Technische Universität München
Fakultät für Informatik
Dr. M. Schneider, Dr. M. Pizka,
Dr. A. Vilbig, Dr. R. Letz,
G. Bauer, B. Zhang

SS 2003
Lösungsvorschläge zu Blatt 5
16. Mai 2003

Übungen zur Vorlesung Einführung in die Informatik IV

Aufgabe 19 Turing Maschine zur Addition von Dualzahlen

(a) Lösungsidee:

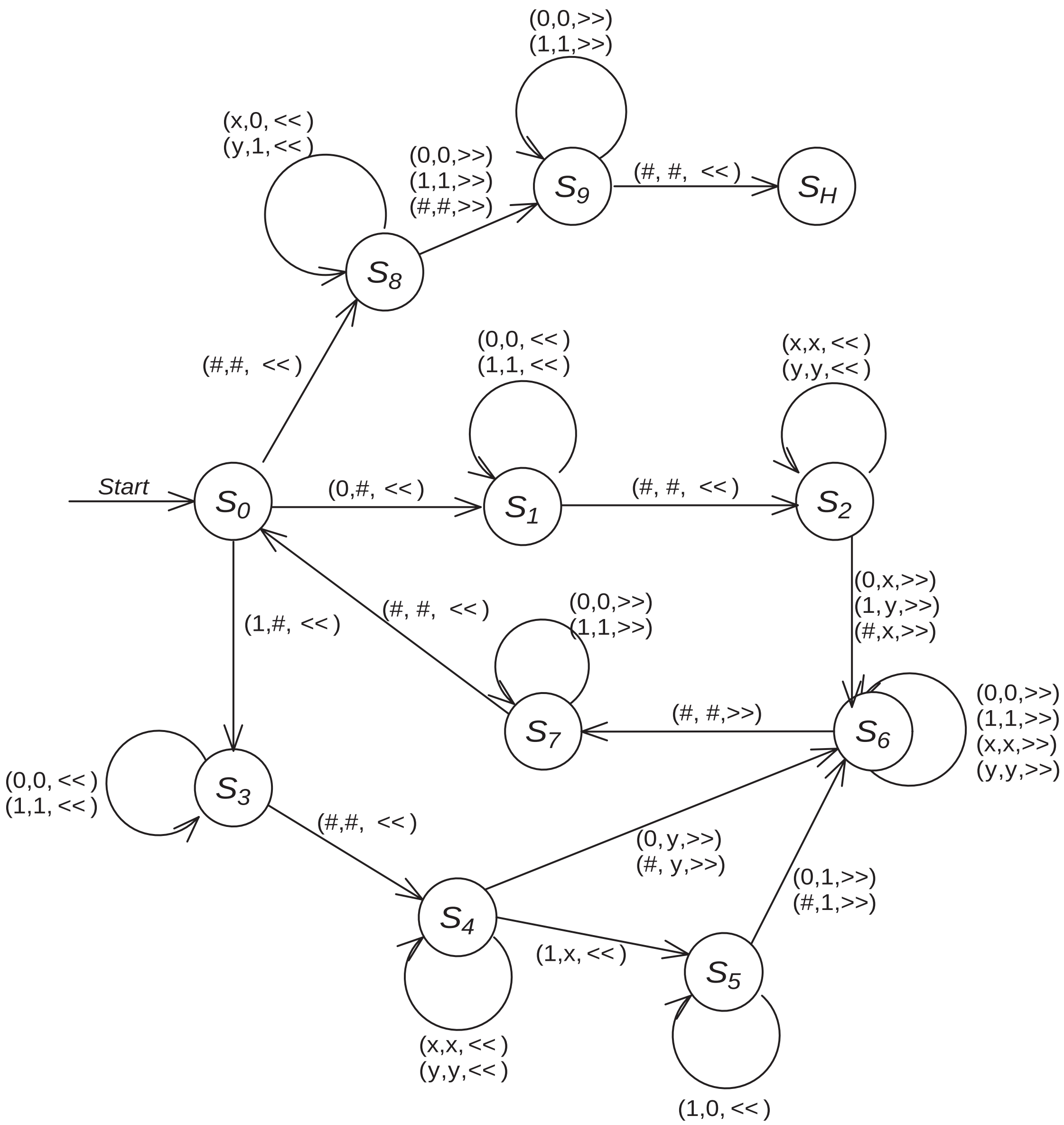
- Auf dem Band sind die beiden Dualzahlen a und b durch ein # getrennt (b sei die rechte und a die linke Zahl).
- Zu Beginn steht der Kopf auf dem Blank rechts vom niedrigsten Bit der Zahl b.
- Dann wird das niedrigste Bit von b gelesen, ein entsprechender Zustand eingenommen und das Bit gelöscht.
- Anschließend wandert der Kopf zum niedrigsten Bit von a und überschreibt dieses mit x oder y, je nachdem ob die Summe der beiden niedrigsten Bits 0 oder 1 (mod 2) ist. Ein Übertrag wird hierbei gegebenenfalls durch einen entsprechenden Zustand signalisiert.
- Im Falle eines Übertrags inkrementiert man die Dualzahl links von x bzw. y
- Dann wandert der Kopf wieder nach rechts und sucht das niedrigste Bit von b. Dieses wird nun auf die beschriebene Art zu dem ersten Bit, das links von x bzw. y steht addiert.
- Der Vorgang wiederholt sich bis alle Bits von b verarbeitet und damit gelöscht sind.
- Schließlich müssen alle x mit 0 und alle y mit 1 überschrieben werden.

(b) Die Übergangsfunktion,Übergangsdiagramm und Beispielkonfiguration

Übergangsfunktion:

δ	0	1	#(*Blank*)	*x*	*y*
s_0	(s_1,#,$\leftarrow$)	(s_3,#,$\leftarrow$)	(s_8,#,$\leftarrow$)	—	—
s_1	(s_1,0,$\leftarrow$)	(s_1,1,$\leftarrow$)	(s_2,#,$\leftarrow$)	—	—
s_2	(s_6,x,$\rightarrow$)	(s_6,y,$\rightarrow$)	(s_6,x,$\rightarrow$)	(s_2,x,$\leftarrow$)	(s_2,y,$\leftarrow$)
s_3	(s_3,0,$\leftarrow$)	(s_3,1,$\leftarrow$)	(s_4, #,$\leftarrow$)	—	—
s_4	(s_6,y,$\rightarrow$)	(s_5,x,$\leftarrow$)	(s_6,y,$\rightarrow$)	(s_4,x,$\leftarrow$)	(s_4,y,$\leftarrow$)
s_5	(s_6,1,$\rightarrow$)	(s_5,0,$\leftarrow$)	(s_6,1,$\rightarrow$)	—	—
s_6	(s_6,0,$\rightarrow$)	(s_6,1,$\rightarrow$)	(s_7,#,$\rightarrow$)	(s_6,x,$\rightarrow$)	(s_6,y,$\rightarrow$)
s_7	(s_7,0,$\rightarrow$)	(s_7,1,$\rightarrow$)	(s_0,#,$\leftarrow$)	—	—
s_8	(s_9,0,$\rightarrow$)	(s_9,1,$\rightarrow$)	(s_9,#,$\rightarrow$)	(s_8,0,$\leftarrow$)	(s_8,1,$\leftarrow$)
s_9	(s_9,0,$\rightarrow$)	(s_9,1,$\rightarrow$)	(s_{Halt},#,$\leftarrow$)	—	—
s_{Halt}	—	—	—	—	—

Übergangsdiagramm:



Beispielkonfiguration:(a=110,b=11)

$(s_0,\#110\#1,1,\#) \rightarrow (s_3,\#110\#,1,\#) \rightarrow (s_3,\#110,\#,1\#) \rightarrow (s_4,\#11,0,\#1\#) \rightarrow (s_6,\#11y,\#,1\#)$
$\rightarrow (s_7,\#11y\#,1,\#) \rightarrow (s_7,\#11y\#1,\#,\#) \rightarrow (s_0,\#11y\#,1,\#) \rightarrow (s_3,\#11y,\#,\#) \rightarrow (s_4,\#11,y,\#\#)$
$\rightarrow (s_4,\#1,1,y\#\#) \rightarrow (s_5,\#,1,xy\#\#) \rightarrow (s_5,\#,\#,0xy\#\#) \rightarrow (s_6,\#1,0,xy\#\#) \rightarrow (s_6,\#10,x,y\#\#)$
$\rightarrow (s_6,\#10x,y,\#\#) \rightarrow (s_6,\#10xy,\#,\#) \rightarrow (s_7,\#10xy\#,\#,\#) \rightarrow (s_0,\#10xy,\#,\#) \rightarrow (s_8,\#10x,y,\#\#)$
$\rightarrow (s_8,\#10,x,1\#\#) \rightarrow (s_8,\#1,0,01\#\#) \rightarrow (s_9,\#10,0,1\#\#) \rightarrow (s_9,\#100,1,\#\#) \rightarrow (s_9,\#1001,\#,\#)$
$\rightarrow (s_{Halt},\#100,1,\#)$

Aufgabe 20 Registermaschinen

(a) Wir übertragen die Zuweisung von Register s_i an Register s_j ($i \neq j$) von der Notation $s_j := s_i$ auf die n-RM:

n-RM-Programm simuliert folgendes Programm:

$while_j\ (pred_j);$	$s_j := 0;$
$while_i\ (succ_j;\ pred_i);$	$(s_j, s_i) := (s_i, 0)$

Dieses RM-Programm zerstört den Inhalt von Register s_i. Soll dieser Wert jedoch am Programmende unverändert vorliegen, muss ein Hilfsregister verwendet werden, um ihn zwischenzuspeichern. Dies führt zu folgendem RM-Programm mit s_h als Hilfsregister:

n-RM-Programm simuliert folgendes Programm:

$while_j\ (pred_j);$	$s_j := 0;$
$while_h\ (pred_h);$	$s_h := 0;$
$while_i\ (succ_j;\ succ_h;\ pred_i)$	$(s_j, s_h, s_i) := (s_i, s_i, 0);$
$while_h\ (succ_i;\ pred_h):$	$(s_i, s_h) := (s_h, 0)$

(b) Wir übertragen die bedingte Zuweisung

if $s_m = 0$ **then** $s_j := s_i$ **else** $s_j := s_k$ **fi**

auf die n-RM.
Dazu formen wir das Programm zuerst um, um näher an die Konstrukte der RM zu gelangen. Insbesondere müssen wir die Abfrage $s_m = 0$ ohne bedingte Anweisung ausdrücken:

switch := 1;
while $s_m > 0$ **do** $s_j := s_k$; switch := 0; $s_m := 0$ **od**;
while *switch* > 0 **do** $s_j := s_i$; switch := 0 **od**

Dies übertragen wir nun in ein RM-Programm (der Einfachheit halber ohne Retten von s_m, s_i, s_k). Dabei verwenden wir s_w als Register für *switch*.

n-RM-Programm simuliert folgendes Programm:

$while_w\ (pred_w);\ succ_w;$	$s_w := 1;$
$while_j\ (pred_j);$	$s_j := 0;$
$while_m$ (	**while** $s_m > 0$ **do**
$while_k\ (succ_j;\ pred_k);$	$(s_j, s_k) := (s_k, 0);$
$pred_w;$	$s_w := 0$; (* da vorher $s_w = 1$ gilt *)
$while_m\ (pred_m)$	$s_m := 0$
)	**od**;
$while_w$ (	**while** $s_w > 0$ **do**
$while_i\ (succ_j;\ pred_i);$	$(s_j, s_i) := (s_i, 0);$
$pred_w$	$s_w := 0$; (* da vorher $s_w = 1$ gilt *)
)	**od**

(c) **(H)** Die totale Subtraktion *sub* ist definiert durch

$$sub(x,y) = \begin{cases} x - y, & \text{falls } x \geq y \\ 0, & \text{falls } x < y \end{cases}$$

Ein n-RM-Programm, das $s_j := sub(s_j, s_k)$ simuliert, ist:

$while_k\ (pred_j;\ pred_k)$

Dieses Programm leistet das Gewünschte, weil $pred_j$ total definiert ist:

$(s, pred_j) \rightarrow ((s_1, \ldots, s_{j-1}, k, s_{j+1}, \ldots, s_n), \varepsilon)$

mit

$$k = \begin{cases} s_j - 1, & \text{falls } s_j > 0 \\ 0, & \text{falls } s_j = 0 \end{cases}$$

Im obigen RM-Programm werden die Inhalte der Register s_j und s_k zerstört. Folgendes RM-Programm simuliert den Befehl

$s_j := sub(s_i, s_k)$

unter Rettung der Registerinhalte von s_i und s_k (in den zusätzlichen Hilfsregistern s_h und s_l):

n-RM-Programm simuliert folgendes Programm:

$while_j\,(pred_j);$	$s_j := 0;$
$while_h\,(pred_h);$	$s_h := 0;$
$while_l\,(pred_l);$	$s_l := 0;$
$while_i\,(succ_j;\, succ_h;\, pred_i);$	$(s_j, s_h, s_i) := (s_i, s_i, 0);$
$while_k\,(pred_j;\, succ_l;\, pred_k);$	$(s_j, s_l, s_k) := (sub(s_j, s_k), s_k, 0);$
$while_h\,(succ_i;\, pred_h);$	$(s_i, s_h) := (s_h, 0);$
$while_l\,(succ_k;\, pred_l)$	$(s_k, s_l) := (s_l, 0);$

Aufgabe 21 Turingmaschinen, Bezug zu formalen Sprachen

(a) Turingmaschine, die der Grammatik mit $\rightarrow = \{ab \rightarrow Z, aZb \rightarrow Z\}$ entspricht

Die akzeptierte Sprache der Grammatik ist offenbar $\{a^n b^n : n > 0\}$.

Für den Bau einer entsprechenden Turingmaschine $TM = (T, S, \delta, s_0, S_Z)$ nehmen wir an, der Kopf der Maschine stehe am Anfang auf dem ersten (linkesten) Buchstaben des Eingabewortes.

Idee der Simulation:

- Beim Lesen von ab: Überschreiben von b mit Z sowie Löschen von a vom linken Wortende (= Bandende)
- Beim Lesen von aZb: Löschen von b am rechten Bandende, danach Löschen von a vom linken Bandende
- Akzeptieren beim Lesen von Z allein auf Band

Es seien $T = \{a, b, Z\}$, $S = \{s_0, s_1, s_a, s_l, s'_l, s_r, s'_r, s_z\}$, $S_Z = \{s_z\}$.

Die Übergangsrelation δ enthalte genau die folgenden Tupel:

$((s_0, a), (s_a, a, \gg))$	a am Wortanfang gelesen (danach weiter nach rechts)
$((s_a, a), (s_a, a, \gg))$	a^+ gelesen (danach weiter nach rechts)
$((s_a, b), (s_l, Z, \ll))$	a^+b gelesen und b überschrieben (danach a ganz links löschen)
$((s_a, Z), (s_z, Z, \gg))$	a^+Z gelesen (danach weiter nach rechts)
$((s_z, b), (s_r, Z, \gg))$	a^+Zb gelesen (b ganz rechts, danach a ganz links löschen)

Löschen von a am linken Bandende (danach in Anfangszustand):
$((s_l, a), (s_l, a, \ll))$, $((s_l, b), (s_l, b, \ll))$, $((s_l, Z), (s_l, Z, \ll))$, $((s_l, \#), (s'_l, \#, \gg))$, $((s'_l, a), (s_0, \#, \gg))$

Löschen von b am rechten Bandende (danach a links löschen):
$((s_r, b), (s_r, b, \gg))$, $((s_r, \#), (s'_r, \#, \ll))$, $((s'_r, b), (s_l, \#, \ll))$

Akzeptieren:
$((s_0,Z),(s_1,Z,\gg))$, $((s_1,\#),(s_z,\#,\downarrow))$

Bemerkung: Die angegebene Turingmaschine ist deterministisch.

(b) Allgemeines Verfahren von Chomsky-0-Grammatik zu Turingmaschine

Konstruktion einer *nichtdeterministischen* Turingmaschine (T,S,δ,s_0,S_Z) mit folgenden Eigenschaften:

(1) Wir nehmen an, am Anfang stehe genau das zu akzeptierende Wort auf dem Band

(2) Im Anfangszustand s_0 soll beliebig nach rechts oder links gelaufen werden können (Nichtdeterminismus), d.h. δ enthalte für alle a aus dem Alphabet der Grammatik die Tupel: $((s_0,a),(s_0,a,\ll))$, $((s_0,a),(s_0,a,\gg))$

(3) Der Versuch der Anwendung einer Regel r der Grammatik wird simuliert mittels interner Zustände $r_1,\dots,r_l$, die das Durchlaufen der Buchstaben in der Regel speichern. Jede Regel wird also buchstabenweise gelesen und die rechte Seite geschrieben.

(4) Von s_0 aus kann durch Zustandswechsel in eine beliebige Regel r der Grammatik „eingestiegen" werden und damit der Versuch der Anwendung der Regel simuliert werden:
Für alle a aus dem Bandalphabet enthalte δ das Tupel $((s_0,a),(r_1,a,\downarrow))$, wobei r_1 der Anfangszustand der Regelanwendung sei

(5) Wenn auf dem Band nur noch Z steht, wird akzeptiert

Wir betrachten nun den schwierigen Teil (3). Gegeben also eine beliebige Grammatik-Regel

$$r = u\circ v\circ w \to u\circ v'\circ w, \text{ wobei}$$

$$u = u_1\cdots u_m,\ \ w = w_1\cdots w_n,\ \ v = v_1\cdots v_p,\ \ v' = v'_1\cdots v'_q$$

Fall 1: $|v| = |v'|$, d.h. $p = q$. Dann können wir einfach die linke Seite der Regel mit der rechten überschreiben. Dazu geben wir die folgenden Übergänge mittels für jede Regel neuer interner Zustände $r_1,\dots,r_{m+p+n}$ an:

$((r_1,u_1),(r_2,u_1,\gg)),\dots,((r_m,u_m),(r_{m+1},u_m,\gg))$,
$((r_{m+1},v_1),(r_{m+2},v'_1,\gg)),\dots,((r_{m+p},v_n),(r_{m+p+1},v'_n,\gg))$,
$((r_{m+p+1},w_1),(r_{m+p+2},w_1,\gg)),\dots,((r_{m+p+n},w_n),(s_0,w_n,\gg))$

Fall 2: $|v| > |v'|$, d.h. es gibt ein $k > 0 : p = q+k$

Problem: Nachdem $u\circ(v_1\cdots v_q)$ gelesen wurde, muss das Teilwort $v_{q+1}\cdots v_p$ verschluckt werden.

Lösungsidee: Beim Lesen des Restwortes $v_{q+1}\cdots v_p$ schreiben wir jeweils #. Wenn wir v gelesen haben, verschieben wir den gesamten rechten Bandinhalt R um k Positionen nach links.

Verschieben des rechten Bandinhalts R um k nach links:

- Merken des ersten Buchstabens a von R und Stelle mit # überschreiben
- k Schritte nach links
- a schreiben und $k+1$ Schritte nach rechts

- wenn # auf Band, stopp (Verschiebung ist abgeschlossen)
- ansonsten Verfahren wiederholen

Nun muss man noch die Stelle wiederfinden, an der das rechte Kontextwort w beginnt (vorher markieren!) und fährt fort wie in Fall 1. (Markieren von Stellen: Verdoppelung des Alphabets um jeweils markierte Variante eines Buchstabens)

Fall 3: $|v| < |v'|$, d.h. es gibt ein $k > 0 : p = q - k$

Problem: Nachdem $u \circ (v_1 \cdots v_p)$ gelesen wurde, muss das Teilwort $v'_{p+1} \cdots v'_q$ eingeschoben werden.

Lösung analog zu Fall 3 durch Verschiebung des gesamten Bandinhaltes rechts von der aktuellen Stelle um k Positionen nach rechts. Danach geht man wieder zur Unterbrechungsstelle zurück. Nun muss man aber $v'_{p+1} \cdots v'_q$ schreiben, bevor man mit dem Lesen von w wie in Fall 1 fortfährt.

Aufgabe 22 (P) Deterministische Turing-Maschine

Die vorgestellte Lösung ist verhältnismäßig umfangreich, jedoch wird ein Großteil der benötigten Klassen als Vorlage im WWW zur Verfügung gestellt. Im folgenden sind daher nur die nicht vorgegebenen Teile der Klassen `TMTape` und `TuringMachine` aufgeführt.

Teilaufgabe a)

```
/**
 * Represents a left-bounded input/output tape for a Turing Machine
 */
public class TMTape {
    private List tape = new Vector();

    // Read symbol at position pos with 0 <= pos < tape.size
    public TMSymbol read(int pos) {
        TMSymbol result = null;
        try {
            result = (TMSymbol) tape.get(pos);
        } catch (IndexOutOfBoundsException ex) {
            System.out.println("*** Could not read from position " +
                               pos + ", index out of bounds!");
            ex.printStackTrace();
        }
        return result;
    }

    // Write symbol at position pos, creating new space on the right if necessary
    public void write(int pos, TMSymbol symbol) {
        try {
            if (0 <= pos && pos < tape.size())
                tape.set(pos, symbol);
            if (pos == tape.size())
                tape.add(pos, symbol);
        } catch (IndexOutOfBoundsException ex) {
            System.out.println("*** Could not write symbol " + symbol +
                               " to position " + pos + ", index out of bounds!");
```

```
            ex.printStackTrace();
        }
    }

    // Clear the complete tape for new input/output
    public void clear() {
        tape.clear();
    }

    // Formatted output of the tape content
    public String toString() {
        String result = new String("|");
        ListIterator lit = tape.listIterator();
        while (lit.hasNext()) {
            result += lit.next() + "|";
        }
        return result;
    }
}
```

Teilaufgabe b)

```
/**
 * Implements a deterministic Turing Machine (TM) with a single, left-bounded tape
 */
public class TuringMachine {
    // Integer constants for head movements
    public static final int HOLD = 0, LEFT = 1, RIGHT = 2;

    // Tabular representation of deterministic transition matrix
    private TMResult[][] transitionMatrix;
    private TMTape tape;
    private TMState currentState;
    private int currentPosition;
    private Set states, symbols;
    // Utility data structure to map names to instances for states and symbols
    private Map nameToState, nameToSymbol;
    private int numStates, numSymbols;

    // Create a new instance for a TM with maximal number of states and symbols
    public TuringMachine(int maxStates, int maxSymbols) {
        tape = new TMTape();
        states = new HashSet(); symbols = new HashSet();
        nameToState = new HashMap(); nameToSymbol = new HashMap();
        numStates = 0; numSymbols = 0;
        transitionMatrix = new TMResult[maxStates][maxSymbols];
    }

    // Retrieve tape for preparing of input
    public TMTape getTape() {
```

```
        return tape;
    }

    // Add a new state to the TM if it does not already exist
    public TMState addState(String name, boolean isStart) {
        TMState s = (TMState) nameToState.get(name);
        if (s == null) {
            s = new TMState(name, isStart, numStates);
            states.add(s);
            nameToState.put(name, s);
            numStates++;
        }
        return s;
    }

    // Add a new symbol to the TM if it does not already exist
    public TMSymbol addSymbol(String symbol) {
        TMSymbol s = (TMSymbol) nameToSymbol.get(symbol);
        if (s == null) {
            s = new TMSymbol(symbol, numSymbols);
            symbols.add(s);
            nameToSymbol.put(symbol, s);
            numSymbols++;
        }
        return s;
    }

    // Add a new transition to the TM,
    // all referenced states & symbols have to exist already!
    public void addTransition(String fromState, String inputSymbol,
                              String toState, String outputSymbol, int direction) {
        TMState origin = (TMState) nameToState.get(fromState);
        TMState target = (TMState) nameToState.get(toState);
        TMSymbol input = (TMSymbol) nameToSymbol.get(inputSymbol);
        TMSymbol output = (TMSymbol) nameToSymbol.get(outputSymbol);
        TMResult result = new TMResult(target, output, direction);
        transitionMatrix[origin.getId()][input.getId()] = result;
    }

    // Initialize TM with start state and given position on the tape
    public void init(int pos) {
        currentPosition = pos;
        Iterator stateIt = states.iterator();
        while (stateIt.hasNext()) {
            TMState s = (TMState) stateIt.next();
            if (s.isStart()) {
                currentState = s;
                break;
            }
        }
    }
```

```
    // Execute TM after previous initialization, providing trace output of
    // current state, position and tape content if desired
    public void execute(boolean doTrace) {
        boolean running = true;
        while (running) {
            if (doTrace) {
                for (int i = 0; i < currentPosition; i++)
                    System.out.print("  ");
                System.out.print(" " + currentState.getId() + "\n");
                System.out.println(tape);
            }
            running = step();
        }
    }

    // Perform a singe step of the TM,
    // returns true if the TM is able continue or false otherwise
    private boolean step() {
        TMSymbol symbol = tape.read(currentPosition);
        TMResult result = transitionMatrix[currentState.getId()][symbol.getId()];
        if (result != null) {
            currentState = result.getState();
            tape.write(currentPosition, result.getSymbol());
            switch (result.getHeadDirection()) {
                case LEFT: currentPosition--; break;
                case RIGHT: currentPosition++; break;
                case HOLD: ; // Do nothing
            }
            return true;
        }
        return false;
    }
}
```