Aufgabe 1

Die Eigenschaften aus b.), c.), e.) und f.) sind entscheidbar**!** Dies ist kein Widerspruch zum 1. Satz von Rice, da Eigenschaften b.), c.) und f.) keine Eigenschaft der akzeptierten Sprache, sondern der TM sind. Eigenschaft e.) ist zwar eine Eigenschaft der Sprache, allerdings ist es eine triviale Eigenschaft, da sie für jede TM M gilt.

a.) "Ist L(M) linear?" ist nicht entscheidbar, da die Eigenschaft nicht-trivial* ist und eine Eigenschaft der akzeptierten Sprache von M ist. Also ist der 1. Satz von Rice anwendbar, der besagt, daß die Eigenschaft nicht entscheidbar ist.

*) wir müssen dazu eine aufzählbare Sprache angeben, die die Eigenschaft erfüllt und eine, die die Eigenschaft nicht erfüllt:

$a^n b^n c^n$ ist aufzählbar (Blatt 11, Aufg 1)
$a^n b^n c^n$ ist nicht kontextfrei, also auch nicht linear.

$a^n b^n$ ist aufzählbar (Vorlesung) und linear.
(tatsächlich impliziert linear aufzählbar).

Aufgabe 1 (Forts.)

b.) Diese Eigenschaft ist entscheidbar, wir zählen die Anzahl der Zustände.

Achtung: Streng genommen können wir die Zustände gar nicht zählen, sondern nur die von ihnen ausgehenden Übergänge

c.) "Akzeptiert die TM M das Wort w nach höchstens n Schritten?" ist entscheidbar: Wir simulieren mit Hilfe der universellen TM U die TM M auf Eingabe w für n Schritte (vgl. beschränkte Simulation).

d.) "Ist L(M) entscheidbar?" ist wegen des 1. Satzes von Rice nicht entscheidbar.

- Es ist eine Eigenschaft der akzept. Sprache
- Die Eigenschaft ist nicht-trivial:
  "$a^n b^n$" ist entscheidbar
  L_H ist nicht entscheidbar

e.) "Ist L(M) aufzählbar?" ist für jede TM M wahr, also entscheidbar. → Antwort ist immer ja

f.) "Gibt es ein Wort w, das von M nach höchstens n Schritten akzeptiert wird?" ist entscheidbar**!**

<u>Begründung</u>: Wenn ein solches Wort w existiert, dann gibt es ein solches Wort w' mit $|w'| \leq n$; denn in n Schritten kann die TM höchstens die ersten n Zeichen von w "ansehen".

Aufgabe 1 f. (Forts.)

Wir müssen uns also nur endlich viele Wörter ansehen; für jedes dieser Wörter müssen wir uns nur die Berechnungen ansehen, die höchstens n Schritte haben. Davon gibt es auch nur endlich viele!

g) "Akzeptiert M mind. n Wörter?" ist wegen des ersten Satzes von Rice nicht entscheidbar:

- Es ist eine Eigenschaft der akzeptierten Sprache $|L| \geq 1$
- Die Eigenschaft ist nicht-trivial:

  $\emptyset$ erfüllt die Eigenschaft nicht

  $\{a, aa, aaa, \dots, \underbrace{aa\dots a}_{n}\}$ erfüllt die Eigenschaft

Aufgabe 2

Sie sollten antworten, daß Sie ihm das nicht abnehmen! Denn ein einfaches Diagonalisierungsverfahren zeigt, daß es mind. eine intuitiv berechenbare Funktion $f: \mathbb{N} \to \mathbb{N}$ gibt, die mit dieser Programmiersprache nicht berechnet werden kann:

Sei $f_0, f_1, f_2, f_3, \dots$ eine beliebige Aufzählung aller Funktionen $f_i: \mathbb{N} \to \mathbb{N}$ der Programmiersprache, in der alle einstelligen Funktionen der Programmiersprache vorkommen.

Eine solche Aufzählung existiert immer, wenn die syntaktische Korrektheit der Programmiersprache entscheidbar ist.

Dann ist die Funktion $f: \mathbb{N} \to \mathbb{N}$ mit $f(n) = f_n(n)+1$ offensichtlich intuitiv berechenbar.

Generiere das Programm für f_n und berechne dann $f_n(n)$; da alle Programme der Programmiersprache terminieren, liefert $f_n(n)$ irgendwann einen Wert. Dann bilden wir $f_n(n)+1$.

Nun kann aber f keine Funktion $f_i: \mathbb{N} \to \mathbb{N}$ sein; sie kann also in der Programmiersprache nicht implementiert werden:

Annahme $f = f_i \Rightarrow f(i) = f_i(i)+1$

Aufgabe 2 (Forts.)

Dieses Argument funktioniert für alle Programmiersprachen bzw. Berechnungsmodelle, für die

1. die syntaktische Korrektheit entscheidbar ist und für die
2. für jede Eingabe die Terminierung gewährleistet ist.

Insbes. gilt dies für

- die primitiv rekursiven Funktionen

  $f(n) \rightsquigarrow f(n-1)$ ← jeder rekursive Aufruf verringert n

- "reine" for-Programme

  ```
  for i := 1 to exp
    [ Li
  ```

  ← i und exp werden hier nicht modifiziert

Abgesehen davon ist man häufig an Programmen interessiert, die keine Funktion berechnen; beispielsweise sollte ein Betriebssystem im Idealfall nicht terminieren (vgl. Kapitel VII).