Lösungen zu den Übungsaufgaben im Buch *Automaten und Sprachen: Theoretische Informatik für die Praxis* von Andreas Müller, ISBN 978-3-662-70145-4 (Softcover), ISBN 978-3-662-70146-1 (eBook), https://link.springer.com/book/10.1007/978-3-662-70146-1. Website zum Buch: https://autospr.ch

Kapitel 11: Entscheidbarkeit

11.1. Zeigen Sie, dass eine endliche Sprache L entscheidbar ist.

Lösung. Da die Sprache endlich ist, gibt es einen regulären Ausdruck, der genau auf die Wörter von L passt. Der zugehörige deterministische endliche Automat kann auf einer Turing-Maschine simuliert werden, die somit ein Entscheider für die Sprache ist. ○

11.2. Seien L_1 und L_2 zwei Turing-entscheidbare Sprachen über dem Alphabet Σ. Zeigen Sie, dass die folgenden Sprachen Turing-entscheidbar sind:

a) $L_1 \cup L_2$

b) $L_1 L_2$

c) L_1^*

d) $\Sigma^* \setminus L_1$

e) $L_1 \cap L_2$

Hinweis. Turing-entscheidbar heißt, dass es zwei Funktionen

```
boolean l1(String w);
boolean l2(String w);
```

gibt, welche genau dann `true` zurückgeben, wenn `w` in L_1 bzw. L_2 ist. Die Aufgabe besteht dann darin, für jede Sprache L in den Teilaufgaben eine neue Funktion

```
boolean l(String w) {
  ....
}
```

zu programmieren, welche genau dann `true` zurückgibt, wenn `w` in L ist.

Lösung. a) Um zu entscheiden, ob ein Wort w in $L_1 \cup L_2$ ist, lassen wir zuerst M_1 auf dem Input w laufen, und falls diese das Wort verwirft, M_2. Akzeptiert eine der Maschinen das Wort, wird es akzeptiert, andernfalls wird es verworfen.

Mit den Notationen des Hinweises könnte man dieses Problem mit folgendem Code lösen:

```
boolean l(String w) {
    return l1(w) || l2(w);
}
```

b) Für jede der $|w| + 1$ Unterteilungen des Wortes w in zwei Teilwörter $xy = w$ lassen wir M_1 auf x und M_2 auf y laufen. Akzeptieren die Maschinen für eine der Unterteilungen das Wort, akzeptieren wir, wir verwerfen, wenn für keine Unterteilung beide Maschinen akzeptieren.

Mit den Notationen des Hinweises könnte man dieses Problem mit folgendem Code lösen:

```
boolean l(String w) {
    for (int i = 0; i <= w.length(); i++) {
        String w1 = w.substring(0, i);
        String w2 = w.substring(i);
        if (l1(w1) && l2(w2)) {
            return true;
        }
    }
    return false;
}
```

c) Wenn $w \in L_1^*$, dann kann man w schreiben als $w = x_1 \dots x_n$, wobei $x_i \in L_1$ für alle i und $x_i \neq \varepsilon$. Es gibt endlich viele Unterteilungen von w in maximal n Teilwörter mit einer Länge von mindestens einem Zeichen. Für alle solchen Unterteilungen testen wir jede Komponente x_i mit der Turingmaschine M_1. Falls M_1 alle Komponenten einer Unterteilung akzeptiert, akzeptieren wir das Wort, wenn dies in keinem Fall vorkommt, verwerfen wir das Wort.

Man kann dieses Problem sehr elegant mit Rekursion lösen. Die Funktion untersucht alle Unterteilungen des Wortes in zwei Teile x_1 und x_2, indem sie zuerst den Entscheider für L_1 mit Input x_1 aufruft, und dann sich selbst mit Input x_2 aufruft. Diese Rekursion ist endlich.

Mit den Notationen des Hinweises könnte man dieses Problem mit folgendem Code lösen:

```
boolean l(String w) {
    // leeres Wort, kein Test in L1 notwendig
    if (w.length() == 0) {
        return true;
    }
    // Beendigung der Rekursion wenn das Wort in L1 ist
    if (l1(w)) {
        return true;
    }
    // Rekursion erforderlich, untersuche alle Unterteilungen
    // (fuer w.length() == 1 wird hier nichts ausgefuehrt)
    for (int i = 1; i < w.length(); i++) {
        String w1 = w.substring(0, i);
        String w2 = w.substring(i);
        if (l1(w1) && l(w2)) {
            return true;
        }
    }
    return false;
}
```

d) Für die Sprache $\Sigma^* \setminus L_1$ verwenden wir die Turing Maschine $\bar{M}_1$, in welcher gegenüber M_1 die Zustände q_{accept} und q_{reject} ihre Rollen vertauschen. $\bar{M}_1$ ist ein Entscheider, der genau die Wörter von $\Sigma^* \setminus L_1$ akzeptiert.

Mit den Notationen des Hinweises könnte man dieses Problem mit folgendem Code lösen:

```
boolean l(String w) {
    return !l1(w);
}
```

e) Wir lassen M_1 auf w laufen und anschließen M_2 auf w. Wenn beide Maschinen akzeptieren, akzeptieren wir das Wort, andernfalls verwerfen wir.

Mit den Notationen des Hinweises könnte man dieses Problem mit folgendem Code lösen:

```
boolean l(String w) {
    return l1(w) && l2(w);
}
```

○

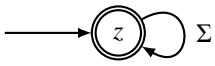
11.3. Sei

$$ALL_{\text{DEA}} = \{ \langle A \rangle \mid A \text{ ist ein DEA und } L(A) = \Sigma^*. \}.$$

Zeigen sie: ALL_{DEA} ist entscheidbar.

Lösung. Drei mögliche Lösungen:

1. Der folgende Algorithmus entscheidet: Bilde den minimalen Automaten von A und akzeptiere, falls A der Automat



ist.

2. Der folgende Algorithmus entscheidet: bilde den Automaten, der $\overline{L(A)}$ akzeptiert (dieser Automat existiert, weil das Komplement einer regulären Sprache wieder regulär ist) und wende den Entscheider von E_{DEA} darauf an.

3. Σ^* ist regulär, es gibt also einen Automaten B, der Σ^* akzeptiert. Jetzt wendet man den Entscheider von EQ_{DEA} auf A und B an, und akzeptiert, wenn dieser akzeptiert. Andernfalls verwirft man. ○

11.4. Sei

$$A\varepsilon_{CFG} = \{ \langle G \rangle \mid G \text{ ist eine kontextfreie Grammatik und } \varepsilon \in L(G). \}.$$

Zeigen Sie: $A\varepsilon_{\text{CFG}}$ ist entscheidbar.

Lösung. Zwei mögliche Lösungen:

1. Der folgende Algorithmus entscheidet: bilde die Chomsky Normalform von G und akzeptiere, falls die Regel $S \to \varepsilon$ vorkommt.
2. Man verwendet den Enscheider für A_{CFG}, dessen Existenz auf Seite 266 bewiesen wurde (dabei wurde auch die CNF verwendet), und wendet ihn auf das Paar (G, ε) an. ○

11.5. Sei

$$INFINITE_{\text{DEA}} = \{\langle A\rangle \mid A \text{ ist ein DEA und } L(A) \text{ unendlich.}\}.$$

Zeigen Sie, dass $INFINITE_{\text{DEA}}$ entscheidbar ist.

Lösung. Der folgende Algorithmus entscheidet: bilde den minimalen Automaten und wandle ihn in einem regulären Ausdruck um. Falls der reguläre Audruck einen $*$-Operator enthält, der auf einen nichtleeren Teilstring angewandt wird, akzeptiere, andernfalls verwerfe. ○

11.6. Sei

$$MIRROR_{\text{DEA}} = \left\{ \langle A\rangle \;\middle|\; \begin{array}{l}\text{Der DEA } A \text{ akzeptiert } w \text{ genau dann, wenn}\\ \text{er das gespiegelte Wort } w^t \text{ akzeptiert.}\end{array} \right\}.$$

Zeigen Sie: $MIRROR_{\text{DEA}}$ ist entscheidbar.

Lösung. Die Bedingung bedeutet $L(A) = L(A)^t$, man muss also entscheiden, ob $L(A) = L(A)^t$. Dazu wendet man den Entscheider für EQ_{DEA} auf das Paar (A, A^t) an. ○

11.7. Ein E-Learning-System soll Schülern arithmetische Ausdrücke zur Auswertung geben und die Antworten der Schüler überprüfen. Die Qualitätssicherung verlangt vom Programmierer dieses Moduls, dass er einen unabhängigen Test schreibt, welcher aus dem Source-Code des Moduls ableiten kann, ob je ein inkorrekter arithmetischer Ausdruck als Aufgabe gestellt werden könnte. Kann der Programmierer dieses Problem lösen?

Lösung. Das Modul soll nur Wörter einer Sprache von korrekten arithmetischen Ausdrücken produzieren. Die vom Modul akzeptierte Sprache soll also die Eigenschaft

$$P = \text{“enthält nur korrekte arithmetische Ausdrücke”}$$

haben. Diese Eigenschaft ist nicht trivial. Die Sprache

$$L_1 = \{w \mid w \text{ ist ein korrekter arithemtischer Ausdruck}\}$$

ist Turing erkennbar (sogar Turing-entscheidbar, dank des CYK-Algorithmus), und hat die Eigenschaft P. Die Sprache

$$L_2 = \{\text{“7−”}\}$$

ist als endliche und damit reguläre Sprache ebenfalls Turing-erkennbar, hat aber die Eigenschaft P nicht. Nach dem Satz von Rice folgt daher, dass nicht entscheidbar ist, ob die von dem Modul akzeptierte Sprache von Wörtern nur aus korrekten arithmetischen Ausdrücken besteht. So einen Test kann der Programmierer also nicht schreiben. ○

Lösung. Man kann auch direkt eine Reduktion von A_{TM} auf das vorliegende Problem konstruieren. Wir nehmen also an, wir hätten einen Entscheider, der herausfinden kann, ob eine Turingmaschine ausschließlich korrekte arithmetische Ausdrücke akzeptiert. Wir konstruieren jetzt die Reduktionsabbildung, die $\langle M, w\rangle$ auf das folgende Programm M' mit Input u abbildet:

1. Teste, ob u kein arithmetischer Ausdruck ist. Falls u ein arithemtischer Ausdruck ist: q_{reject}
2. Lasse M auf w laufen. Falls M akzeptiert: q_{accept}, andernfalls q_{reject}

Dieses Programm akzeptiert genau die Sprache, die aus lauter Wörtern besteht, die keine arithemetischen Ausdrücke sind, oder die leere Sprache, und zwar abhängig davon ob M das Wort w akzeptiert oder nicht. Indem man den Entscheider für arithmetische Ausdrücke auf M' anwendet kann man jetzt also entscheiden, ob M auf w anhält, man hat einen Entscheider für A_{TM} gefunden. Da es einen solchen nicht geben kann, ist auch das ursprüngliche Problem nicht entscheidbar. ○

11.8. Der chinesische Präsident Xi Jinping hat sich nicht darüber gefreut, dass er auf dem Internet mit Winnie the Pooh verglichen wird. Er hat seine Zensoren angewiesen, Bilder von Winnie the Pooh zu blockieren, sogar in der in China verbreiteten Chat-Anwendung WeChat werden Meldungen, die die Zeichenkette `Winnie the Pooh` enthalten, mit einer Fehlermeldung quittiert. Trotzdem gibt es natürlich immer noch zahllose Apps, die Winnie the Pooh nicht blockieren und damit die chinesischen Zensoren brüskieren. Gibt es eine Möglichkeit, Apps zu blockieren, die zur Laufzeit die Zeichenkette `Winnie the Pooh` erzeugen können?

Lösung. Sei P irgend ein Programm, welches wir daraufhin untersuchen wollen, ob es den String `Winnie the Pooh` produzieren kann. Mit dem Debugger könnten wir feststellen, ob ein Programm den genannten String produziert und könnten das Programm anhalten lassen. Herauszufinden, ober der String produziert wird, ist also gleichbedeutend damit, herauszufinden, ob ein Programm anhält. Letzteres ist das Halteproblem, welches nicht entscheidbar ist.

Etwas formaler können wir auch mithilfe einer Reduktion vom Halteproblem nachweisen, dass Problem nicht entscheidbar ist. Sei also Q ein Programm welches keinen Output produziert, so wie das in der Theorie zum Halteproblem vorausgesetzt wurde. Von Q wollen wir entscheiden, ob es anhält oder nicht. Wir konstruieren das folgende neue Programm Q':

1. Lasse Q laufen
2. Gebe die Zeichenkette `Winnie the Pooh` aus

Dieses Programm produziert genau dann den Output `Winnie the Pooh`, wenn Q anhält. Die Abbildung $Q \to Q'$ ist daher eine Reduktion vom Halteproblem auf das Winnie-the-Pooh-Problem. Da das Halteproblem nicht entscheidbar ist, ist auch das Winnie-the-Pooh-Problem nicht entscheidbar. ○

11.9. Zeigen Sie, dass die Sprache

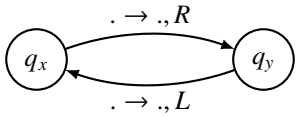
$$B_{\text{TM}} = \left\{ \langle M\rangle \;\middle|\; \begin{array}{l}\text{Die Turing-Maschine } M \text{ beschreibt}\\ \text{einen beschränkten Teil des Bandes.}\end{array} \right\}$$

nicht entscheidbar ist.

Lösung. Wir verwenden eine Reduktion vom Problem $HALT\varepsilon_{\mathrm{TM}}$. Sei also M eine Turing-Maschine, wir konstruieren eine neue Maschine wie folgt. Dazu betrachten wir zunächst die Zweibandmaschine M_1, die in jedem Schritt auf dem ersten Band einen Schritt der Maschine M ausführt. Gleichzeitig führt schreibt Sie ein Zeichen 1 auf das zweite Band und bewegt den Kopf nach rechts.

Die Maschine S_0 beschreibt einen Teil des zweiten Bandes, der genau so lang ist wie die Laufzeit des Programms. Wenn M anhält, ist der beschriebene Teil beider Bänder beschränkt, es können höchstens so viele Zeichen geschrieben werden, wie die Laufzeit angibt.

Falls die Maschine M nicht anhält, dann kann der beschriebene Teil des ersten Bandes immer noch beschränkt sein, zum Beispiel wenn die Maschine in eine Endlosschleife wie



stecken bleibt. Der beschriebene Teil des zweiten Bandes ist aber in jedem Fall unbeschränkt.

Die Maschine S_0 kann in eine Standardmaschine S verwandelt werden. Die Maschine S hat dann die Eigenschaft, dass sie genau dann einen unbeschränkten Bereich des Bandes beschreibt, wenn M auf leerem Band anhält.

Die Abbildung $f\colon M \mapsto S$ ist daher eine Reduktion $f\colon HALT\varepsilon_{\mathrm{TM}} \leq B_{\mathrm{TM}}$. Da $HALT\varepsilon_{\mathrm{TM}}$ nicht entscheidbar ist, kann auch B_{TM} nicht entscheidbar sein. ○