Script generated by TTT

Title: seidl: Theoretische_Informatik (31.05.2012)

Date: Thu May 31 15:59:45 CEST 2012

Duration: 94:30 min

Pages: 76

Hauptresultate:

- Kellerautomaten akzeptieren genau die kontextfreien Spachen.

Hauptresultate:

- Kellerautomaten akzeptieren genau die kontextfreien Spachen.
- Nichtdeterministische Kellerautomaten (PDAs) sind mächtiger als deterministische Kellerautomaten (DPDAs).

3.9 Tabellarischer Überblick

Abschlusseigenschaften

	Schnitt	Vereinigung	Komplement	Produkt	Stern

3.9 Tabellarischer Überblick

Abschlusseigenschaften

	Schnitt	Vereinigung	Komplement	Produkt	Stern
Regulär	ja	ja	ja	ja	ja

3.9 Tabellarischer Überblick

Abschlusseigenschaften

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

3.9 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

3.9 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				

3.9 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	$O(n)$			
DPDA	$O(n)$			
CFG	$O(n^3)$			

3.9 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	$O(n)$	ja		
DPDA	$O(n)$	ja		
CFG	$O(n^3)$	ja		

3.9 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	
DFA	$O(n)$	ja	ja	
DPDA	$O(n)$	ja	ja	
CFG	$O(n^3)$	ja	nein(*)	

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

3.9 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

3.9 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	ja
DPDA	$O(n)$	ja	ja	nein(*)
CFG	$O(n^3)$	ja	nein(*)	nein(*)

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

3.10 Die Chomsky-Hierarchie

Definition 3.52
Eine Grammatik G ist ein 4-Tupel (V, Σ, P, S),
wobei P eine endliche Teilmenge von $(V \cup \Sigma)^+ \times (V \cup \Sigma)^*$ ist,

3.10 Die Chomsky-Hierarchie

Definition 3.52
Eine Grammatik G ist ein 4-Tupel (V, Σ, P, S),
wobei P eine endliche Teilmenge von $(V \cup \Sigma)^+ \times (V \cup \Sigma)^*$ ist,

dh jede Produktion ist von der Form $\alpha \hookrightarrow \beta$.

3.10 Die Chomsky-Hierarchie

Definition 3.52
Eine Grammatik G ist ein 4-Tupel (V, Σ, P, S),
wobei P eine endliche Teilmenge von $(V \cup \Sigma)^+ \times (V \cup \Sigma)^+$ ist,
dh jede Produktion ist von der Form $\alpha \to \beta$.
G ist vom

Typ 0

3.10 Die Chomsky-Hierarchie

Definition 3.52
Eine Grammatik G ist ein 4-Tupel (V, Σ, P, S),
wobei P eine endliche Teilmenge von $(V \cup \Sigma)^+ \times (V \cup \Sigma)^+$ ist,
dh jede Produktion ist von der Form $\alpha \to \beta$.
G ist vom

Typ 0

Typ 1 falls $|\alpha| \leq |\beta|$

3.10 Die Chomsky-Hierarchie

Definition 3.52
Eine Grammatik G ist ein 4-Tupel (V, Σ, P, S),
wobei P eine endliche Teilmenge von $(V \cup \Sigma)^+ \times (V \cup \Sigma)^+$ ist,
dh jede Produktion ist von der Form $\alpha \to \beta$.
G ist vom

Typ 0

Typ 1 falls $|\alpha| \leq |\beta|$

Typ 2 falls G vom Typ 1 ist und $\alpha \in V$

3.10 Die Chomsky-Hierarchie

Definition 3.52
Eine Grammatik G ist ein 4-Tupel (V, Σ, P, S),
wobei P eine endliche Teilmenge von $(V \cup \Sigma)^+ \times (V \cup \Sigma)^+$ ist,
dh jede Produktion ist von der Form $\alpha \to \beta$.
G ist vom

Typ 0

Typ 1 falls $|\alpha| \leq |\beta|$

Typ 2 falls G vom Typ 1 ist und $\alpha \in V$

3.10 Die Chomsky-Hierarchie

Definition 3.52
Eine Grammatik G ist ein 4-Tupel (V, Σ, P, S),
wobei P eine endliche Teilmenge von $(V \cup \Sigma)^+ \times (V \cup \Sigma)^+$ ist,
dh jede Produktion ist von der Form $\alpha \to \beta$.
G ist vom

- Typ 0
- Typ 1 falls $|\alpha| \leq |\beta|$
- Typ 2 falls G vom Typ 1 ist und $\alpha \in V$
- Typ 3 falls G vom Typ 2 ist und $\beta \in \Sigma \cup \Sigma V$.

3.10 Die Chomsky-Hierarchie

Definition 3.52
Eine Grammatik G ist ein 4-Tupel (V, Σ, P, S),
wobei P eine endliche Teilmenge von $(V \cup \Sigma)^+ \times (V \cup \Sigma)^+$ ist,
dh jede Produktion ist von der Form $\alpha \to \beta$.
G ist vom

- Typ 0
- Typ 1 falls $|\alpha| \leq |\beta|$
- Typ 2 falls G vom Typ 1 ist und $\alpha \in V$
- Typ 3 falls G vom Typ 2 ist und $\beta \in \Sigma \cup \Sigma V$.

Zusätzlich erlaubt man noch $S \to \epsilon$.

S

3.10 Die Chomsky-Hierarchie

Definition 3.52
Eine Grammatik G ist ein 4-Tupel (V, Σ, P, S),
wobei P eine endliche Teilmenge von $(V \cup \Sigma)^+ \times (V \cup \Sigma)^+$ ist,
dh jede Produktion ist von der Form $\alpha \to \beta$.
G ist vom

- Typ 0
- Typ 1 falls $|\alpha| \leq |\beta|$
- Typ 2 falls G vom Typ 1 ist und $\alpha \in V$
- Typ 3 falls G vom Typ 2 ist und $\beta \in \Sigma \cup \Sigma V$.

Zusätzlich erlaubt man noch $S \to \epsilon$.

falls S nirgends in β vorkommt

3.10 Die Chomsky-Hierarchie

Definition 3.52
Eine Grammatik G ist ein 4-Tupel (V, Σ, P, S),
wobei P eine endliche Teilmenge von $(V \cup \Sigma)^+ \times (V \cup \Sigma)^+$ ist,
dh jede Produktion ist von der Form $\alpha \to \beta$.
G ist vom

- Typ 0
- Typ 1 falls $|\alpha| \leq |\beta|$
- Typ 2 falls G vom Typ 1 ist und $\alpha \in V$
- Typ 3 falls G vom Typ 2 ist und $\beta \in \Sigma \cup \Sigma V$.

Zusätzlich erlaubt man noch $S \to \epsilon$.

Offensichtlich gilt:

$$\text{Typ 3} \subset \text{Typ 2} \subset \text{Typ 1} \subset \text{Typ 0}$$

Grammatiken, Maschinen und Spachklassen:

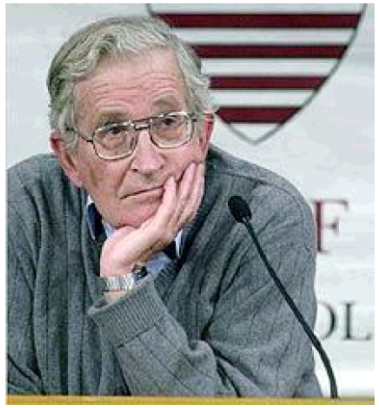
Typ 3	Rechtslineare Grammatik Endlicher Automat
Typ 2	kontextfreie Grammatik Kellerautomat
Typ 1	Kontextsensitive Grammatik Linear beschränkter Automat
Typ 0	Chomsky-Grammatik, Phrasenstrukturgrammatik Turingmaschine

Grammatiken, Maschinen und Spachklassen:

Typ 3	Rechtslineare Grammatik Endlicher Automat
Typ 2	kontextfreie Grammatik Kellerautomat
Typ 1	Kontextsensitive Grammatik Linear beschränkter Automat
Typ 0	Chomsky-Grammatik, Phrasenstrukturgrammatik Turingmaschine

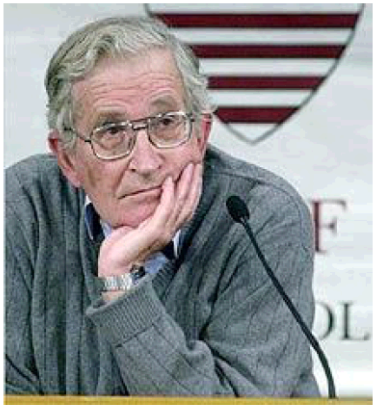
Satz 3.53
$L(\text{Typ } 3) \subset L(\text{Typ } 2) \subset L(\text{Typ } 1) \subset L(\text{Typ } 0)$

Noam Chomsky.
Three Models for the Description of Language. Transactions on Information Theory, 113-124, 1956.



Noam Chomsky (* 7. Dezember 1928) ist Professor für Linguistik am MIT und ein bedeutender Sprachwissenschaftler. Er entwickelte die Chomsky-Hierarchie.

Noam Chomsky.
Three Models for the Description of Language. Transactions on Information Theory, 113-124, 1956.



Noam Chomsky (* 7. Dezember 1928) ist Professor für Linguistik am MIT und ein bedeutender Sprachwissenschaftler. Er entwickelte die Chomsky-Hierarchie.
Neben seiner linguistischen Arbeit gilt Chomsky als einer der bedeutendsten Intellektuellen Nordamerikas und ist als scharfer Kritiker der US-amerikanischen Außenpolitik bekannt.

Kapitel II
Berechenbarkeit, Entscheidbarkeit, Komplexität

Kapitel II
Berechenbarkeit, Entscheidbarkeit, Komplexität

Überblick:

- Was kann man berechnen?

Kapitel II
Berechenbarkeit, Entscheidbarkeit, Komplexität

Überblick:

- Was kann man berechnen? Dh:
  Welche Funktionen kann man in endlicher Zeit berechnen?

Kapitel II
Berechenbarkeit, Entscheidbarkeit, Komplexität

Überblick:

- Was kann man berechnen? Dh:
  Welche Funktionen kann man in endlicher Zeit berechnen?
- Mit welchen Sprachen/Maschinen?
- Welche Eigenschaften von Programmen sind entscheidbar?

Kapitel II
Berechenbarkeit, Entscheidbarkeit, Komplexität

Überblick:

- Was kann man berechnen? Dh:
  Welche Funktionen kann man in endlicher Zeit berechnen?
- Mit welchen Sprachen/Maschinen?
- Welche Eigenschaften von Programmen sind entscheidbar?
  ZB Termination? Code-Erreichbarkeit? Überlauf?

Kapitel II
Berechenbarkeit, Entscheidbarkeit, Komplexität

Überblick:

- Was kann man berechnen? Dh:
  Welche Funktionen kann man in endlicher Zeit berechnen?
- Mit welchen Sprachen/Maschinen?
- Welche Eigenschaften von Programmen sind entscheidbar?
  ZB Termination? Code-Erreichbarkeit? Überlauf?

- Was kann man in polynomieller Zeit berechnen?

Kapitel II
Berechenbarkeit, Entscheidbarkeit, Komplexität

Überblick:

- Was kann man berechnen? Dh:
  Welche Funktionen kann man in endlicher Zeit berechnen?
- Mit welchen Sprachen/Maschinen?
- Welche Eigenschaften von Programmen sind entscheidbar?
  ZB Termination? Code-Erreichbarkeit? Überlauf?
- Was kann man in polynomieller Zeit berechnen?

  Mit welchen Maschinen?

4. Berechenbarkeit und Entscheidbarkeit

4.1 Der Begriff der Berechenbarkeit

Kapitel II
Berechenbarkeit, Entscheidbarkeit, Komplexität

Überblick:

- Was kann man berechnen? Dh:
  Welche Funktionen kann man in endlicher Zeit berechnen?
- Mit welchen Sprachen/Maschinen?
- Welche Eigenschaften von Programmen sind entscheidbar?
  ZB Termination? Code-Erreichbarkeit? Überlauf?
- Was kann man in polynomieller Zeit berechnen?
  Mit welchen Maschinen?

4. Berechenbarkeit und Entscheidbarkeit

4.1 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?

4. Berechenbarkeit und Entscheidbarkeit

4.1 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?

4. Berechenbarkeit und Entscheidbarkeit

4.1 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.

4. Berechenbarkeit und Entscheidbarkeit

4.1 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.

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 4.1

Der Algorithmus

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

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 4.1

Der Algorithmus

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

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

Beispiel 4.2

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 4.2

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 π kann iterativ auf beliebig viele Dezimalstellen genau berechnet werden.

Beispiel 4.3

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 4.3

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 4.3

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 4.3

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!

Beispiel 4.3

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 4.4

Ist die Funktion

$$f_3(n) := \begin{cases} 1 & \text{falls mindestens } n \text{ mal hintereinander irgendwo in der Dezimalbruchentwicklung von } \pi \text{ eine 0 vorkommt} \\ 0 & \text{sonst} \end{cases}$$

berechenbar?

Beispiel 4.4

Ist die Funktion

$$f_3(n) := \begin{cases} 1 & \text{falls mindestens } n \text{ mal hintereinander irgendwo in der Dezimalbruchentwicklung von } \pi \text{ eine 0 vorkommt} \\ 0 & \text{sonst} \end{cases}$$

berechenbar?

Ja, denn

- entweder kommt 0^n für beliebig große n vor,

Beispiel 4.4

Ist die Funktion

$$f_3(n) := \begin{cases} 1 & \text{falls mindestens } n \text{ mal hintereinander irgendwo in der Dezimalbruchentwicklung von } \pi \text{ eine 0 vorkommt} \\ 0 & \text{sonst} \end{cases}$$

berechenbar?

Ja, denn

- entweder kommt 0^n für beliebig große n vor,
  dann ist $f_3(n) = 1$ für alle n,
- oder es gibt eine längste vorkommende Sequenz 0^m,

Beispiel 4.4

Ist die Funktion

$$f_3(n) := \begin{cases} 1 & \text{falls mindestens } n \text{ mal hintereinander irgendwo in der Dezimalbruchentwicklung von } \pi \text{ eine 0 vorkommt} \\ 0 & \text{sonst} \end{cases}$$

berechenbar?

Ja, denn

- entweder kommt 0^n für beliebig große n vor,
  dann ist $f_3(n) = 1$ für alle n,
- oder es gibt eine längste vorkommende Sequenz 0^m,
  dann ist $f_3(n) = 1$ für $n \leq m$ und $f_3(n) = 0$ sonst.

Beispiel 4.4

Ist die Funktion

$$f_3(n) := \begin{cases} 1 & \text{falls mindestens } n \text{ mal hintereinander irgendwo in der Dezimalbruchentwicklung von } \pi \text{ eine 0 vorkommt} \\ 0 & \text{sonst} \end{cases}$$

berechenbar?

Ja, denn

- entweder kommt 0^n für beliebig große n vor,
  dann ist $f_3(n) = 1$ für alle n,
- oder es gibt eine längste vorkommende Sequenz 0^m,
  dann ist $f_3(n) = 1$ für $n \leq m$ und $f_3(n) = 0$ sonst.

Beide Funktionen sind berechenbar.

Dies ist ein nicht-konstruktiver Beweis:

Satz 4.5

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

Satz 4.5

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

Beweis:

Es gibt nur abzählbar viele Algorithmen, aber überabzählbar viele Funktionen in $\mathbb{N} \to \{0, 1\}$ (Beweis wie zu Satz 1.8).

□

Verschiedene Formalisierungen des Begriffs der Berechenbarkeit:

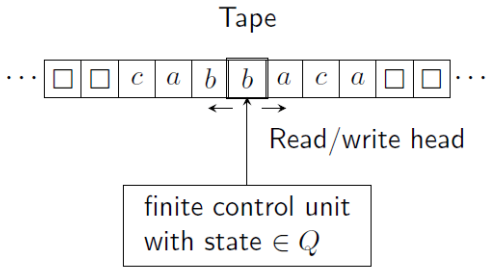
- Turing-Maschinen (Turing 1936)
- λ-Kalkül (Church 1936)
- μ-rekursive Funktionen
- Markov-Algorithmen
- Registermaschinen
- Awk, Basic, C, Dylan, Eiffel, Fortran, Java, Lisp, Modula, Oberon, Pascal, Python, Ruby, Simula, TEX, . . . -Programme
- while-Programme
- goto-Programme
- DNA-Algorithmen
- Quantenalgorithmen
- . . .

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

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

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

4.2 Turingmaschinen



Definition 4.6
Eine Turingmaschine (TM) ist ein 7-Tupel $M = (Q, \Sigma, \Gamma, \delta, q_0, \square, F)$ so dass

- Q ist eine endliche Menge von Zuständen.

Definition 4.6
Eine Turingmaschine (TM) ist ein 7-Tupel
$M = (Q, \Sigma, \Gamma, \delta, q_0, \square, F)$ so dass

- Q ist eine endliche Menge von Zuständen.
- Σ ist eine endliche Menge, das Eingabealphabet.
- Γ ist eine endliche Menge, das Bandalphabet, mit $\Sigma \subset \Gamma$

Definition 4.6
Eine Turingmaschine (TM) ist ein 7-Tupel
$M = (Q, \Sigma, \Gamma, \delta, q_0, \square, F)$ so dass

- Q ist eine endliche Menge von Zuständen.
- Σ ist eine endliche Menge, das Eingabealphabet.
- Γ ist eine endliche Menge, das Bandalphabet, mit $\Sigma \subset \Gamma$
- $\delta : Q \times \Gamma \to Q \times \Gamma \times \{L, R, N\}$ ist die Übergangsfunktion.

Definition 4.6
Eine Turingmaschine (TM) ist ein 7-Tupel
$M = (Q, \Sigma, \Gamma, \delta, q_0, \square, F)$ so dass

- Q ist eine endliche Menge von Zuständen.
- Σ ist eine endliche Menge, das Eingabealphabet.
- Γ ist eine endliche Menge, das Bandalphabet, mit $\Sigma \subset \Gamma$
- $\delta : Q \times \Gamma \to Q \times \Gamma \times \{L, R, N\}$ ist die Übergangsfunktion.
- δ darf partielle sein.

Definition 4.6
Eine Turingmaschine (TM) ist ein 7-Tupel
$M = (Q, \Sigma, \Gamma, \delta, q_0, \square, F)$ so dass

- Q ist eine endliche Menge von Zuständen.
- Σ ist eine endliche Menge, das Eingabealphabet.
- Γ ist eine endliche Menge, das Bandalphabet, mit $\Sigma \subset \Gamma$
- $\delta : Q \times \Gamma \to Q \times \Gamma \times \{L, R, N\}$ ist die Übergangsfunktion.
- δ darf partielle sein.
- $q_0 \in Q$ ist der Startzustand.
- $\square \in \Gamma \setminus \Sigma$ ist das Leerzeichen.

Definition 4.6

Eine Turingmaschine (TM) ist ein 7-Tupel $M = (Q, \Sigma, \Gamma, \delta, q_0, \square, F)$ so dass

- Q ist eine endliche Menge von Zuständen.
- Σ ist eine endliche Menge, das Eingabealphabet.
- Γ ist eine endliche Menge, das Bandalphabet, mit $\Sigma \subset \Gamma$
- $\delta : Q \times \Gamma \to Q \times \Gamma \times \{L, R, N\}$ ist die Übergangsfunktion.
- δ darf partielle sein.
- $q_0 \in Q$ ist der Startzustand.
- $\square \in \Gamma \setminus \Sigma$ ist das Leerzeichen.
- $F \subseteq Q$ ist die Menge der akzeptierenden oder Endzustände.

Definition 4.6

Eine Turingmaschine (TM) ist ein 7-Tupel $M = (Q, \Sigma, \Gamma, \delta, q_0, \square, F)$ so dass

- Q ist eine endliche Menge von Zuständen.
- Σ ist eine endliche Menge, das Eingabealphabet.
- Γ ist eine endliche Menge, das Bandalphabet, mit $\Sigma \subset \Gamma$
- $\delta : Q \times \Gamma \to Q \times \Gamma \times \{L, R, N\}$ ist die Übergangsfunktion.
- δ darf partielle sein.
- $q_0 \in Q$ ist der Startzustand.
- $\square \in \Gamma \setminus \Sigma$ ist das Leerzeichen.
- $F \subseteq Q$ ist die Menge der akzeptierenden oder Endzustände.

Eine nichtdeterministische Turingmaschine hat eine Übergangsfunktion $\delta : Q \times \Gamma \to \mathcal{P}(Q \times \Gamma \times \{L, R, N\})$.

Intuitiv bedeutet $\delta(q, a) = (q', b, d)$:

Intuitiv bedeutet $\delta(q, a) = (q', b, d)$:

- Wenn sich M im Zustand q befindet,
- und auf dem Band a liest,
- so geht M im nächsten Schritt in den Zustand q' über,
- überschreibt a mit b,

Intuitiv bedeutet $\delta(q, a) = (q', b, d)$:

- Wenn sich M im Zustand q befindet,
- und auf dem Band a liest,
- so geht M im nächsten Schritt in den Zustand q' über,
- überschreibt a mit b,
- und bewegt danach den Schreib-/Lesekopf nach rechts (falls $d = R$), nach links (falls $d = L$), oder nicht (falls $d = N$).

Definition 4.7
Eine Konfiguration einer Turingmaschine ist ein Tripel $(\alpha, q, \beta) \in \Gamma^* \times Q \times \Gamma^*$.

Definition 4.7
Eine Konfiguration einer Turingmaschine ist ein Tripel $(\alpha, q, \beta) \in \Gamma^* \times Q \times \Gamma^*$.

Dies modelliert

- Bandinhalt: $\ldots \square\alpha\beta\square \ldots$
- Zustand: q
- Kopf auf dem ersten Zeichen von $\beta\square$

Definition 4.7
Eine Konfiguration einer Turingmaschine ist ein Tripel $(\alpha, q, \beta) \in \Gamma^* \times Q \times \Gamma^*$.

Dies modelliert

- Bandinhalt: $\ldots \square\alpha\beta\square \ldots$
- Zustand: q
- Kopf auf dem ersten Zeichen von $\beta\square$

Die Startkonfiguration der Turingmaschine bei Eingabe $w \in \Sigma^*$ ist (ϵ, q_0, w).

Die Berechnung der TM M wird als Relation $\rightarrow_M$ auf Konfigurationen formalisiert. Falls $\delta(q, \mathit{first}(\beta)) = (q', c, d)$:

$(\alpha, q, \beta) \rightarrow_M$