Beispiele

1)

Die Sprache L soll all binären Wörter akzepieren, die immer abwechselnd nullen und einsen haben.

$S \to \varepsilon|0A|1B$

$A \to \varepsilon|1B$

$B \to \varepsilon|0A$

Wir wollen diese Sprache jetzt in einen Automaten transformiern.

2)

Die selbe Sprache, nur muss es gleich viele Nullen und Einsen geben.

$S \to \varepsilon|0A|1B$

$A \to 1C$

$B \to 0D$

$C \to \varepsilon|0A$

$D \to \varepsilon|1B$

3)

Die binäre Sprache, die eine gerade Anzahl an Nullen und Einsen

$S \to \varepsilon|0A|1B$

$A \to 0S|1C$

$B \to 1S|0C$

$C \to 1A|0B$

Formale Definition

DFA

(Deterministischer Finiter Automat)

Ein DFA ist wie folgt definiert:

$(\Sigma, Q, F, q_0, \delta)$

- Σ ist unser Alphabet
- Q ist die Menge an Zuständen
- F ist die Menge finalen Zuständen. Dabei ist F natürlich eine Untermenge von Q
- q_0 ist unser Anfangszustand. (In unseren Beispielen auch S genannt)
- δ ist unsere Transitionsfunktion. (Analog zur Produktionsmenge bei Grammatiken)

δ ist dabei wie folgt definiert:

$\delta : \Sigma \times Q \to Q$

Wir wollen eigentlich eine Funktion, die ein ganzes Wort bekommt und uns den Endzutand.

Dafür definieren wir eine neue Funktion $\bar{\delta}$:

$\bar{\delta} : \Sigma^* \times Q \to Q$

Wir definieren $\bar{\delta}$ rekursiv:

- $\bar{\delta}(\varepsilon, q) = q$
- $\bar{\delta}(aw, q) = \bar{\delta}(w, \delta(a, q))$

Als Beispiel:

$0101, q_0 \to 101, q_1 \to 01, q_2 \to \ldots \to \varepsilon, q_x$

1)

Für die Sprache P :

- $S \to \varepsilon | 0A | 1B$
- $A \to \varepsilon | 1B$
- $B \to \varepsilon | 0A$

Für den Automaten δ :

- $\delta(0, q_S) = q_A$
- $\delta(1, q_S) = q_B$
- $\delta(0, q_A) = q_{Fehler}$
- $\delta(1, q_A) = q_B$
- $\delta(0, q_B) = q_A$
- $\delta(1, q_B) = q_{Fehler}$
- $\delta(0, q_{Fehler}) = q_{Fehler}$
- $\delta(1, q_{Fehler}) = q_{Fehler}$

Für Diagramm siehe Bild.

Als Tabelle:

δ	0	1
q_S	q_A	q_B
q_A	q_{Fehler}	q_B
q_B	q_A	q_{Fehler}
q_{Fehler}	q_{Fehler}	q_{Fehler}

Ihm ist es egal welche Darstellung wir für Automaten nutzen.

DFA kann alle Sprachen der L_3 darstellen.

NFA

(Nicht-deterministiche Finite Automat)

Unterscheidet sich vom DFA nur in der δ-Funktion. Speziell ist die wie folgt definiert:

$\delta : \Sigma \times Q \to 2^Q$

(In Worten: Für jede Eingabe und jeden Zustand geht die δ-Fkt. in eine Menge an Zuständen bzw. in eine Untermenge von Q)

Als Beispiel:

- $\delta(0, q_0) = \{q_1, q_2\}$
- ...

Es ist eindeutig, dass jeder DFA einfach in einen NFA verwandelt werden kann, indem wir die Ausgabe der δ-Fkt. in eine Menge des einen Elementes verwandeln. (Indem wir einfach geschwungene Klammern drum machen).

Beispielhaft:

$\delta(a, q) = q_x \to \delta(a, q) = \{q_x\}$

Wichtig:

$L(NFA) = L(DFA)$

PDA

($PDA = NPDA =$ "Nicht-deterministischer Push-Down-Automata")

Ist definiert als NFA mit einem Stack (/Keller).

- Stack = Γ

Wir definieren entsprechend auch eine neue Transitions-Fkt.:

$\delta : \Sigma^* \times Q \times \Gamma^* \to 2^{Q \times \Gamma^*}$

Selbst wenn das leere Wort kommt, können wir noch von einem Zustand in den nächsten gehen, weil wir den Stack haben.

Nicht-deterministisch $\to$ beim leeren Wort können wir beim Zustand bleiben oder in weiteren gehen, wenn der Stack noch nicht leer ist.

- $L(NPDA) = L_2$
- Es gibt zwei Arten, die NPDAs zu definieren. Entweder ist ein Wort akzeptiert, wenn das leere Wort da ist und wir in einem finalen Zustand. Oder wenn der Stack leer ist und wir in einem finalen Zustand sind. Beide Definitionen sind äquivalent, d.h. wir können über beide Definitionen die selbe Klasse an Sprachen darstellen.
  - $L(NPDA)_{leeresWort} = L(NPDA)_{leererStack}$

DPDA

Der DPDA ist deterministisch. D.h.wenn das leere Wort da ist, dann können wir nicht in einen weiteren Zustand übergehen.

Wir definieren entsprechend auch eine neue Transitions-Fkt.:

$\delta : \Sigma^* \times Q \times \Gamma^* \to Q \times \Gamma^*$

$L(DPDA) \neq L_2$, stattdessen gilt $L_3 \subset L(DPDA) \subset L_2$.

Wir können wieder die beiden Definitionen wie beim $NPDA$ benutzen, aber diesmal sind sie nicht äquivalent:

- $L(DPDA)_{leerenStack} \subset L(DPDA)_{leerenWort}$

Präfixeigenschaft bzw. Präfixfreiheit

Defintion:

$\forall a, b \in L : ab \notin L$

Wichtigkeit für PDAs

Wenn eine Sprache L nicht präfixfrei ist, dann kann sie nicht von einem $DPDA$ modelliert.

D.h.: $L(DPDA)_{leerenStack} = \{L \in L_2 | L \text{ ist präfixfrei } \wedge L(DPDA)_{leerenWort} \text{ akzeptiert L}\}$

- $L(DPDA)_{leererStack} \to L$ ist präfixfrei
- $\neg L(DPDA)_{leererStack} \to L$ nicht präfixfrei

Z.B. Die Srache dt. Namen:

"Anna" & "Annalena" sind beide in der Sprache

Wenn wir "Anna" lesen, dann ist unklar, ob wir schon fertig sind oder ob noch weitere Buchstaben für "Annalena" kommen. Da diese Sprache offensichtlich nicht präfixfrei ist, kann sie nur nicht-deterministisch dargestellt.

Turing-Maschine

(Speziell: "Deterministische Turing-Maschinen")

Turing-Maschine ist wie der DPDA nur statt einem Stack, gibt es ein unendlich langes Tape.

Formale Definition:

$(\Sigma, b, M, Q, F, q_0, \delta)$

- Σ ist unser Alphabet
- b ist unser Symbol um Wörter von einander zu trennen. Es ist nicht im Alphabet.
- M ist unsere Menge an Richtungsbefehlen Speziell haben wir "Links", "Rechts", "Nicht bewegen"
- Q ist unsere Menge an Zuständen
- F ist unsere Menge an finalen/akzeptierten Zuständen
- q_0 ist unser Startzustand
- δ ist unsere Transitionsfunktion

Wir definieren δ:

$\delta : Q \times (\Sigma \cup \{b\}) \to Q \times (\Sigma \cup \{b\}) \times M$

Eigenschaften:

- $L(TM) = L_0$
- TM können alle Sprachen mit abzählbar unendlich vielen Wörter darstellen.

Warum kann die Turing Maschine mehr als ein NPDA, obwohl die Turing Maschine keinen Stack hat:

- Weil das unendlich lange Tape der Turing Maschine auch einen Stack darstellen kann, aber halt auch mehr.

Wortproblem

Gegeben ein Wort und eine Sprache,bestimme ob das Wort in der Sprache ist. (Hier ist n die Länge des Wortes)

- In L_3 bzw. bei einem FA gilt: $O(n) \rightarrow \in P$
- Bei $DPDA$ gilt: $O(n) \rightarrow \in P$
- In L_2 bzw. bei einem PDA gilt: $O(n^3) \rightarrow \in P$
- In L_1 gilt: $\in P\text{-}SPACE$
- In L_0 bzw. bei einer TM gilt: Unberechenbar

Warum ist das Wortproblem in L_0 unberechenbar?

- Berechenbarkeit heißt, dass es rekursiv aufzählbar ist und entsprechend darf es höchstens abzählbar unendlich viele Schritte geben, um das Problem zu lösen.
- Es gibt Sprachen in L_0, die unabzählbar viele Wörter in ihrer Sprache haben.
- D.h. es muss Wörter geben, die nicht mit abzählbar vielen Schritten bestimmbar in der Sprache sind
- Damit ist das Wortproblem in L_0 nicht berechenbar.

Entscheidbarkeit

Eine Sprache L ist entscheidbar, wenn sowohl L als auch $\bar{L}$ semi-entscheidbar sind.

Eine Sprache L ist semi-entscheidbar, wenn für alle Wörter in der Sprache in abzählbar vielen Schritten entschieden werden kann, dass das Wort in der Sprache liegt. Falls ein Wort nicht in der Sprache liegt, ist es nicht sicher, dass wir eine Antwort bekommen.

D.h. entscheidbare Sprachen (Sprachen, bei denen immer entschieden/berechnet werden kann, ob ein Wort in der Sprache liegt oder nicht) haben immer nur abzählbar unendlich viele Wörter in L und in $\bar{L}$.

Es gibt allerdings Sprachen, die nicht entscheidbar sind.

Halting-Problem

Problem: Gegeben ein Programm für unsere Turing-Maschiene, wollen wir entscheiden, ob das Program terminiert oder nicht.

Idee: Wir zeigen, dass es unabzählbar unendlich viele Programme gibt. Dann haben wir nach der selben Begründung des Wortproblems ein unberechenbares Problem.

Wir machen eine Tabelle mit allen Algorithmen und allen Inputs und de Antwort, ob der Algorithmus be diesem Input terminiert (d.h. wir gehen davon aus, dass wir das Halting-Problem schon gelöst hätten):

	1	2	3	4	...
A1	ja	nein	ja	ja	...
A2	nein	ja	JA	nein	...
A3	nein	nein	nein	nein	...
...	...	...	...	...	...

Wir erstellen jetzt einen weiteren Algorithmus A, der existiert, welcher nicht in unserer Tabelle ist. Entsprechen können wir für diesen Algorithmus A nicht sagen, ob er terminiert oder nicht.

Wie definieren wir A?

- $A(1) = \neg A1(1)$
- $A(2) = \neg A2(2)$
- $A(3) = \neg A3(3)$
- ...

(Diagonale in Tabelle zeichnen)

Da dieser Algorithmus nicht in der Tabelle sein kann, muss es mehr als abzählbar unendlich viele Algorithmen geben. Warum?

Nehme an, der ALgorithmus A wäre in der Tabelle. D.h. es gibt eine Zeile i, in der der Algorithmus steht. Per Def. von A gilt für $A(i) = \neg Ai(i)$. Da A aber in der i-ten Zeile steht, ist $A = Ai$. D.h wenn $Ai(i)$ terminiert, dann terminiert $A(i)$ nicht (und umgedreht). Da A und Ai aber gleich sind, muss der Algorithmus sowohl terminieren als auch nicht terminieren. Paradoxon -> Annahme falsch -> Der Algorithmus steht nicht in der Tabelle.

Mengenbeziehungen zwischen Sprachklassen

$L_3 = L(DFA) = L(NFA) \subset L(DPDA)_{leerenStack} \subset L(DPDA)_{leerenWort} = L_{präfixfrei} \subset L_2 = L(NPDA) \subset L_1 \subset L_{entscheidbar} \subset L_0 =$