Grundlagen der Programmierung (Vorlesung 23)

Ralf Möller, FH-Wedel

- Vorige Vorlesung
  - Anwendung im Bereich Compilerbau
- Inhalt dieser Vorlesung
  - Turing-Maschinen
  - Berechenbarkeitstheorie, Halteproblem
- Lernziele
  - Kennenlernen der Grenzen von Berechnungen durch Algorithmen

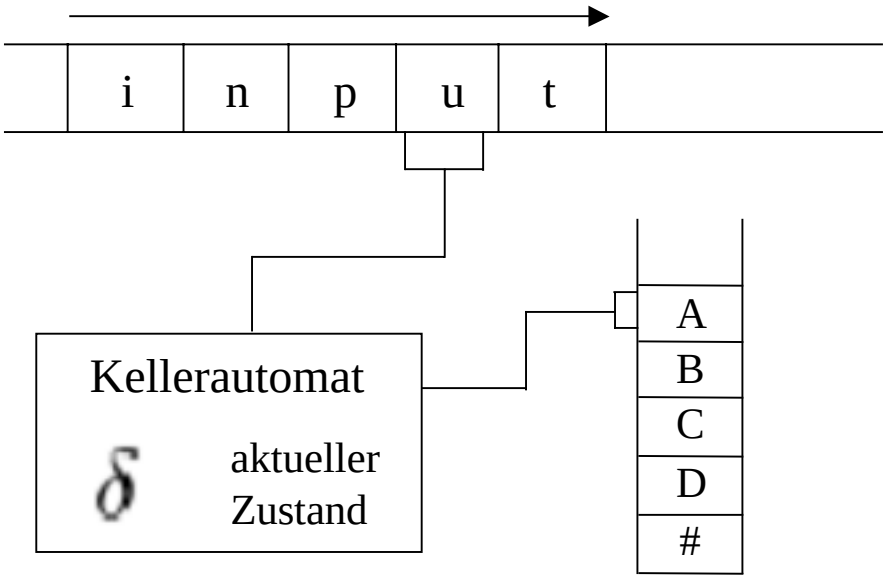
Danksagung

- Teile der Präsentationen sind an den Inhalt des Buches "Theoretische Informatik kurzgefaßt" von Uwe Schöning angeleht und wurden aus den Unterlagen zu der Vorlesung "Informatik IV – Theoretische Informatik" an der TU München von Angelika Steger übernommen
- Die übernommenen Teile sind in den Originalunterlagen zu finden unter:
  http://www14.in.tum.de/lehre/2000SS/info4/

Automat für Typ-2-Sprachen

Definition. Ein *nichtdeterministischer Kellerautomat* (englisch: pushdown automata, kurz PDA) wird durch ein 6-Tupel $M = (Z, \Sigma, \Gamma, \delta, z_0, \#)$ beschrieben, das folgende Bedingungen erfüllt:

- Z ist eine endliche Menge von *Zuständen*.
- Σ ist eine endliche Menge, das *Eingabealphabet*.
- Γ ist eine endliche Menge, das *Kelleralphabet*.
- $\delta : Z \times (\Sigma \cup \{\varepsilon\}) \times \Gamma \to \langle$endliche Teilmengen von $Z \times \Gamma^*\rangle$, die *Übergangsfunktion*.
- $z_0 \in Z$ ist der *Startzustand*.
- $\# \in \Gamma$ ist das *unterste Kellerzeichen*.



Beispiel: $S \to ab \mid aSb$ kontextfreie Grammatik

Konstruktion des Kellerautomaten:

- nur ein Zustand: z
- Kelleralphabet: $\{S, a, b\}$
- unterstes Kellerzeichen: S
- Definition von δ:
  Die Regel $S \to ab$ führt zu

$$\delta(z, \epsilon, S) \ni (z, ab). \qquad (1)$$

  Die Regel $S \to aSb$ führt zu

$$\delta(z, \epsilon, S) \ni (z, aSb). \qquad (2)$$

  Zusätzlich noch:

$$\delta(z, a, a) \ni (z, \varepsilon), \qquad \delta(z, b, b) \ni (z, \varepsilon). \qquad (3)$$

Arbeitsweise bei Erkennung des Wortes $aabb$:

	ungelesene Zeichen der Eingabe	Kellerinhalt
Startzustand:	$aabb$	S
Regel (2):	$aabb$	aSb
Regel (3):	abb	Sb
Regel (1):	abb	abb
Regel (3):	bb	bb
Regel (3):	b	b
Regel (3):	ε	ε

1 $\delta(z, \epsilon, S) \ni (z, ab)$

2 $\delta(z, \epsilon, S) \ni (z, aSb)$

3 $\delta(z, a, a) \ni (z, \varepsilon)$

$\delta(z, b, b) \ni (z, \varepsilon)$

Deterministische Kellerautomaten

Definition: *deterministischer Kellerautomat*

Ein nichtdeterministischer Kellerautomat heißt *deterministisch*, falls gilt:

Für alle $z \in Z, a \in \Sigma, A \in \Gamma$ ist

$$|\delta(z, a, A)| + |\delta(z, \varepsilon, A)| \leq 1.$$

Erläuterung: Dies bedeutet, daß der Automat zu jedem Zeitpunkt höchstens eine Alternative hat. Daher der Name *deterministisch*.

Um die von einem deterministischen Kellerautomaten M akzeptierte Sprache $L(M)$ zu definieren, müssen wir $M = (Z, \Sigma, \Gamma, \delta, z_0, \#)$ noch um eine Menge $E \subseteq Z$ von *Endzuständen* erweitern. $L(M)$ besteht dann aus genau den Worten, für die es eine Folge von Übergängen gibt, so daß sich M nach Abarbeiten des Wortes in einem Endzustand befindet.

Definition. Eine kontextfreie Grammatik G heißt *deterministisch kontextfrei*, falls es einen deterministischen Kellerautomaten M gibt mit $L(G) = L(M)$.

Wortproblem für deterministisch-kontextfreie Sprachen

- Das Wortproblem für deterministisch-kontextfreie Sprachen kann in linearer Zeit gelöst werden
- Leider gibt es nicht für jede kontextfreie Sprache auch einen deterministischen Kellerautomaten (DPDA)
- Aber: Wenn man zu einer Typ-2-Grammatik einen DPDA findet, ist man in Anwendungen im Vorteil

Komplexität von Algorithmen zur Lösung des Wortproblems

Wortproblem	
Typ 3, gegeben als DFA	lineare Laufzeit
Det. kf, gegeben als DPDA	lineare Laufzeit
Typ 2, gegeben durch Grammatik in CNF	CYK-Algorithmus, Laufzeit $\mathcal{O}(n^3)$
Typ 1	exponentiell
Typ 0	⟨nächstes Kapitel⟩

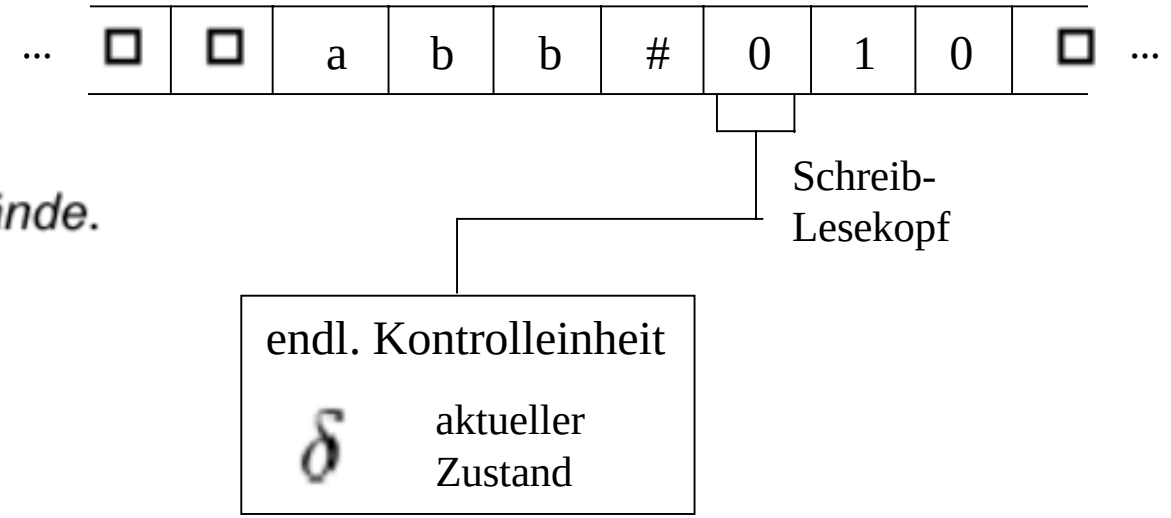
Auf dem Weg zu Maschinen für Typ-0-Sprachen

- Wesentliche Beschränkung des Kellerautomaten: Zugriffsmöglichkeit auf seinen Speicher
- Nur "Last-in-first-out"-Prinzip
- A.M. Turing schlägt (um 1940) ein Automatenmodell vor, daß nur wenig "berechnungsstärker" als der Kellerautomat ist
- Das Berechnungsmodell heißt heute Turing-Maschine

Turing-Maschine

Definition. Eine *nichtdeterministische Turingmaschine* (kurz TM) wird durch ein 7-Tupel $M = (Z, \Sigma, \Gamma, \delta, z_0, \square, E)$ beschrieben, das folgende Bedingungen erfüllt:

- Z ist eine endliche Menge von *Zuständen*.
- Σ ist eine endliche Menge, das *Eingabealphabet*.
- Γ ist eine endliche Menge, das *Bandalphabet*.
- $\delta : Z \times \Gamma \to \mathcal{P}(Z \times \Gamma \times \{L, R, N\})$, die *Übergangsfunktion*.
- $z_0 \in Z$ ist der *Startzustand*.
- $\square \in \Gamma \setminus \Sigma$, das *Leerzeichen*.
- $E \subseteq Z$, die Menge der *Endzustände*.



Deterministische Turing-Maschinen

Die Turingmaschine heißt *deterministisch*, falls gilt

$$|\delta(z,a)| = 1 \qquad \text{für alle } z \in Z, a \in \Gamma.$$

Erläuterung: Intuitiv bedeutet $\delta(z,a) = (z',b,x)$ bzw. $\delta(z,a) \ni (z',b,x)$:

Wenn sich M im Zustand z befindet und unter dem Schreib-/Lesekopf das Zeichen a steht, so geht M im nächsten Schritt in den Zustand z' über, schreibt an die Stelle des a's das Zeichen b und bewegt danach den Schreib-/Lesekopf um eine Position nach rechts (falls $x = R$), links (falls $x = L$) oder läßt ihn unverändert (falls $x = N$).

Definition. Eine Turingmaschine heißt *linear beschränkt* (kurz: LBA), falls für alle $z \in \mathbb{Z}$ gilt:

$$(z', a) \in \delta(z, \square) \quad \rightarrow \quad a = \square.$$

D.h., ein Leerzeichen wird nie durch ein anderes Zeichen überschrieben. Mit anderen Worten: Die Turingmaschine darf ausschliesslich die Positionen beschreiben, an denen zu Beginn die Eingabe x stand.

Satz. Die von linear beschränkten, nichtdeterministischen Turingmaschinen akzeptierten Sprachen sind genau die kontextsensitiven Sprachen.

Bew: siehe Literatur.

Analog kann man zeigen:

Satz. Die von Turingmaschinen akzeptierten Sprachen sind genau die Typ 0 Sprachen.

Hinweis. Hier ist es egal, ob man die Turingmaschine deterministisch oder nichtdeterministisch wählt. Genauer gilt: Zu jeder nichtdeterministischen Turingmaschine M gibt es eine deterministische Turingmaschine M' mit $L(M) = L(M')$.

Chomsky-Hierarchie

Typ 3	reguläre Grammatik DFA NFA regulärer Ausdruck
Det. kf.	$LR(k)$-Grammatik deterministischer Kellerautomat
Typ 2	kontextfreie Grammatik (nichtdeterministischer) Kellerautomat
Typ 1	kontextsensitive Grammatik (nichtdeterministische) linear beschränkte Turingmaschine
Typ 0	endliche Grammatik Turingmaschine

Turings Intention war viel weitreichender ...

- Angabe einer mathematisch klar beschreibbaren Maschine, die stellvertretend für *jeden beliebigen* algorithmischen Berechnungsprozeß stehen kann
- Turings Vorstellung war es, mit der Turing-Maschine den (zunächst nur intuitiv gegebenen) Begriff der *Berechenbarkeit*, des *effektiven Verfahrens* exakt beschrieben zu haben
- Man ist heute davon überzeugt, daß ihm dieses geglückt ist

Berechenbarkeitstheorie

- Bisher: Turing-Maschinen zur Definition (oder Akzeptierung) von Sprachen
- Jetzt: Turing-Maschinen zur Berechnung einer Funktion
- Eine Funktion ist darstellbar durch eine Menge von Tupeln
- Eine Turing-Maschine berechnet eine Funktion, wenn sie einen Sprache definiert, deren Worte den Tupeln der Funktion entsprechen
- Eine Funktion f heißt (Turing-)berechenbar, wenn es eine Turing-Maschine T gibt, deren akzeptierte Sprache den Tupeln der Funktion f entsprechen
- Zu einer Funktion f sei TM(f) die Turing-Maschine, die f berechnet

Aber ...

- ... wir haben doch in dieser Vorlesung schon eine (fiktive) Sprache zur Formulierung von Algorithmen kennengelernt mit Zuweisungen, Fallunterscheidungen, und Schleifen
- Eine solche Sprache heißt WHILE
- Man kann zeigen, daß man für jedes WHILE-Programm auch eine Turing-Maschine definieren kann, die die gleiche Funktion berechnet
- Auch andere Versuche, einen Algorithmusbegriff zu definieren, führten zu nichts Neuem

Church-Turing-These

- Alle vernünftigen Definitionen von "Algorithmus", soweit sie bekannt sind, sind gleichwertig und gleichbedeutend
- Jede vernünftige Definition von "Algorithmus", die jemals irgendwer aufstellt, ist gleichwertig und gleichbedeutend zur Definition von Algorithmen mit Turing-Maschinen
- Dieses ist nur eine Annahme (These), die man nicht beweisen kann, aber eine Annahme, die noch niemand widerlegen konnte!

Entscheidbarkeit

- Ein Problem mit Eingabe D heißt *entscheidbar*, wenn es einen Algorithmus P gibt, der vollständig und korrekt ist und auf jeder Eingabe D hält

Eine Universelle Turing-Maschine ist ...

- ... eine Turing-Maschine, die als Eingabe eine kodierte Turing-Maschine und ein Eingabewort für die Eingabe-Turing-Maschine erhält
- Eine solche Turing-Maschine heißt *Universelle Turing-Maschine* und interpretiert die Eingabe-Turing-Maschine, d.h. sie simuliert deren Verhalten bzgl. des Eingabeworts
- Wir nennen die Universelle Turing-Maschine U
- Praktische Variante: Simulation eines Rechners (inkl. Betriebssystem) durch einen anderen

Entscheidbarkeit und das Halteproblem

- Terminiert eine beliebige Turing-Maschine auf jeder möglichen Eingabe? Stop-Tester(T, w) : B
- Dieses Problem ist nicht entscheidbar, d.h. es gibt keinen Algorithmus für Stop-Tester, der vollständig und korrekt ist und bezüglich jeder Eingabe T, w hält.
- Doch wie zeigt man, daß es keinen Algorithmus gibt?
- Beweistechnik: Annahme eines Algorithmus und Herleiten eines Widerspruches aus der Annahme

Stop-Tester etwas genauer betrachtet...

- Stop-Tester(T, w) : B
  if "T hält mit Eingabe w"
  then true
  else false
  end if

Beweis (1)

- Annahme: Stop-Tester(T, w) : B ist berechenbar für alle T und w
- Dann muß auch folgende Funktion berechenbar sein:
- Stop-Tester-1(T) : B
  Stop-Tester(U, T/T)
- Verwendung von Stop-Tester-1:

```
f(T) : B
  if Stop-Tester-1(T)
     then while true do end while; false
     else true
  end if
```

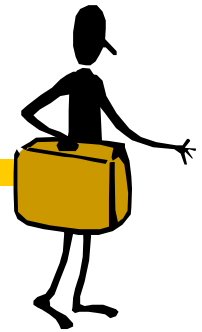
Beweis (2)

- Wir betrachten folgenden Aufruf: f(TM(f))
- Stop-Tester-1 bestimmt, ob TM(f) hält.
  - Annahme: Stop-Tester-1(TM(f)) liefert true
    - Dann hält f nicht (und damit TM(f) auch nicht)
  - Annahme: Stop-Tester-1(TM(f)) liefert false
    - Dann hält f (und damit TM(f) auch)
  - Widerspruch: Die Annahme, Stop-Tester(T, w) : B ist berechenbar für alle T und w muß falsch sein.

Konsequenz

- Nicht alle formal eindeutig gegebenen Probleme sind entscheidbar
- Und damit: Es gibt keine Algorithmus zur Entwicklung von Algorithmen

Zusammenfassung, Kernpunkte



- Turing-Maschinen
- Chomsky-Hierarchie
- Berechenbarkeitstheorie
- Entscheidbarkeit

Was kommt beim nächsten Mal?

- Fortsetzung Berechenbarkeitstheorie
- Genauere Analyse entscheidbarer aber "schwieriger" Probleme