1) Sei $M = (Q, \Sigma, \Gamma, \vdash, \sqcup, \delta, \{\mathbf{0}\}, \{\mathsf{halt\text{-}accept}\}, \{\mathsf{halt\text{-}reject}\})$ mit

- $Q = \{\mathbf{0}, \mathsf{add}, \mathsf{left}, \mathsf{right1}, \mathsf{right2}, \mathsf{incr}, \mathsf{overflow}, \mathsf{zeros}, \mathsf{halt\text{-}accept}, \mathsf{halt\text{-}reject}\}$,
- $\Sigma = \{0, 1, \mathbf{,}\}$,
- Übergangsfunktion δ siehe Abbildung 1,

eine deterministische Turingmaschine, welche einen Eingabestring $\vdash x\mathbf{,}y\sqcup^\omega$ liest, wobei x und y natürliche Zahlen in Binärkodierung sind, und die Zahl y für jedes Vorkommen von 1 in x jeweils um 1 inkrementiert. Der reguläre Ausdruck $\vdash(0+1)^*\mathbf{,}(0+1)^*\sqcup^+$ beschreibt zulässige Bandinhalte.

Beispiel: Für die Eingabe $\vdash 11010101\mathbf{,}10\sqcup^\omega$ berechnet die Turingmachine das Ergebnis $\vdash\vdash\vdash\vdash\vdash\vdash\vdash\vdash\vdash\mathbf{,}111\sqcup^\omega$.

a) Testen Sie die gegebene Turingmaschine händisch und mithilfe von Turing machine simulator. Der ausführbare Code ist in Abbildung 2 gegeben und ist zusätzlich unter diesem Link verfügbar.

b) Beschreiben Sie was die verschiedenen Zustände repräsentieren und wie die Turingmaschine funktioniert. Geben Sie ein Eingabe an, bei welcher die Turingmaschine im Zustand $\mathsf{halt\text{-}reject}$ hält.

c) Wieviele Schritte braucht M bis sie beim obigen Beispiel hält? Wieviele davon werden im Zuständ incr verbracht?

2) Modifizieren Sie die Turingmaschine M aus Aufgabe 1 mit dem entsprechenden Code in Abbildung 2 so, dass sie die selbe Funktion wie zuvor ausführt, jedoch mit dem Unterschied, dass y nun eine ternär kodierte natürliche Zahl ist.
Beispiel: Für die Eingabe $\vdash 01010100\mathbf{,}12\sqcup^\omega$ berechnet die Turingmaschine die Ausgabe $\vdash\vdash\vdash\vdash\vdash\vdash\vdash\vdash\vdash\mathbf{,}22\sqcup^\omega$.

3) Wir definieren die Sprache

$$\mathsf{HP}^{1000} := \{M\#x \mid M \text{ hält bei Eingabe } x \text{ innerhalb von 1000 Schritten}\} ,$$

welche alle möglichen Kodierungen einer Turingmaschine M und der Eingabe x enthält, sodass M bei der Eingabe x innerhalb von 1000 Schritten hält. Skizzieren Sie, wie man eine totale Turingmaschine K^{1000} konstruieren kann, welche die Sprache HP^{1000} akzeptiert.

4) Zeigen Sie, dass es eine *rekursive* Sprache L gibt, die aber von *keiner* TM innerhalb von 2^n Schritten entschieden werden kann (n ist die Länge der Eingabe).

Hinweis: Verwenden Sie die Methode der Diagonalisierung: Beschreiben Sie eine TM K, die bei Eingabe $x \in \{0,1\}^*$, 2^n (n=$\ell(x)$) Schritte von M_x bei Eingabe x simuliert (M_x ist die TM mit Code x). Wie muss der Endzustand von K abhängen von dem Zustand von M_x nach 2^n Schritten, damit M_x mit Sicherheit nicht $L(K)$ innerhalb von 2^n Schritten entscheidet? (Genauer: Es soll entweder $L(M_x) \neq L(K)$ gelten, oder, dass M_x nicht rechtzeitig hält.)

5) Wir wissen, dass es Sprachen gibt, die nicht rekursiv, wohl aber rekursiv aufzählbar oder co-rekursiv aufzählbar sind (z.B. das Halting Problem, bzw. dessen Komplement).

   a) Ist jede Sprache rekursiv aufzählbar oder co-rekursiv aufzählbar? (Decken die rekursiv aufzählbaren und die co-rekursiv aufzählbaren Sprachen alle möglichen Sprachen ab?)

   b) Eine Sprache L wird konstruiert, indem für jedes Element $x \in \{0,1\}^*$ durch einen Münzenwurf entschieden wird, ob $x \in L$ ist. Welches ist die Wahrscheinlichkeit, dass die so entstandene Sprache rekursiv / rekursiv aufzählbar / co-rekursiv aufzählbar ist?

   Hinweis: Wieviele rekursiv aufzählbare Sprachen gibt es? Wieviele co-rekrusiv aufzählbare? Wieviele Sprachen über dem Alphabet $\{0,1\}^*$?

$$\delta(\mathbf{0}, a) := \begin{cases} (\textsf{add}, \vdash, \mathrm{R}) & \text{wenn } a = \vdash, \\ (\textsf{halt-reject}, a, \mathrm{R}) & \text{sonst.} \end{cases}$$

$$\delta(\textsf{add}, a) := \begin{cases} (\textsf{add}, \vdash, \mathrm{R}) & \text{wenn } a = 0, \\ (\textsf{right1}, \vdash, \mathrm{R}) & \text{wenn } a = 1, \\ (\textsf{halt-accept}, \mathbf{,}, \mathrm{R}) & \text{wenn } a = \mathbf{,}, \\ (\textsf{halt-reject}, a, \mathrm{R}) & \text{sonst.} \end{cases}$$

$$\delta(\textsf{right1}, a) := \begin{cases} (\textsf{right1}, a, \mathrm{R}) & \text{wenn } a \in \{0, 1\}, \\ (\textsf{right2}, \mathbf{,}, \mathrm{R}) & \text{wenn } a = \mathbf{,}, \\ (\textsf{halt-reject}, a, \mathrm{R}) & \text{sonst.} \end{cases}$$

$$\delta(\textsf{right2}, a, \mathrm{R}) := \begin{cases} (\textsf{right2}, a, \mathrm{R}) & \text{wenn } a \in \{0, 1\}, \\ (\textsf{inc}, \sqcup, \mathrm{L}) & \text{wenn } a = \sqcup, \\ (\textsf{halt-reject}, a, \mathrm{R}) & \text{sonst.} \end{cases}$$

$$\delta(\textsf{inc}, a) := \begin{cases} (\textsf{left}, 1, \mathrm{L}) & \text{wenn } a = 0, \\ (\textsf{inc}, 0, \mathrm{L}) & \text{wenn } a = 1, \\ (\textsf{overflow}, \mathbf{,}, \mathrm{R}) & \text{wenn } a = \mathbf{,}, \\ (\textsf{halt-reject}, a, \mathrm{R}) & \text{sonst.} \end{cases}$$

$$\delta(\textsf{overflow}, a) := \begin{cases} (\textsf{zeros}, 1, \mathrm{R}) & \text{wenn } a = 0, \\ (\textsf{left}, 1, \mathrm{R}) & \text{wenn } a = \sqcup, \\ (\textsf{halt-reject}, a, \mathrm{R}) & \text{sonst.} \end{cases}$$

$$\delta(\textsf{zeros}, a) := \begin{cases} (\textsf{zeros}, 0, \mathrm{R}) & \text{wenn } a = 0, \\ (\textsf{left}, 0, \mathrm{R}) & \text{wenn } a = \sqcup, \\ (\textsf{halt-reject}, a, \mathrm{R}) & \text{sonst.} \end{cases}$$

$$\delta(\textsf{left}, a) := \begin{cases} (\textsf{add}, \vdash, \mathrm{R}) & \text{wenn } a = \vdash, \\ (\textsf{left}, a, \mathrm{L}) & \text{sonst.} \end{cases}$$

$$\delta(\textsf{halt-accept}, a) := (\textsf{halt-accept}, a, \mathrm{R})$$

$$\delta(\textsf{halt-reject}, a) := (\textsf{halt-reject}, a, \mathrm{R})$$

Abbildung 1: Definition der Funktion δ.

```
; machine starts at state 0.
0 ⊢ ⊢ r add
0 * * r halt-reject
;
add 0 ⊢ r add
add 1 ⊢ r right1
add , , r halt-accept
add * * r halt-reject
;
right1 0 0 r right1
right1 1 1 r right1
right1 , , r right2
right1 * * r halt-reject
;
right2 0 0 r right2
right2 1 1 r right2
right2 _ _ l inc
right2 * * r halt-reject
;
inc 0 1 l left
inc 1 0 l inc
inc , , r overflow
inc * * r halt-reject
;
overflow 0 1 r zeros
overflow _ 1 r left
overflow * * r halt-reject
;
zeros 0 0 r zeros
zeros _ 0 r left
zeros * * r halt-reject
;
left ⊢ ⊢ r add
left * * l left
;
halt-accept * * r halt-accept
;
halt-reject * * r halt-reject
```

Abbildung 2: Code für Turing machine simulator.