produktionen enthält:

- $S \to [q_0, Z_0, q]$ für alle $q \in Q$
- Für alle $\delta(q, b, Z) \ni (r_0, Z_1 \dots Z_k)$ und für alle $r_1, \dots, r_k \in Q$:

$$[q, Z, r_k] \to b[r_0, Z_1, r_1][r_1, Z_2, r_2] \dots [r_{k-1}, Z_k, r_k]$$

Idee: $[q, Z, r_k] \to_G^* w$ gdw $(q, w, Z) \to_M^* (r_k, \epsilon, \epsilon)$

Die $r_1, \dots, r_k$ sind potenzielle Zwischenzustände beim Akzeptieren der Teilwörter von $bu_1 \dots u_k = w$, die zu $Z_1, \dots, Z_k$ gehören. (Zerlegungssatz!)

Lemma 4.61

$[q, Z, p] \to_G^n w$ *gdw* $(q, w, Z) \to_M^n (p, \epsilon, \epsilon)$

Beweis:

($\Rightarrow$): Wenn $[q, Z, p] \to_G^n w$ dann $(q, w, Z) \to_M^n (p, \epsilon, \epsilon)$.

Lemma 4.61

$[q, Z, p] \to_G^n w$ *gdw* $(q, w, Z) \to_M^n (p, \epsilon, \epsilon)$

Beweis:

($\Rightarrow$): Wenn $[q, Z, p] \to_G^n w$ dann $(q, w, Z) \to_M^n (p, \epsilon, \epsilon)$.

Mit Induktion über n.

IA: Beh. gilt für alle Werte $< n$. Zz: Beh. gilt für n.

Lemma 4.61
$[q, Z, p] \to_G^n w$ *gdw* $(q, w, Z) \to_M^n (p, \epsilon, \epsilon)$

Beweis:
($\Rightarrow$): Wenn $[q, Z, p] \to_G^n w$ dann $(q, w, Z) \to_M^n (p, \epsilon, \epsilon)$.
Mit Induktion über n.
IA: Beh. gilt für alle Werte $< n$. Zz: Beh. gilt für n.
Die Ableitung $[q, Z, p] \to_G^n w$ muss von folgender Form sein:

$$\begin{array}{lll} [q, Z, r_k] & \to_G & b[r_0, Z_1, r_1] \dots [r_{k-1}, Z_k, r_k] \\ & \to_G^{n-1} & bu_1 \dots u_k = w \end{array}$$

mit $r_k = p$, $(r_0, Z_1 \dots Z_k) \in \delta(q, b, Z)$,
$[r_{i-1}, Z_i, r_i] \to_G^{n_i} u_i$ $(i = 1, \dots, k)$ und $\sum n_i = n - 1$.

Lemma 4.61
$[q, Z, p] \to_G^n w$ *gdw* $(q, w, Z) \to_M^n (p, \epsilon, \epsilon)$

Beweis:
($\Rightarrow$): Wenn $[q, Z, p] \to_G^n w$ dann $(q, w, Z) \to_M^n (p, \epsilon, \epsilon)$.
Mit Induktion über n.
IA: Beh. gilt für alle Werte $< n$. Zz: Beh. gilt für n.
Die Ableitung $[q, Z, p] \to_G^n w$ muss von folgender Form sein:

$$\begin{array}{lll} [q, Z, r_k] & \to_G & b[r_0, Z_1, r_1] \dots [r_{k-1}, Z_k, r_k] \\ & \to_G^{n-1} & bu_1 \dots u_k = w \end{array}$$

mit $r_k = p$, $(r_0, Z_1 \dots Z_k) \in \delta(q, b, Z)$,
$[r_{i-1}, Z_i, r_i] \to_G^{n_i} u_i$ $(i = 1, \dots, k)$ und $\sum n_i = n - 1$.
IA $\Rightarrow$ $(r_{i-1}, u_i, Z_i) \to_M^{n_i} (r_i, \epsilon, \epsilon)$

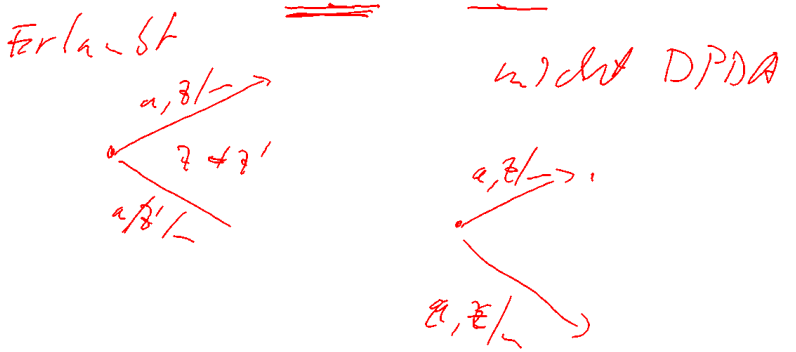
211

Beweis (Forts.):
($\Leftarrow$): Wenn $(q, w, Z) \to_M^n (p, \epsilon, \epsilon)$ dann $[q, Z, p] \to_G^n w$.
Mit Induktion über n.

4.10 Deterministische Kellerautomaten

Definition 4.63
Ein PDA heißt deterministisch (DPDA) gdw für alle $q \in Q$, $a \in \Sigma$ und $Z \in \Gamma$

$$|\delta(q, a, Z)| + |\delta(q, \epsilon, Z)| \leq 1$$

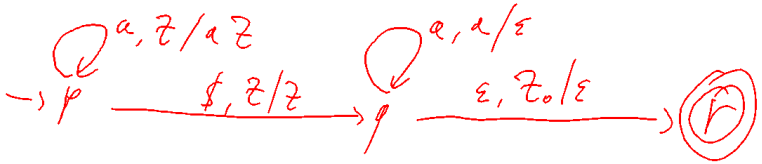


4.10 Deterministische Kellerautomaten

Definition 4.63
Ein PDA heißt deterministisch (DPDA) gdw für alle $q \in Q$, $a \in \Sigma$ und $Z \in \Gamma$

$$|\delta(q, a, Z)| + |\delta(q, \epsilon, Z)| \leq 1$$

Beispiel 4.64
Der folgende DPDA mit $\Sigma = \{0, 1, \$\}$, $\Gamma = \{Z_0, 0, 1\}$ akzeptiert die Sprache $\{w\$w^R \mid w \in \{0, 1\}^*\}$.



$q \in \{p, q\}$
$Z \in \Gamma$

Definition 4.65
Eine CFL heißt deterministisch (DCFL) gdw sie von einem DPDA akzeptiert wird.
Man kann zeigen: Die CFL $\{ww^R \mid w \in \{0, 1\}^*\}$ ist nicht deterministisch.

Da man jeden DFA leicht mit einem DPDA simulieren kann:

Fakt 4.66
Jede reguläre Sprache ist eine DCFL.

Definition 4.65
Eine CFL heißt deterministisch (DCFL) gdw sie von einem DPDA akzeptiert wird.
Man kann zeigen: Die CFL $\{ww^R \mid w \in \{0, 1\}^*\}$ ist nicht deterministisch.

Eine Sprache erfüllt die Präfix Bedingung gdw wenn sie keine zwei Wörter enthält, so dass das eine ein *echtes* Präfix des anderen ist.

Beispiel 4.67

- Die Sprache a^* erfüllt die Präfix Bedingung nicht. a, aa
- Die Sprache $\{w\$w^R \mid w \in \{0, 1\}^*\}$ erfüllt die Präfix Bedingung.

$w_1 \$ w_1^R$
$w_2 \$ w_2^R$

Eine Sprache erfüllt die Präfix Bedingung gdw wenn sie keine zwei Wörter enthält, so dass das eine ein *echtes* Präfix des anderen ist.

Beispiel 4.67

- Die Sprache a^* erfüllt die Präfix Bedingung nicht.
- Die Sprache $\{w\$w^R \mid w \in \{0,1\}^*\}$ erfüllt die Präfix Bedingung.

Lemma 4.68
$\exists$*DPDA* M. $L = L_\epsilon(M)$ $\Leftrightarrow$
$\exists$*DPDA* M. $L = L_F(M)$ *und* L *erfüllt die Präfix Bedingung*

Beweis: Übung!

Eine Sprache erfüllt die Präfix Bedingung gdw wenn sie keine zwei Wörter enthält, so dass das eine ein *echtes* Präfix des anderen ist.

Beispiel 4.67

- Die Sprache a^* erfüllt die Präfix Bedingung nicht.
- Die Sprache $\{w\$w^R \mid w \in \{0,1\}^*\}$ erfüllt die Präfix Bedingung.

Lemma 4.68
$\exists$*DPDA* M. $L = L_\epsilon(M)$ $\Leftrightarrow$
$\exists$*DPDA* M. $L = L_F(M)$ *und* L *erfüllt die Präfix Bedingung*

Beweis: Übung!

Es gibt also einen DPDA M mit $L_F(M) = L(a^*)$ aber keinen DPDA mit $L_\epsilon(M) = L(a^*)$.
Im Folgenden: Akzeptanz durch Endzustand.

216

Satz 4.69
Die Klasse der DCFLs ist unter Komplement abgeschlossen.

Satz 4.69
Die Klasse der DCFLs ist unter Komplement abgeschlossen.

Beweis: Komplex. Siehe zB Erk&Priese oder Kozen.

217

Satz 4.69
Die Klasse der DCFLs ist unter Komplement abgeschlossen.

Beweis: Komplex. Siehe zB Erk&Priese oder Kozen.

Da die CFLs *nicht* unter Komplement abgeschlossen sind:

Korollar 4.70
Die DCFLs sind eine echte Teilklasse der CFLs.

Hier ist eine konkrete Sprache in CFL\DCFL:
Sei $L_{ww} := \{ww \mid w \in \{a,b\}^*\}$.

Satz 4.69
Die Klasse der DCFLs ist unter Komplement abgeschlossen.

Beweis: Komplex. Siehe zB Erk&Priese oder Kozen.

Da die CFLs *nicht* unter Komplement abgeschlossen sind:

Korollar 4.70
Die DCFLs sind eine echte Teilklasse der CFLs.

Hier ist eine konkrete Sprache in CFL\DCFL:
Sei $L_{ww} := \{ww \mid w \in \{a,b\}^*\}$.
Dann ist $L := \{a,b\}^* \setminus L_{ww}$ eine CFL aber keine DCFL.

Satz 4.69
Die Klasse der DCFLs ist unter Komplement abgeschlossen.

Beweis: Komplex. Siehe zB Erk&Priese oder Kozen.

Da die CFLs *nicht* unter Komplement abgeschlossen sind:

Korollar 4.70
Die DCFLs sind eine echte Teilklasse der CFLs.

Hier ist eine konkrete Sprache in CFL\DCFL:
Sei $L_{ww} := \{ww \mid w \in \{a,b\}^*\}$.
Dann ist $L := \{a,b\}^* \setminus L_{ww}$ eine CFL aber keine DCFL.
L wird von folgender CFG erzeugt (ohne Beweis):

$$S \to AB \mid BA \mid A \mid B$$
$$A \to CAC \mid a \qquad B \to CBC \mid b \qquad C \to a \mid b$$

Satz 4.69
Die Klasse der DCFLs ist unter Komplement abgeschlossen.

Beweis: Komplex. Siehe zB Erk&Priese oder Kozen.

Da die CFLs *nicht* unter Komplement abgeschlossen sind:

Korollar 4.70
Die DCFLs sind eine echte Teilklasse der CFLs.

Hier ist eine konkrete Sprache in CFL\DCFL:
Sei $L_{ww} := \{ww \mid w \in \{a,b\}^*\}$.
Dann ist $L := \{a,b\}^* \setminus L_{ww}$ eine CFL aber keine DCFL.
L wird von folgender CFG erzeugt (ohne Beweis):

$$S \to AB \mid BA \mid A \mid B$$
$$A \to CAC \mid a \qquad B \to CBC \mid b \qquad C \to a \mid b$$

Wäre L eine DCFL, dann auch $\overline{L} = L_{ww}$ (wegen Satz 4.69).

Satz 4.69
Die Klasse der DCFLs ist unter Komplement abgeschlossen.

Beweis: Komplex. Siehe zB Erk&Priese oder Kozen.

Da die CFLs *nicht* unter Komplement abgeschlossen sind:

Korollar 4.70
Die DCFLs sind eine echte Teilklasse der CFLs.

Hier ist eine konkrete Sprache in CFL\DCFL:
Sei $L_{ww} := \{ww \mid w \in \{a,b\}^*\}$.
Dann ist $L := \{a,b\}^* \setminus L_{ww}$ eine CFL aber keine DCFL.
L wird von folgender CFG erzeugt (ohne Beweis):

$$S \to AB \mid BA \mid A \mid B$$
$$A \to CAC \mid a \qquad B \to CBC \mid b \qquad C \to a \mid b$$

Wäre L eine DCFL, dann auch $\overline{L} = L_{ww}$ (wegen Satz 4.69).
Aber mit dem Pumping Lemma kann man zeigen, dass L_{ww} keine CFL ist, und daher erst recht keine DCFL.

Lemma 4.71
Die Klasse der DCFLs ist weder unter Schnitt noch unter Vereinigung abgeschlossen.

Lemma 4.71
Die Klasse der DCFLs ist weder unter Schnitt noch unter Vereinigung abgeschlossen.

Beweis:
Erinnerung: CFLs nicht unter Schnitt abgeschlossen:

$$L_1 := \{a^i b^i c^j \mid i,j \in \mathbb{N}\} \text{ ist kontextfrei}$$
$$L_2 := \{a^i b^j c^j \mid i,j \in \mathbb{N}\} \text{ ist kontextfrei}$$

Lemma 4.71
Die Klasse der DCFLs ist weder unter Schnitt noch unter Vereinigung abgeschlossen.

Beweis:
Erinnerung: CFLs nicht unter Schnitt abgeschlossen:

$$L_1 := \{a^i b^i c^j \mid i,j \in \mathbb{N}\} \text{ ist kontextfrei}$$
$$L_2 := \{a^i b^j c^j \mid i,j \in \mathbb{N}\} \text{ ist kontextfrei}$$
$$L_1 \cap L_2 = \{a^i b^i c^i \mid i \in \mathbb{N}\} \text{ ist nicht kontextfrei}$$

Aber L_1 und L_2 sind sogar DCFLs.

Lemma 4.71
Die Klasse der DCFLs ist weder unter Schnitt noch unter Vereinigung abgeschlossen.

Beweis:
Erinnerung: CFLs nicht unter Schnitt abgeschlossen:

$$\begin{aligned} L_1 &:= \{a^i b^i c^j \mid i,j \in \mathbb{N}\} \text{ ist kontextfrei} \\ L_2 &:= \{a^i b^j c^j \mid i,j \in \mathbb{N}\} \text{ ist kontextfrei} \\ L_1 \cap L_2 &= \{a^i b^i c^i \mid i \in \mathbb{N}\} \text{ ist nicht kontextfrei} \end{aligned}$$

Aber L_1 und L_2 sind sogar DCFLs.

Da $L_1 \cap L_2 = \overline{\overline{L_1} \cup \overline{L_2}}$ können die DCFLs auch nicht unter Vereinigung abgeschlossen sein. □

Lemma 4.72
Jede DCFL ist nicht inhärent mehrdeutig, dh sie wird von einer nicht-mehrdeutigen Grammatik erzeugt.

Beweisidee: Die Konversion PDA → CFG erzeugt aus einem DPDA eine nicht-mehrdeutige CFG. [HMU]

Lemma 4.72
Jede DCFL ist nicht inhärent mehrdeutig, dh sie wird von einer nicht-mehrdeutigen Grammatik erzeugt.

Beweisidee: Die Konversion PDA → CFG erzeugt aus einem DPDA eine nicht-mehrdeutige CFG. [HMU]

Satz 4.73
Das Wortproblem für DCFLs ist in linearer Zeit lösbar.

Lemma 4.72
Jede DCFL ist nicht inhärent mehrdeutig, dh sie wird von einer nicht-mehrdeutigen Grammatik erzeugt.

Beweisidee: Die Konversion PDA → CFG erzeugt aus einem DPDA eine nicht-mehrdeutige CFG. [HMU]

Satz 4.73
Das Wortproblem für DCFLs ist in linearer Zeit lösbar.

Mehr Information: Vorlesungen zum Übersetzerbau

4.11 Tabellarischer Überblick

Abschlusseigenschaften

	Schnitt	Vereinigung	Komplement	Produkt	Stern

4.11 Tabellarischer Überblick

Abschlusseigenschaften

	Schnitt	Vereinigung	Komplement	Produkt	Stern
Regulär	ja	ja	ja	ja	ja

4.11 Tabellarischer Überblick

Abschlusseigenschaften

	Schnitt	Vereinigung	Komplement	Produkt	Stern
Regulär	ja	ja	ja	ja	ja
DCFL	nein	nein	ja	nein	nein
CFL	nein	ja	nein	ja	ja

Entscheidbarkeit

	Wortproblem			
DFA				
DPDA				
CFG				

4.11 Tabellarischer Überblick

Abschlusseigenschaften

	Schnitt	Vereinigung	Komplement	Produkt	Stern
Regulär	ja	ja	ja	ja	ja
DCFL	nein	nein	ja	nein	nein
CFL	nein	ja	nein	ja	ja

Entscheidbarkeit

	Wortproblem	Leerheit		
DFA	$O(n)$			
DPDA	$O(n)$			
CFG	$O(n^3)$			

4.11 Tabellarischer Überblick

Abschlusseigenschaften

	Schnitt	Vereinigung	Komplement	Produkt	Stern
Regulär	ja	ja	ja	ja	ja
DCFL	nein	nein	ja	nein	nein
CFL	nein	ja	nein	ja	ja

Entscheidbarkeit

	Wortproblem	Leerheit	Äquivalenz	Schnittproblem
DFA	$O(n)$	ja	ja	
DPDA	$O(n)$	ja	ja	
CFG	$O(n^3)$	ja	nein(*)	

Sénizergues (1997), Stirling (2001)
(*) Vorschau

Teil 2
Berechenbarkeit, Entscheidbarkeit, Komplexität

Überblick:

- Was kann man berechnen?
  - Welche Funktionen kann man in endlicher Zeit berechnen?
  - Welche Eigenschaften von Objekten können in endlicher Zeit entschieden werden?
- Mit welchen Sprachen/Maschinen?

5.1 Der Begriff der Berechenbarkeit

5.1 Der Begriff der Berechenbarkeit

Eine Funktion $f : \mathbb{N}^k \to \mathbb{N}$ ist intuitiv berechenbar, wenn es einen Algorithmus gibt, der bei Eingabe $(n_1, \ldots, n_k) \in \mathbb{N}^k$

- nach endlich vielen Schritten mit Ergebnis $f(n_1, \ldots, n_k)$ hält, falls $f(n_1, \ldots, n_k)$ definiert ist,
- und nicht terminiert, falls $f(n_1, \ldots, n_k)$ nicht definiert ist.

5.1 Der Begriff der Berechenbarkeit

Eine Funktion $f : \mathbb{N}^k \to \mathbb{N}$ ist intuitiv berechenbar, wenn es einen Algorithmus gibt, der bei Eingabe $(n_1, \ldots, n_k) \in \mathbb{N}^k$

- nach endlich vielen Schritten mit Ergebnis $f(n_1, \ldots, n_k)$ hält, falls $f(n_1, \ldots, n_k)$ definiert ist,
- und nicht terminiert, falls $f(n_1, \ldots, n_k)$ nicht definiert ist.

Was bedeutet „Algorithmus“? Assembler? C? Java? OCaml? Macht es einen Unterschied?

5.1 Der Begriff der Berechenbarkeit

Eine Funktion $f : \mathbb{N}^k \to \mathbb{N}$ ist intuitiv berechenbar, wenn es einen Algorithmus gibt, der bei Eingabe $(n_1, \ldots, n_k) \in \mathbb{N}^k$

- nach endlich vielen Schritten mit Ergebnis $f(n_1, \ldots, n_k)$ hält, falls $f(n_1, \ldots, n_k)$ definiert ist,
- und nicht terminiert, falls $f(n_1, \ldots, n_k)$ nicht definiert ist.

Was bedeutet „Algorithmus“? Assembler? C? Java? OCaml? Macht es einen Unterschied?

Achtung: Berechenbarkeit setzt zwei verschiedene Dinge in Beziehung:

- Algorithmen, dh endliche Wörter.
- Mathematische Funktionen, dh Mengen von Paaren.

5.1 Der Begriff der Berechenbarkeit

Eine Funktion $f : \mathbb{N}^k \to \mathbb{N}$ ist intuitiv berechenbar, wenn es einen Algorithmus gibt, der bei Eingabe $(n_1, \ldots, n_k) \in \mathbb{N}^k$

- nach endlich vielen Schritten mit Ergebnis $f(n_1, \ldots, n_k)$ hält, falls $f(n_1, \ldots, n_k)$ definiert ist,
- und nicht terminiert, falls $f(n_1, \ldots, n_k)$ nicht definiert ist.

Was bedeutet „Algorithmus“? Assembler? C? Java? OCaml? Macht es einen Unterschied?

Achtung: Berechenbarkeit setzt zwei verschiedene Dinge in Beziehung:

- Algorithmen, dh endliche Wörter.
- Mathematische Funktionen, dh Mengen von Paaren.

ZB beschreibt $x \mapsto x^2$ die Funktion

$$\{(0,0), (1,1), (2,4), (3,9), (4,16), (5,25), \ldots\}$$

Terminologie: Eine Funktion $f : A \to B$ ist

total gdw $f(a)$ für alle $a \in A$ definiert ist.

Terminologie: Eine Funktion $f : A \to B$ ist

total gdw $f(a)$ für alle $a \in A$ definiert ist.

partiell gdw $f(a)$ auch undefiniert sein kann.

Terminologie: Eine Funktion $f : A \to B$ ist

total gdw $f(a)$ für alle $a \in A$ definiert ist.

partiell gdw $f(a)$ auch undefiniert sein kann.

echt partiell gdw sie nicht total ist.

Terminologie: Eine Funktion $f : A \to B$ ist

total	gdw $f(a)$ für alle $a \in A$ definiert ist.
partiell	gdw $f(a)$ auch undefiniert sein kann.
echt partiell	gdw sie nicht total ist.

Beispiel 5.1

Jeder Algorithmus berechnet eine partielle Funktion.

Terminologie: Eine Funktion $f : A \to B$ ist

total	gdw $f(a)$ für alle $a \in A$ definiert ist.
partiell	gdw $f(a)$ auch undefiniert sein kann.
echt partiell	gdw sie nicht total ist.

Beispiel 5.1

Jeder Algorithmus berechnet eine partielle Funktion.

Der Algorithmus

```
input(n);
while true do ;
```

berechnet die überall undefinierte Funktion, dh $\emptyset \subset \mathbb{N} \to \mathbb{N}$.

224

$\pi = 3,1415$

Beispiel 5.2

Ist die Funktion

$$f_1(n) := \begin{cases} 1 & \text{falls } n \text{ als Ziffernfolge Anfangsstück der Dezimalbruchentwicklung von } \pi \text{ ist} \\ 0 & \text{sonst} \end{cases}$$

berechenbar? (Bsp: $f_1(31415) = 1$ aber $f_1(315) = 0$)

Beispiel 5.2

Ist die Funktion

$$f_1(n) := \begin{cases} 1 & \text{falls } n \text{ als Ziffernfolge Anfangsstück der Dezimalbruchentwicklung von } \pi \text{ ist} \\ 0 & \text{sonst} \end{cases}$$

berechenbar? (Bsp: $f_1(31415) = 1$ aber $f_1(315) = 0$)

Ja, denn es Algorithmen gibt, die π iterativ auf beliebig viele Dezimalstellen genau berechnet werden.

225

Beispiel 5.3

Ist die Funktion

$$f_2(n) := \begin{cases} 1 & \text{falls } n \text{ als Ziffernfolge irgendwo in der Dezimalbruchentwicklung von } \pi \text{ vorkommt} \\ 0 & \text{sonst} \end{cases}$$

berechenbar? (Bsp: $f_2(415) = 1$)

Beispiel 5.3

Ist die Funktion

$$f_2(n) := \begin{cases} 1 & \text{falls } n \text{ als Ziffernfolge irgendwo in der Dezimalbruchentwicklung von } \pi \text{ vorkommt} \\ 0 & \text{sonst} \end{cases}$$

berechenbar? (Bsp: $f_2(415) = 1$)

Unbekannt!

Beispiel 5.3

Ist die Funktion

$$f_2(n) := \begin{cases} 1 & \text{falls } n \text{ als Ziffernfolge irgendwo in der Dezimalbruchentwicklung von } \pi \text{ vorkommt} \\ 0 & \text{sonst} \end{cases}$$

berechenbar? (Bsp: $f_2(415) = 1$)

Unbekannt!

Durch schrittweise Approximation und Suche in der Dezimalbruchentwicklung von π kann man feststellen, dass n vorkommt.

Beispiel 5.3

Ist die Funktion

$$f_2(n) := \begin{cases} 1 & \text{falls } n \text{ als Ziffernfolge irgendwo in der Dezimalbruchentwicklung von } \pi \text{ vorkommt} \\ 0 & \text{sonst} \end{cases}$$

berechenbar? (Bsp: $f_2(415) = 1$)

Unbekannt!

Durch schrittweise Approximation und Suche in der Dezimalbruchentwicklung von π kann man feststellen, dass n vorkommt.

Aber wie stellt man fest, dass n *nicht* vorkommt?

Beispiel 5.3

Ist die Funktion

$$f_2(n) := \begin{cases} 1 & \text{falls } n \text{ als Ziffernfolge irgendwo in der Dezimalbruchentwicklung von } \pi \text{ vorkommt} \\ 0 & \text{sonst} \end{cases}$$

berechenbar? (Bsp: $f_2(415) = 1$)

Unbekannt!

Durch schrittweise Approximation und Suche in der Dezimalbruchentwicklung von π kann man feststellen, dass n vorkommt. Aber wie stellt man fest, dass n *nicht* vorkommt? Nichttermination statt 0!

Vielleicht gibt es aber einen (noch zu findenden) mathematischen Satz, der genaue Aussagen über die in π vorkommenden Ziffernfolgen macht.

Beispiel 5.4

Ist die Funktion

$$f_4(n) := \begin{cases} 1 & \text{falls mindestens } n \text{ mal hintereinander irgendwo in der Dezimalbruchentwicklung von } \pi \text{ eine 0 vorkommt} \\ 0 & \text{sonst} \end{cases}$$

berechenbar?

Beispiel 5.4

Ist die Funktion

$$f_4(n) := \begin{cases} 1 & \text{falls mindestens } n \text{ mal hintereinander irgendwo in der Dezimalbruchentwicklung von } \pi \text{ eine 0 vorkommt} \\ 0 & \text{sonst} \end{cases}$$

berechenbar?

Ja, denn

- entweder kommt 0^n für beliebig große n vor, dann ist $f_4(n) = 1$ für alle n,

Beispiel 5.4

Ist die Funktion

$$f_4(n) := \begin{cases} 1 & \text{falls mindestens } n \text{ mal hintereinander irgendwo in der Dezimalbruchentwicklung von } \pi \text{ eine 0 vorkommt} \\ 0 & \text{sonst} \end{cases}$$

berechenbar?

Ja, denn

- entweder kommt 0^n für beliebig große n vor, dann ist $f_4(n) = 1$ für alle n,
- oder es gibt eine längste vorkommende Sequenz 0^m, dann ist $f_4(n) = 1$ für $n \leq m$ und $f_4(n) = 0$ sonst.

Beispiel 5.4

Ist die Funktion

$$f_4(n) := \begin{cases} 1 & \text{falls mindestens } n \text{ mal hintereinander irgendwo in der Dezimalbruchentwicklung von } \pi \text{ eine 0 vorkommt} \\ 0 & \text{sonst} \end{cases}$$

berechenbar?

Ja, denn

- entweder kommt 0^n für beliebig große n vor, dann ist $f_4(n) = 1$ für alle n,
- oder es gibt eine längste vorkommende Sequenz 0^m, dann ist $f_4(n) = 1$ für $n \leq m$ und $f_4(n) = 0$ sonst.

Beide Funktionen sind berechenbar.

Satz 5.5

Es gibt nicht-berechenbare Funktionen $\mathbb{N} \to \{0, 1\}$.

Satz 5.5

Es gibt nicht-berechenbare Funktionen $\mathbb{N} \to \{0, 1\}$.

Beweis:

Es gibt nur abzählbar viele Algorithmen, aber überabzählbar viele Funktionen in $\mathbb{N} \to \{0, 1\}$. □

Erinnerung: Eine Menge M ist abzählbar falls es eine injektive Funktion $M \to \mathbb{N}$ gibt.

Erinnerung: Eine Menge M ist abzählbar falls es eine injektive Funktion $M \to \mathbb{N}$ gibt.

Äquivalente Definitionen:

Erinnerung: Eine Menge M ist abzählbar falls es eine injektive Funktion $M \to \mathbb{N}$ gibt.

Äquivalente Definitionen:

- Entweder gibt es eine Bijektion $M \to \{0, \ldots, n\}$ für ein $n \in \mathbb{N}$, oder eine Bijektion $M \to \mathbb{N}$.
- Es gibt eine Nummerierung der Elemente von M.

Erinnerung: Eine Menge M ist abzählbar falls es eine injektive Funktion $M \to \mathbb{N}$ gibt.

Äquivalente Definitionen:

- Entweder gibt es eine Bijektion $M \to \{0, \ldots, n\}$ für ein $n \in \mathbb{N}$, oder eine Bijektion $M \to \mathbb{N}$.
- Es gibt eine Nummerierung der Elemente von M.

Eine Menge ist überabzählbar wenn sie nicht abzählbar ist.

Lemma 5.6

Σ^* ist abzählbar, falls Σ endlich ist.

Lemma 5.6

Σ^* *ist abzählbar, falls* Σ *endlich ist.*

Beweis:

Durch Beispiel:

$$\begin{array}{lcl} \{0,1\}^* & = & \{\epsilon,\ 0,\ 1,\ 00,\ 01,\ 10,\ 11,\ 000,\ldots\} \\ \mathbb{N} & = & \{0,\ 1,\ 2,\ 3,\ 4,\ 5,\ 6,\ 7,\ldots\} \end{array}$$

□

Lemma 5.6

Σ^* *ist abzählbar, falls* Σ *endlich ist.*

Beweis:

Durch Beispiel:

$$\begin{array}{lcl} \{0,1\}^* & = & \{\epsilon,\ 0,\ 1,\ 00,\ 01,\ 10,\ 11,\ 000,\ldots\} \\ \mathbb{N} & = & \{0,\ 1,\ 2,\ 3,\ 4,\ 5,\ 6,\ 7,\ldots\} \end{array}$$

□

Annahme:

Ein Algorithmus ist ein Wort über einem endlichen Alphabet.

Gilt für alle bekannten Programmiersprachen.

Lemma 5.6

Σ^* *ist abzählbar, falls* Σ *endlich ist.*

Beweis:

Durch Beispiel:

$$\begin{array}{lcl} \{0,1\}^* & = & \{\epsilon,\ 0,\ 1,\ 00,\ 01,\ 10,\ 11,\ 000,\ldots\} \\ \mathbb{N} & = & \{0,\ 1,\ 2,\ 3,\ 4,\ 5,\ 6,\ 7,\ldots\} \end{array}$$

□

Annahme:

Ein Algorithmus ist ein Wort über einem endlichen Alphabet.

Gilt für alle bekannten Programmiersprachen.

$\Longrightarrow$ Die Menge der Algorithmen ist abzählbar.

Satz 5.7

Die Menge aller Funktionen $\mathbb{N} \to \{0,1\}$ *ist überabzählbar.*

Satz 5.7

Die Menge aller Funktionen $\mathbb{N} \to \{0,1\}$ ist überabzählbar.

Beweis:

Indirekt. Nimm an, die Menge der Funktionen sei abzählbar.

f_0	
f_1	
f_2	
f_3	
$\vdots$	

Satz 5.7

Die Menge aller Funktionen $\mathbb{N} \to \{0,1\}$ ist überabzählbar.

Beweis:

Indirekt. Nimm an, die Menge der Funktionen sei abzählbar.

	0	1	2	3	...
f_0					
f_1					
f_2					
f_3					
$\vdots$					

Satz 5.7

Die Menge aller Funktionen $\mathbb{N} \to \{0,1\}$ ist überabzählbar.

Beweis:

Indirekt. Nimm an, die Menge der Funktionen sei abzählbar.

	0	1	2	3	...
f_0	0	1	1	0	...
f_1					
f_2					
f_3					
$\vdots$					

Satz 5.7

Die Menge aller Funktionen $\mathbb{N} \to \{0,1\}$ ist überabzählbar.

Beweis:

Indirekt. Nimm an, die Menge der Funktionen sei abzählbar.

	0	1	2	3	...
f_0	0	1	1	0	...
f_1	1	0	0	0	...
f_2					
f_3					
$\vdots$					

Satz 5.7

Die Menge aller Funktionen $\mathbb{N} \to \{0,1\}$ ist überabzählbar.

Beweis:

Indirekt. Nimm an, die Menge der Funktionen sei abzählbar.

	0	1	2	3	...
f_0	0	1	1	0	...
f_1	1	0	0	0	...
f_2	0	0	1	0	...
f_3	0	0	1	0	...
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$

Satz 5.7

Die Menge aller Funktionen $\mathbb{N} \to \{0,1\}$ ist überabzählbar.

Beweis:

Indirekt. Nimm an, die Menge der Funktionen sei abzählbar.

	0	1	2	3	...
f_0	0	1	1	0	...
f_1	1	0	0	0	...
f_2	0	0	1	0	...
f_3	0	0	1	0	...
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$

Eine Funktion f, die nicht in der Tabelle ist: ¬Diagonale!

f	1	1	0	1	...

Widerspruch!

Formal: Sei $f(i) := 1 - f_i(i)$. Dann $f \neq f_i$ für alle i, da $f(i) \neq f_i(i)$. □

Satz 5.8

Wenn Algorithmen als endliche Wörter kodiert werden können, dann gibt es nicht-berechenbare Funktionen $\mathbb{N} \to \{0,1\}$.

Beweis:

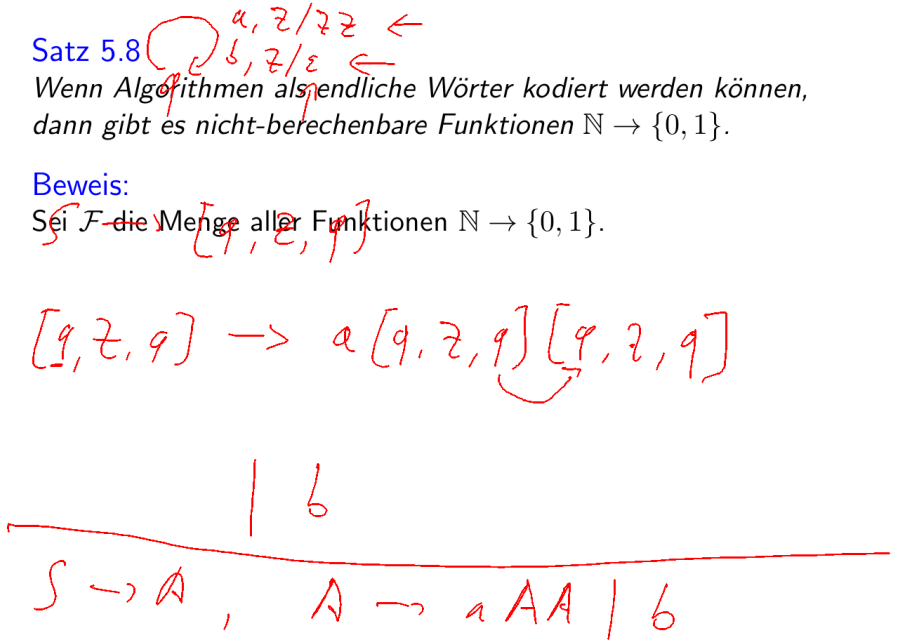
Sei $\mathcal{F}$ die Menge aller Funktionen $\mathbb{N} \to \{0,1\}$.
Sei $\mathcal{A}$ die Menge der Wörter, die Algorithmen kodieren.
Beweis durch Widerspruch.
Annahme: Alle Funktionen von $\mathcal{F}$ sind berechenbar.

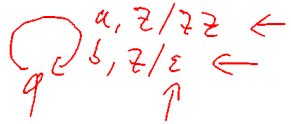
Satz 5.8

Wenn Algorithmen als endliche Wörter kodiert werden können, dann gibt es nicht-berechenbare Funktionen $\mathbb{N} \to \{0,1\}$.

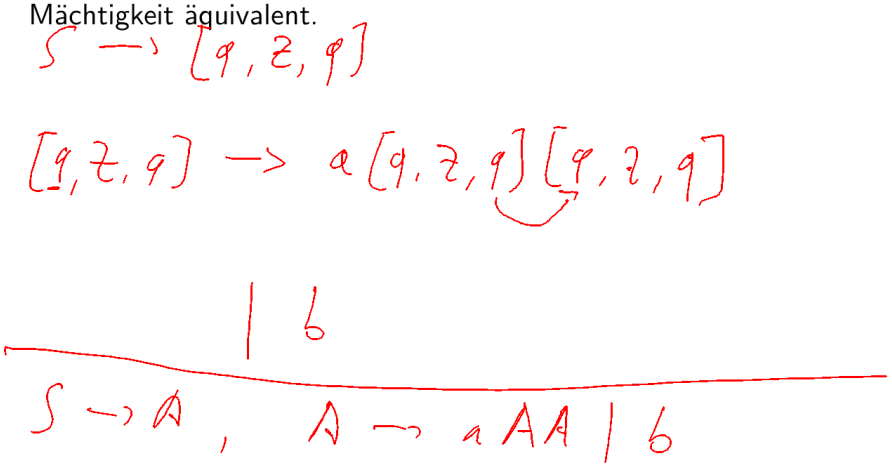
Beweis:

Sei $\mathcal{F}$ die Menge aller Funktionen $\mathbb{N} \to \{0,1\}$.





Es wurde bewiesen: Alle diese Beschreibungsmethoden sind in ihrer Mächtigkeit äquivalent.

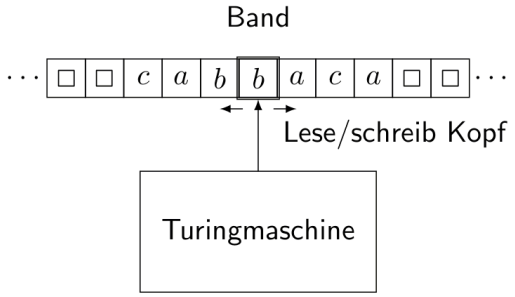


Es wurde bewiesen: Alle diese Beschreibungsmethoden sind in ihrer Mächtigkeit äquivalent.

Church-Turing These
Der formale Begriff der Berechenbarkeit mit Turing-Maschinen (bzw λ-Kalkül etc) stimmt mit dem intuitiven Berechenbarkeitsbegriff überein.

Die Church-Turing-These ist keine formale Aussage und so nicht beweisbar. Sie wird jedoch allgemein akzeptiert.

5.2 Turingmaschinen



Satz 5.8
Wenn Algorithmen als endliche Wörter kodiert werden können, dann gibt es nicht-berechenbare Funktionen $\mathbb{N} \to \{0,1\}$.

Beweis:
Sei $\mathcal{F}$ die Menge aller Funktionen $\mathbb{N} \to \{0,1\}$.
Sei $\mathcal{A}$ die Menge der Wörter, die Algorithmen kodieren.
Beweis durch Widerspruch.
Annahme: Alle Funktionen von $\mathcal{F}$ sind berechenbar.
Da $\mathcal{A}$ abzählbar ist (Lemma 5.6), gibt es eine injektive Funktion $anum\colon \mathcal{A} \to \mathbb{N}$.
Definiere $fnum\colon \mathcal{F} \to \mathbb{N}$ durch

$$fnum(f) = \min\{anum(a) \mid a \text{ berechnet } f\ \}$$

$fnum$ is injektiv: Wenn $fnum(f_1) = fnum(f_2)$ dann gibt es einen Algorithmus der sowohl f_1 wie auch f_2 berechnet und so $f_1 = f_2$.
Aber dann ist $\mathcal{F}$ abzählbar. Widerspruch zu Satz 5.7 □

Beispiel 5.4

Ist die Funktion

$$f_4(n) := \begin{cases} 1 & \text{falls mindestens } n \text{ mal hintereinander irgendwo in der Dezimalbruchentwicklung von } \pi \text{ eine 0 vorkommt} \\ 0 & \text{sonst} \end{cases}$$

berechenbar?

Ja, denn

- entweder kommt 0^n für beliebig große n vor, dann ist $f_4(n) = 1$ für alle n,

Beispiel 5.4

Ist die Funktion

$$f_4(n) := \begin{cases} 1 & \text{falls mindestens } n \text{ mal hintereinander irgendwo in der Dezimalbruchentwicklung von } \pi \text{ eine 0 vorkommt} \\ 0 & \text{sonst} \end{cases}$$

berechenbar?

Beispiel 5.4

Ist die Funktion

$$f_4(n) := \begin{cases} 1 & \text{falls mindestens } n \text{ mal hintereinander irgendwo in der Dezimalbruchentwicklung von } \pi \text{ eine 0 vorkommt} \\ 0 & \text{sonst} \end{cases}$$

berechenbar?

Ja, denn

- entweder kommt 0^n für beliebig große n vor, dann ist $f_4(n) = 1$ für alle n,
- oder es gibt eine längste vorkommende Sequenz 0^m, dann ist $f_4(n) = 1$ für $n \leq m$ und $f_4(n) = 0$ sonst.

input (4);
if 4 ≤ m then 1 else 0

Beispiel 5.3

Ist die Funktion

$$f_2(n) := \begin{cases} 1 & \text{falls } n \text{ als Ziffernfolge irgendwo in der Dezimalbruchentwicklung von } \pi \text{ vorkommt} \\ 0 & \text{sonst} \end{cases}$$

berechenbar? (Bsp: $f_2(415) = 1$)

Unbekannt!

Durch schrittweise Approximation und Suche in der Dezimalbruchentwicklung von π kann man feststellen, dass n vorkommt. Aber wie stellt man fest, dass n *nicht* vorkommt? Nichttermination statt 0!

4.10 Deterministische Kellerautomaten

Definition 4.63

Ein PDA heißt deterministisch (DPDA) gdw für alle $q \in Q$, $a \in \Sigma$ und $Z \in \Gamma$

$$|\delta(q, a, Z)| + |\delta(q, \epsilon, Z)| \leq 1$$

Beispiel 4.64

Der folgende DPDA mit $\Sigma = \{0, 1, \$\}$, $\Gamma = \{Z_0, 0, 1\}$ akzeptiert die Sprache $\{w\$w^R \mid w \in \{0, 1\}^*\}$.