TECHNISCHE UNIVERSITÄT DRESDEN

Fakultät Informatik, Institut für Künstliche Intelligenz, Professur für Computational Logic

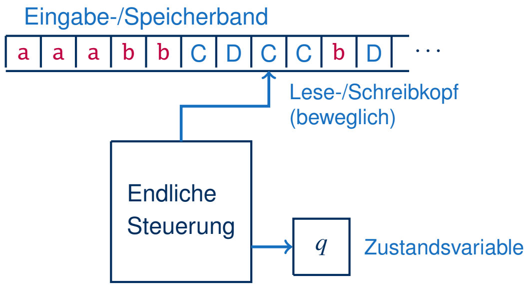
FORMALE SYSTEME

20. Vorlesung: Typ 0 und Typ 1

Hannes Straß

TU Dresden, 16. Dezember 2021

Die Turingmaschine



Eine (deterministische) Turingmaschine (DTM) ist ein Tupel $\mathcal{M} = \langle Q, \Sigma, \Gamma, \delta, q_0, F\rangle$ bestehend aus Zustandsmenge Q, Eingabealphabet Σ, Arbeitsalphabet $\Gamma \supseteq \Sigma \cup \{\llcorner\!\lrcorner\}$, Startzustand $q_0 \in Q$, Endzuständen $F \subseteq Q$, und einer partiellen Übergangsfunktion

$$\delta : Q \times \Gamma \to Q \times \Gamma \times \{L, R, N\}.$$

Turing-Mächtigkeit

- Die Turingmaschine ist das mächtigste bekannte Berechnungsmodell.
  $\rightsquigarrow$ Was nicht Turing-berechenbar ist, gilt als unberechenbar.
- Zahlreiche andere Modelle sind ebenso Turing-mächtig:
  - Turingmaschinen in vielen Varianten (deterministisch/nichtdeterministisch, Einband/Mehrband, einseitig/zweiseitig unendlich, ...)
  - alle „echten“ Programmiersprachen (C, Java, PHP, C++, Python, JavaScript, BASIC, Perl, Pascal, Fortran, C♯, COBOL, Ruby, Visual Basic, Lisp, Ada, Assembler, Lua, Ajax, Prolog, Haskell, R, Go, ALGOL, Scratch, ASP.NET, Scheme, Logo, ASP, TeX, APL, awk, Objective-C, Embarcadero Delphi, Smalltalk, D, Swift, Tcl, Scala, Mathematica, ActionScript, Brainfuck, XSLT, Rust, PostScript, VHDL, GNU Octave, Kotlin, Visual Basic Script, Turbo Pascal, Eiffel, ...*)
  - theoretische Kalküle (Prädikatenlogik erster Stufe, λ-Kalkül, allgemeine rekursive Funktionen, ...)
  - manch Unerwartetes (C++-Templates, SQL, Java Generics, Magic: The Gathering, MS Powerpoint, ...)

* populäre Programmiersprachen (geordnet nach Zahl internationaler Wikipedia-Artikel)

Entscheidbarkeit

Das Halteproblem ist das Wortproblem für die Sprache

$$\{\text{enc}(\mathcal{M})\#\#\text{enc}(w) \mid \mathcal{M} \text{ hält bei Eingabe } w\}.$$

- Entscheidbar: Sprache wird von einem Turing-Entscheider erkannt.
- Unentscheidbar: Sprache wird von keinem Turing-Entscheider erkannt.
- Semi-entscheidbar: Sprache wird von einer TM erkannt, die aber eventuell kein Entscheider ist.

Beispiel:
- Die Sprache $\{ww \mid w \in \{a, b\}^*\}$ ist entscheidbar (und damit auch semi-entscheidbar).
- Das Halteproblem ist nicht entscheidbar, aber semi-entscheidbar.
- Das Komplement des Halteproblems ist nicht semi-entscheidbar (und damit auch nicht entscheidbar).

Der Satz von Rice

Ein interessantes Resultat von Henry Gordon Rice zeigt die Probleme Turing-mächtiger Formalismen:

Satz von Rice (informelle Version): Jede nichttriviale Frage über die von einer TM ausgeführte Berechnung ist unentscheidbar.

Satz von Rice (formell): Sei E eine Eigenschaft von Sprachen, die für manche Turing-erkennbare Sprachen gilt und für manche Turing-erkennbare Sprachen nicht gilt (eine „nichttriviale Eigenschaft"). Dann ist das folgende Problem unentscheidbar:

- Eingabe: Turingmaschine $\mathcal{M}$
- Ausgabe: Hat $\mathbf{L}(\mathcal{M})$ die Eigenschaft E?

Beweis: Durch eine nicht sonderlich komplizierte Reduktion auf das Halteproblem. Kein Vorlesungsstoff.

Alles unentscheidbar

Beispiele für Fragen, die nach dem Satz von Rice unentscheidbar sind:

- „Ist $\mathtt{aba} \in \mathbf{L}(\mathcal{M})$?"
- „Ist $\mathbf{L}(\mathcal{M})$ leer?"
- „Ist $\mathbf{L}(\mathcal{M})$ endlich?"
- „Ist $\mathbf{L}(\mathcal{M})$ regulär?"
- ...

Der Satz von Rice ist dagegen nicht anwendbar auf:

- „Hat $\mathcal{M}$ mindestens zwei Zustände?"
  (keine Eigenschaft von $\mathbf{L}(\mathcal{M})$)
- „Ist $\mathbf{L}(\mathcal{M})$ semi-entscheibar?" (trivial)
- ...

Der Satz von Rice lässt sich sinngemäß auf alle Turing-mächtigen Formalismen übertragen.

Quiz: Satz von Rice

Satz von Rice: Sei E eine Eigenschaft von Sprachen, die für manche Turing-erkennbare Sprachen gilt und für manche Turing-erkennbare Sprachen nicht gilt (eine „nichttriviale Eigenschaft"). Dann ist das folgende Problem unentscheidbar:

- Eingabe: Turingmaschine $\mathcal{M}$
- Ausgabe: Hat $\mathbf{L}(\mathcal{M})$ die Eigenschaft E?

Quiz: Welche der folgenden Probleme könnten Sie mit Hilfe des Satzes von Rice als unentscheidbar beweisen?
(Eingabe ist jeweils eine Turingmaschine $\mathcal{M} = \langle Q, \Sigma, \Gamma, \delta, q_0, F\rangle$.) ...

Typ-0-Sprachen

Typ-0-Grammatiken und Turingmaschinen

Turingmaschinen charakterisieren die Typ-0-Sprachen:

> **Satz:** Die Typ-0-Grammatiken erzeugen genau diejenigen Sprachen, die von einer Turingmaschine erkannt werden können.

Direkte Konsequenzen:

- Typ-0-Grammatiken sind ein universelles (Turing-mächtiges) Berechnungsmodell.
- Typ-0-Sprachen sind die größte Klasse von Sprachen, die wir mit einem „implementierbaren“ Formalismus beschreiben können.
- Die Typ-0-Sprachen sind genau die semi-entscheidbaren Sprachen.
- Das Wortproblem für Typ-0-Sprachen ist unentscheidbar.

Typ 0 $\Leftrightarrow$ TM

> **Satz:** Die Typ-0-Grammatiken erzeugen genau diejenigen Sprachen, die von einer Turingmaschine erkannt werden können.

Beweis: Wir zeigen die beiden Richtungen einzeln:

(1) Wenn eine Sprache von einer Typ-0-Grammatik erzeugt wird, dann kann sie von einer TM erkannt werden.

(2) Wenn eine Sprache von einer TM erkannt wird, dann kann sie durch eine Typ-0-Grammatik erzeugt werden.

Typ 0 $\Rightarrow$ TM

Gegeben: Eine Grammatik G

Gesucht: Eine TM $\mathcal{M}$ mit $\mathbf{L}(\mathcal{M}) = \mathbf{L}(G)$

Idee:

- Turingmaschinen können Ableitungsregeln anwenden.
- Bandinhalt: Zwischenstand der Ableitung (aus Terminalen und Nichtterminalen)
- Ableitungsregel wird nichtdeterministisch gewählt.
- TMs beginnen mit dem von der Grammatik erzeugten Wort.
  $\rightsquigarrow$ Ableitungsregeln werden rückwärts angewendet.

Typ 0 $\Rightarrow$ TM (Details)

Die TM für Grammatik $G = \langle V, \Sigma, P, S\rangle$ arbeitet wie folgt:

- Eingabealphabet Σ
- Bandalphabet $\Gamma = \Sigma \cup V \cup \{\llcorner\!\lrcorner\}$
- Arbeitsweise:
  (1) Wähle (nichtdeterministisch) eine Regel $u \to v \in P$ aus.
  (2) Finde (nichtdeterministisch) auf dem Band ein Vorkommen von v.
  (3) Ersetze das gewählte v durch u (dabei muss der restliche Bandinhalt verschoben werden, wenn $|u| \neq |v|$).
  (4) Wiederhole ab (1) bis entweder (a) das Band nur noch S enthält (Akzeptanz) oder (b) kein Vorkommen von v gefunden wird (Ablehnung).

Offenbar gilt: Die TM bei Eingabe w hat genau dann einen erfolgreichen Lauf, wenn die Grammatik eine Ableitung von w zulässt. $\square$

Typ 0 $\Leftarrow$ TM

Gegeben: Eine TM $\mathcal{M}$

Gesucht: Eine Grammatik G mit $\mathbf{L}(G) = \mathbf{L}(\mathcal{M})$

Idee:

- Ein Wort kann die Konfiguration einer TM kodieren.
- Berechnungsschritte können durch Ersetzungen von Teilwörtern simuliert werden.
- Grammatiken müssen die Wörter erzeugen, welche die TM akzeptiert.
  $\leadsto$ Vorgehen einer Grammatik:
  (1) Erzeuge ein beliebiges Eingabewort (nichtdeterministisch).
  (2) Simuliere die TM auf dieser Eingabe.
  (3) Falls TM akzeptiert: Ersetze die simulierte Endkonfiguration durch das ursprüngliche Eingabewort.

Typ 0 $\Leftarrow$ TM (Details 1)

Kodierungstrick:

- Variablen von G kodieren dreierlei Informationen:
  (1) Ursprüngliches Eingabeband: ein Zeichen aus $\Sigma \cup \{\sqcup\}$
  (2) Simuliertes Arbeitsband: ein Zeichen aus Γ
  (3) Simulierte Position und Zustand: ein Zeichen aus $Q \cup \{-\}$

Beispiel: Die Zeichenfolge $\begin{pmatrix}\mathtt{a}\\X\\-\end{pmatrix}\begin{pmatrix}\mathtt{a}\\\mathtt{a}\\-\end{pmatrix}\begin{pmatrix}\mathtt{b}\\X\\q\end{pmatrix}\begin{pmatrix}\mathtt{b}\\\mathtt{b}\\-\end{pmatrix}\begin{pmatrix}\sqcup\\X\\-\end{pmatrix}\begin{pmatrix}\sqcup\\\sqcup\\-\end{pmatrix}$ kodiert:

(a) die Eingabe war $\mathtt{aabb}$,
(b) die aktuell simulierte Konfiguration ist $X\mathtt{a}\,q\,X\mathtt{b}X\sqcup$

- Außerdem verwenden wir Variablen S (Start), A, B (Erzeugung der Startkonfiguration), $\sqcup$ (entsteht beim Aufräumen nach akzeptierender Endkonfiguration).

$$\leadsto V = \{\mathsf{S}, \mathsf{A}, \mathsf{B}, \sqcup\} \cup \big((\Sigma \cup \sqcup) \times \Gamma \times (Q \cup \{-\})\big)$$

Typ 0 $\Leftarrow$ TM (Details 2)

Phase 1: Initialisiere TM für eine beliebige Eingabe:

$$\mathsf{S} \to \begin{pmatrix}\mathtt{a}\\\mathtt{a}\\q_0\end{pmatrix}\mathsf{A} \text{ (für beliebige } \mathtt{a} \in \Sigma) \mid \begin{pmatrix}\sqcup\\\sqcup\\q_0\end{pmatrix}\mathsf{B}$$

$$\mathsf{A} \to \begin{pmatrix}\mathtt{a}\\\mathtt{a}\\-\end{pmatrix}\mathsf{A} \text{ (für beliebige } \mathtt{a} \in \Sigma) \mid \mathsf{B}$$

$$\mathsf{B} \to \begin{pmatrix}\sqcup\\\sqcup\\-\end{pmatrix}\mathsf{B} \mid \epsilon$$

$\leadsto$ Erzeugt Eingabewort und einen beliebig langen (leeren) Arbeitsspeicher.
$\leadsto$ Spur 1 speichert die geratene Eingabe;
$\leadsto$ Spuren 2 und 3 speichern die TM-Startkonfiguration bei dieser Eingabe.

Typ 0 $\Leftarrow$ TM (Details 3)

Phase 2: Simuliere TM-Berechnung auf Spuren 2 und 3:

- Für jeden TM-Übergang $\langle q', y, R\rangle \in \delta(q, x)$, beliebige $a, b \in \Sigma \cup \{\sqcup\}$ und beliebige $z \in \Sigma \cup \Gamma$:

$$\begin{pmatrix}a\\x\\q\end{pmatrix}\begin{pmatrix}b\\z\\-\end{pmatrix} \to \begin{pmatrix}a\\y\\-\end{pmatrix}\begin{pmatrix}b\\z\\q'\end{pmatrix}$$

- Für jeden TM-Übergang $\langle q', y, L\rangle \in \delta(q, x)$, beliebige $a, b \in \Sigma \cup \{\sqcup\}$ und beliebige $z \in \Sigma \cup \Gamma$

$$\begin{pmatrix}a\\z\\-\end{pmatrix}\begin{pmatrix}b\\x\\q\end{pmatrix} \to \begin{pmatrix}a\\z\\q'\end{pmatrix}\begin{pmatrix}b\\y\\-\end{pmatrix}$$

- Für jeden TM-Übergang $\langle q', y, N\rangle \in \delta(q, x)$ und $a \in \Sigma \cup \{\sqcup\}$:

$$\begin{pmatrix}a\\x\\q\end{pmatrix} \to \begin{pmatrix}a\\y\\q'\end{pmatrix}$$

Anmerkung: In Phase 1 vorbereiteter Bandbereich kann nicht verlassen werden!

Typ 0 $\Leftarrow$ TM (Details 4)

Phase 3: Akzeptanz und Aufräumen:

- Für alle $q \in F$ und $x \in \Sigma \cup \Gamma$ mit $\delta(q, x) = \emptyset$ und beliebige $a \in \Sigma \cup \{\sqcup\}$:

$$\begin{pmatrix} a \\ x \\ q \end{pmatrix} \to a$$

- Für alle $a, b \in \Sigma \cup \{\sqcup\}$ und beliebige $x \in \Sigma \cup \Gamma$:

$$a \begin{pmatrix} b \\ x \\ - \end{pmatrix} \to ab \qquad\qquad \begin{pmatrix} b \\ x \\ - \end{pmatrix} a \to ba$$

- Dabei erzeugte Blanks werden entfernt:

$$\sqcup \to \epsilon$$

Diese Grammatik erzeugt ein Wort w genau dann, wenn die TM einen akzeptierenden Lauf für w hat (unter Verwendung von beliebig viel Speicher). □

Zusammenfassung Typ 0

Wir haben also gezeigt:

Satz: Die Typ-0-Grammatiken erzeugen genau diejenigen Sprachen, die von einer Turingmaschine erkannt werden können.

Der Satz von Rice ist daher auf Typ-0-Grammatiken übertragbar:

Satz (informell): Für eine gegebene Typ-0-Grammatik G und eine nichttriviale Eigenschaft E von Typ-0-Sprachen ist es unentscheidbar, ob $\mathbf{L}(G)$ die Eigenschaft E hat.

Probleme wie Leerheit, Universalität, Äquivalenz zu einer anderen Typ-0-Grammatik, usw. sind daher für Typ-0-Grammatiken (wie auch für TMs) unentscheidbar.

Typ-1-Sprachen

Automaten für Typ 1?

Welches Berechnungsmodell entspricht den Typ-1-Sprachen?

- Kellerautomat: zu schwach (Typ 2)
- Turingmaschine: zu stark (Typ 0)

Lösung: Beschränkung des Arbeitsspeichers einer TM:

Ein linear beschränkte Turingmaschine (linear bounded automaton, LBA) ist eine nichtdeterministische Turingmaschine, die den Lese-/Schreibkopf nicht über das letzte Eingabezeichen hinaus bewegen kann. Versucht sie das, so bleibt der Kopf stattdessen an der letzten Bandstelle stehen.

Ein LBA kann also nur die Menge an Speicher nutzen, die durch die Eingabe belegt wird.

Die Definition der Übergangsrelation $\vdash$ für TMs wird für LBAs also wie folgt geändert: Sei $w\,q\,av$ eine Konfiguration, mit $w, v \in \Gamma^*$, $a \in \Gamma$ und $q \in Q$; sei $\delta(q, a) = \langle r, b, R\rangle$.

- falls $v \neq \epsilon$, dann $w\,q\,av \vdash wb\,r\,v$ (wie zuvor); $\quad w\,q\,av \vdash wb\,r\,v$
- falls $v = \epsilon$, dann $w\,q\,av \vdash w\,r\,b$ (neu). $\quad w\,q\,a \vdash w\,r\,b$

Beispiel

Die folgende TM zur Erkennung von $\{\mathtt{a}^i\mathtt{b}^i\mathtt{c}^i \mid i \geq 1\}$ ist ein LBA.

Arbeitsweise:

(1) Ersetze, angefangen von links, Vorkommen von a durch $\hat{a}$.

(2) Immer wenn ein a ersetzt wurde, suche ein b und ersetze es durch $\hat{b}$, suche anschließend rechts davon ein c und ersetze es durch $\hat{c}$.

(3) Gehe danach zurück zum ersten noch nicht ersetzten a und führe die Ersetzung (1) fort, bis alle a ersetzt worden sind.

(4) Akzeptiere, falls der Inhalt des Bandes die Form $\hat{a}^+\hat{b}^+\hat{c}^+$ hat.

(5) Akzeptiere, falls der Inhalt des Bandes die Form $\hat{a}^+\hat{b}^+\hat{c}^+$ hat.
- Laufe durch $\hat{a}$ und anschließend durch $\hat{b}$ durch; ersetze danach jeweils einzeln von links nach rechts jedes $\hat{c}$ durch $\hat{c}'$.
- Wird direkt nach dem Ersetzen von $\hat{c}$ und Kopfbewegung nach rechts wieder $\hat{c}'$ gelesen, ist das Bandende erreicht und die Eingabe wird akzeptiert.

(6) Andernfalls oder falls eine der Ersetzungen in Schritt (2) fehlschlägt, weil es zu wenige b oder c gibt, lehne die Eingabe ab.

(1) Ersetze, angefangen von links, Vorkommen von a durch $\hat{a}$.

(2) Immer wenn ein a ersetzt wurde, suche ein b und ersetze es durch $\hat{b}$, suche anschließend rechts davon ein c und ersetze es durch $\hat{c}$.

(3) Nach Ersetzung von c durch $\hat{c}$, überprüfe, ob bereits das Wortende erreicht ist:

Quiz: LBA

Quiz: Gegeben sei der folgende LBA $\mathcal{M} = \langle Q, \Sigma, \Gamma, \delta, q_0, F\rangle$ mit $\Sigma = \{\mathtt{0}, \mathtt{1}\}$.
...

Typ 1 $\Leftrightarrow$ LBA

Anmerkung: Wir beschränken uns auf Typ-1-Spachen ohne das Wort ϵ. Diesen Sonderfall müssten LBAs anders behandeln, da eine TM nicht mit 0 Speicherzellen arbeiten kann. Das ist nicht schwer,[1] aber auch nicht sehr interessant.

Satz: Die Typ-1-Grammatiken erzeugen genau diejenigen Sprachen, die von einem LBA erkannt werden können.

Beweis: Wir können fast die gleichen Konstruktionen anwenden wie bei Typ 0:

(1) Typ 1 ⇒ LBA: Eine TM kann wie zuvor Grammatikregeln rückwärts anwenden. Bei Typ-1-Regeln ist sichergestellt, dass dabei niemals mehr Speicher benutzt wird als am Anfang.

(2) LBA ⇒ Typ 1: Die Konstruktion liefert schon fast eine Typ-1-Grammatik ...

[1] Z.B. durch Verwendung eines Endzeichens nach der Eingabe.

Typ 1 $\Leftarrow$ LBA (1)

Die zuvor verwendete TM-Grammatik auf einen Blick:

$$\mathsf{S} \to \begin{pmatrix}\mathtt{a}\\ \mathtt{a}\\ q_0\end{pmatrix}\mathsf{A} \ \text{(für beliebige } \mathtt{a} \in \Sigma) \ \mid \ \begin{pmatrix}\sqcup\\ \sqcup\\ q_0\end{pmatrix}\mathsf{B}$$

$$\mathsf{A} \to \begin{pmatrix}\mathtt{a}\\ \mathtt{a}\\ -\end{pmatrix}\mathsf{A} \ \text{(für beliebige } \mathtt{a} \in \Sigma) \ \mid \ \mathsf{B} \qquad \mathsf{B} \to \begin{pmatrix}\sqcup\\ \sqcup\\ -\end{pmatrix}\mathsf{B} \ \mid \ \epsilon$$

$$\begin{pmatrix}a\\ x\\ q\end{pmatrix}\begin{pmatrix}b\\ z\\ -\end{pmatrix} \to \begin{pmatrix}a\\ y\\ -\end{pmatrix}\begin{pmatrix}b\\ z\\ q'\end{pmatrix} \qquad \begin{pmatrix}a\\ z\\ -\end{pmatrix}\begin{pmatrix}b\\ x\\ q\end{pmatrix} \to \begin{pmatrix}a\\ z\\ q'\end{pmatrix}\begin{pmatrix}b\\ y\\ -\end{pmatrix} \qquad \begin{pmatrix}a\\ x\\ q\end{pmatrix} \to \begin{pmatrix}a\\ y\\ q'\end{pmatrix}$$

$$\begin{pmatrix}a\\ x\\ q\end{pmatrix} \to a \qquad a\begin{pmatrix}b\\ x\\ -\end{pmatrix} \to ab \qquad \begin{pmatrix}b\\ x\\ -\end{pmatrix}a \to ba \qquad \sqcup \to \epsilon$$

Problematisch für Typ 1 sind nur die beiden ϵ-Regeln, die aber nur wegen der zusätzlichen Blanks nötig sind.

Typ 1 $\Leftarrow$ LBA (2)

Modifizierte Grammatik zur Simulation von LBAs:

$$S \to \begin{pmatrix} a \\ a \\ q_0 \end{pmatrix} A \ \text{(für beliebige } a \in \Sigma) \ \Big| \ \begin{pmatrix} a \\ a \\ q_0 \end{pmatrix} \ \text{(für beliebige } a \in \Sigma)$$

$$A \to \begin{pmatrix} a \\ a \\ - \end{pmatrix} A \ \text{(für beliebige } a \in \Sigma) \ \Big| \ \begin{pmatrix} a \\ a \\ - \end{pmatrix} \ \text{(für beliebige } a \in \Sigma)$$

$$\begin{pmatrix} a \\ x \\ q \end{pmatrix}\begin{pmatrix} b \\ z \\ - \end{pmatrix} \to \begin{pmatrix} a \\ y \\ - \end{pmatrix}\begin{pmatrix} b \\ z \\ q' \end{pmatrix} \qquad \begin{pmatrix} a \\ z \\ - \end{pmatrix}\begin{pmatrix} b \\ x \\ q \end{pmatrix} \to \begin{pmatrix} a \\ z \\ q' \end{pmatrix}\begin{pmatrix} b \\ y \\ - \end{pmatrix} \qquad \begin{pmatrix} a \\ x \\ q \end{pmatrix} \to \begin{pmatrix} a \\ y \\ q' \end{pmatrix}$$

$$\begin{pmatrix} a \\ x \\ q \end{pmatrix} \to a \qquad a\begin{pmatrix} b \\ x \\ - \end{pmatrix} \to ab \qquad \begin{pmatrix} b \\ x \\ - \end{pmatrix} a \to ba$$

Diese Grammatik simuliert wie zuvor beliebige (N)TMs, aber nur auf dem Speicherbereich, der von der Eingabe belegt wird. □

Konfigurationsgraphen

Das Wortproblem bei Typ 0 ist unentscheidbar. Und bei Typ 1?

Beobachtung:

- Auf einem beschränkten Speicher gibt es nur beschränkt viele Konfigurationen, genauer gesagt:

  Konfigurationszahl bei n Zellen: $\underbrace{|\Gamma|^n}_{\text{Bandinhalt}} \cdot \underbrace{n}_{\text{Kopfpositionen}} \cdot \underbrace{|Q|}_{\text{Zustände}}$

- Man kann entscheiden, ob eine TM von einer Konfiguration in eine andere wechseln kann.

Für eine Eingabe w können wir also den kompletten Graphen aller möglichen LBA-Konfigurationen und Übergänge konstruieren.
$\leadsto$ Konfigurationsgraph

Das Wortproblem für Typ 1

Wortproblem (anders ausgedrückt): Gibt es eine akzeptierende Endkonfiguration, die im Konfigurationsgraphen von der Startkonfiguration aus erreichbar ist?

Daraus folgt:

Satz: Das Wortproblem für Typ-1-Sprachen ist entscheidbar.

Beweis: Man kann den folgenden Algorithmus anwenden: (1) Berechne den (exponentiell großen) Konfigurationsgraphen einer entsprechenden Turingmaschine für das gegebene Wort; (2) prüfe, ob es in diesem Graphen einen Pfad von der Startkonfiguration zu einer Endkonfiguration gibt. □

Unser Algorithmus benötigt (immer) exponentiell viel Zeit.

Aber: Es ist bis heute nicht bekannt, ob es einen Algorithmus gibt, der im schlimmsten Fall weniger als exponentiell viel Zeit benötigt!

Beispiel: Das Halteproblem ist keine Typ-1-Sprache, da es nicht entscheidbar ist.

Abschlusseigenschaften Typ 0 und Typ 1

Bekannte Abschlusseigenschaften

Wir wissen bereits:

Satz (siehe Vorlesung 14): Sowohl die Klasse der Typ-1-Sprachen als auch die Klasse der Typ-0-Sprachen ist unter Vereinigung abgeschlossen.

Satz: Die Klasse der Typ-0-Sprachen ist nicht unter Komplement abgeschlossen.

Beweis: Das Komplement des Halteproblems ist nicht semi-entscheidbar (siehe Vorlesung 19). □

Schnitt, Konkatenation und Kleene-Stern

Weitere Abschlusseigenschaften sind nicht schwer zu finden:

- Schnitt: Simuliere erst die erste TM, dann (bei Akzeptanz) die zweite; verwende ein „mehrspuriges“ Alphabet, um die Eingabe für die zweite Simulation zu speichern
- Konkatenation: Rate und markiere die Trennstelle der beiden Wörter; teste dann jedes der Wörter einzeln
- Kleene-Stern: Rate und teste einen ersten nichtleeren Teilabschnitt; wiederhole dies bis das gesamte Wort erkannt wurde

Diese Konstruktionen funktionieren auch bei linear beschränktem Speicher, also:

Satz: Sowohl die Klasse der Typ-1-Sprachen als auch die Klasse der Typ-0-Sprachen ist unter Schnitt, Konkatenation und Kleene-Stern abgeschlossen.

Die LBA-Probleme

Zwei Probleme sind schon seit Erfindung der LBAs bekannt (Kuroda, 1964):

(1) Erkennen LBA dieselben Sprachen wie deterministische LBA?

(2) Sind die von LBA erkennbaren Sprachen unter Komplement abgeschlossen?

Das zweite Problem lösten überraschend nach über 20 Jahren unabhängig voneinander Robert Szelepcsényi (1987) und Neil Immerman (1988):

Satz von Immerman und Szelepcsényi:
Die Typ-1-Sprachen sind unter Komplement abgeschlossen.

Beweis: siehe Sipser (Abschnitt 8.6) oder Schöning (Abschnitt 1.4) oder Vorlesung Complexity Theory der TU Dresden; kein Stoff dieser Veranstaltung.

Das erste LBA-Problem ist bis heute ungelöst.

Übersicht Abschlusseigenschaften

	Abschluss unter …					
Sprache	$\cap$	$\cup$	$\overline{}$	$\circ$	*	Automat
Typ 0	✓	✓	×	✓	✓	TM (DTM/NTM)
Typ 1	✓	✓	✓	✓	✓	LBA ($\stackrel{?}{=}$ det. LBA)
Typ 2	×	✓	×	✓	✓	PDA
Det. Typ 2	×	×	✓	×	×	DPDA
Typ 3	✓	✓	✓	✓	✓	DFA/NFA

Zusammenfassung und Ausblick

Turingmaschinen charakterisieren Typ-0-Sprachen.

Linear beschränkte Turingmaschinen charakterisieren Typ-1-Sprachen.

Das Wortproblem für Typ-1-Sprachen ist entscheidbar, aber kompliziert.

Offene Fragen:

- Wollten wir nicht auch noch etwas Logik behandeln?
- Was hat das mit Sprachen, Berechnung und TMs zu tun?