Universelle Turingmaschine

nach Minsky

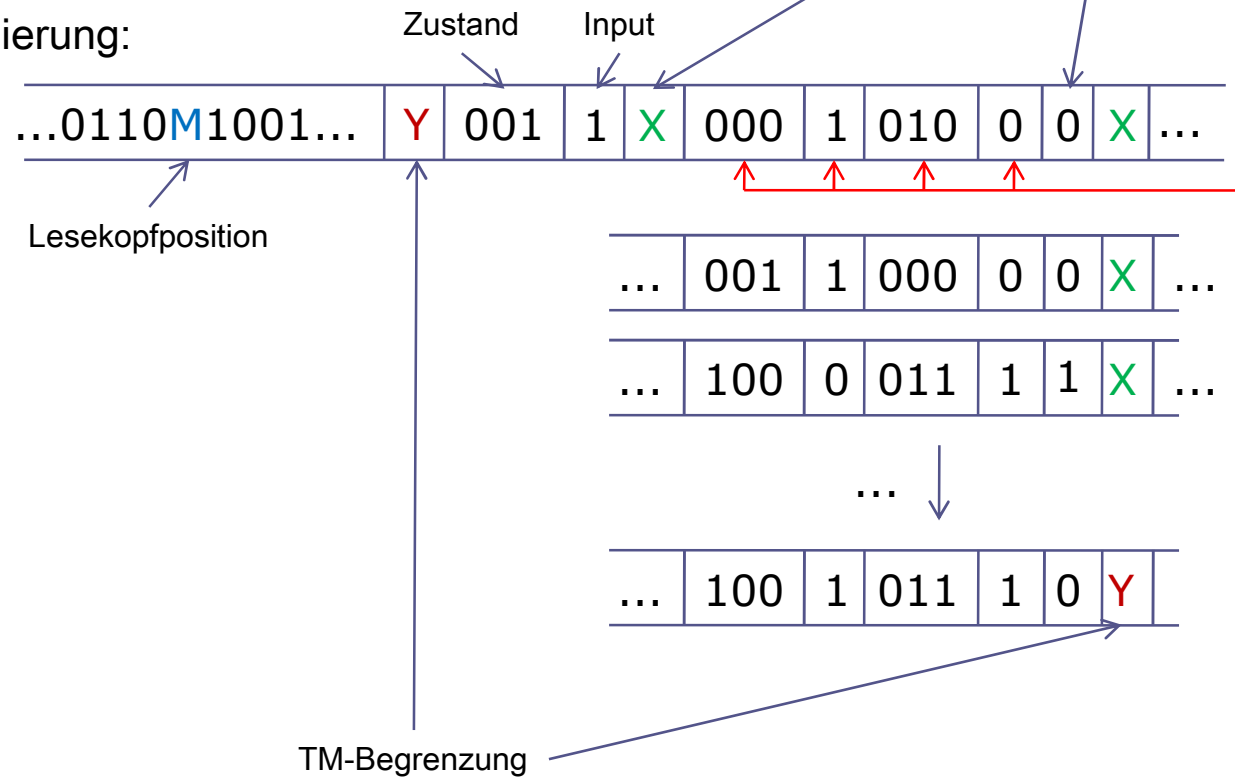
Übersicht zur Vorlesung

- Turingmaschine (TM)
- Universelle Turingmaschine (UTM)
- Haskell-Implementation

Aufbau der UTM

Turingmaschine als Eingabe:

Codierung:



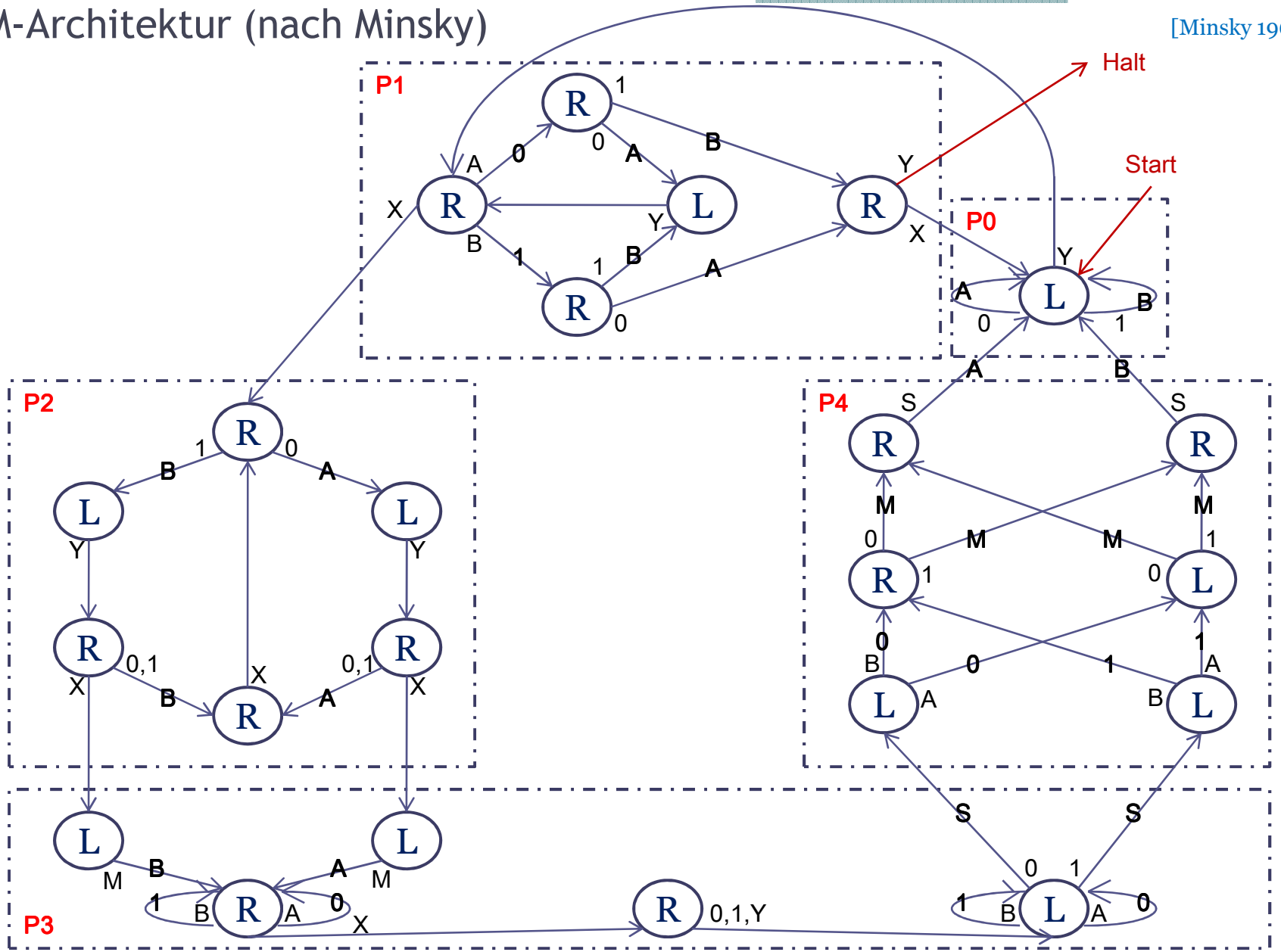
TM:

s	i	s‘	o	a
000	1	010	0	0
001	1	000	0	0
100	0	011	1	1

...

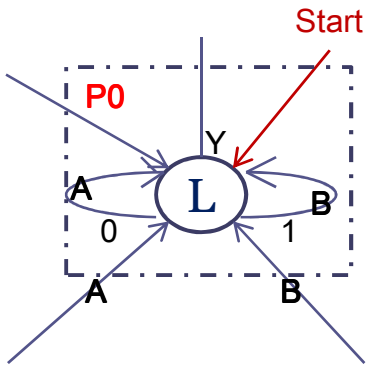
UTM-Architektur (nach Minsky)

[Minsky 1967]

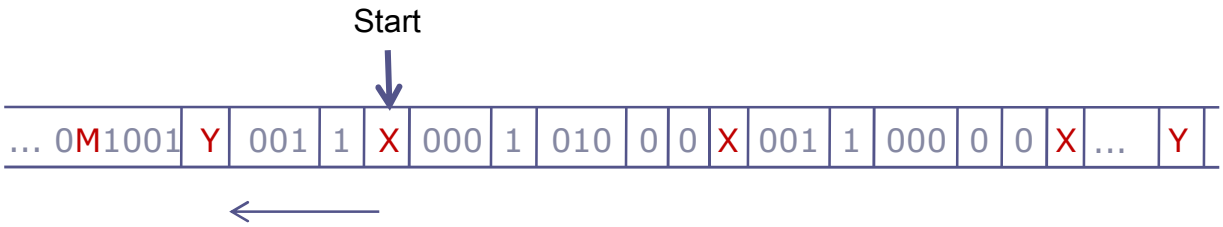


UTM-Architektur (nach Minsky)

[Minsky 1967]



Verarbeitung der UTM I

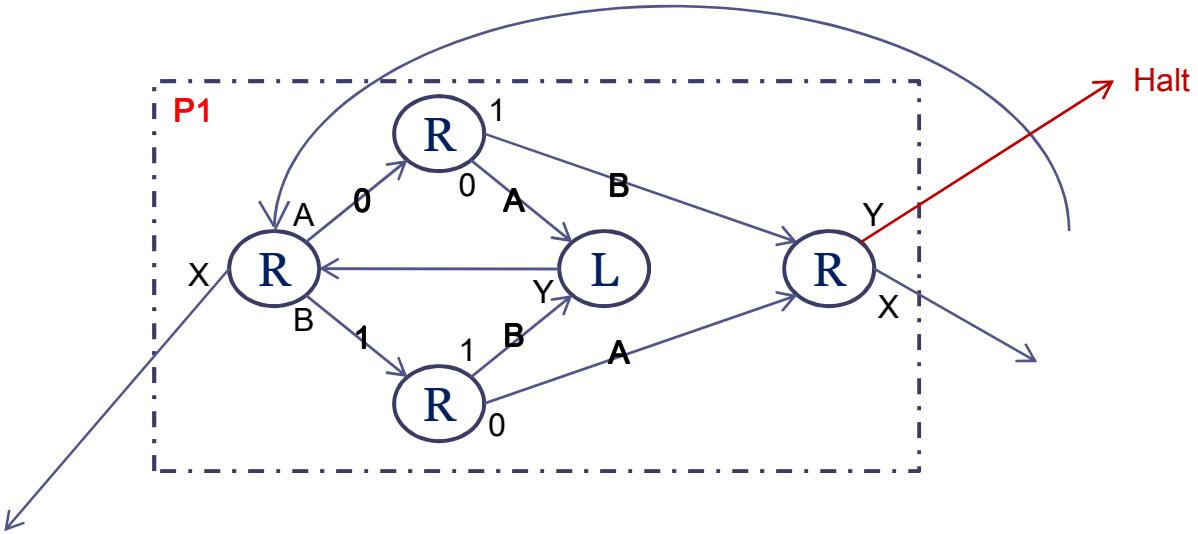


P0:

- gehe nach links bis Y und ersetze 0 mit A und 1 mit B

UTM-Architektur (nach Minsky)

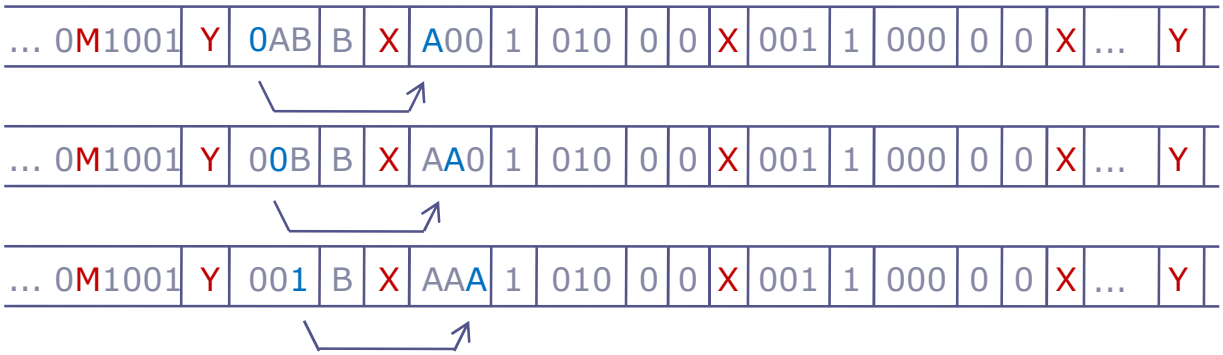
[Minsky 1967]



Verarbeitung der UTM II

P1:

- gehe nach rechts und überprüfe die Signatur von Zustand/Input mit verfügbaren 0-1-Mustern, ersetze dabei das erste Zeichen A durch 0 oder B durch 1 und prüfe erstes Auftreten

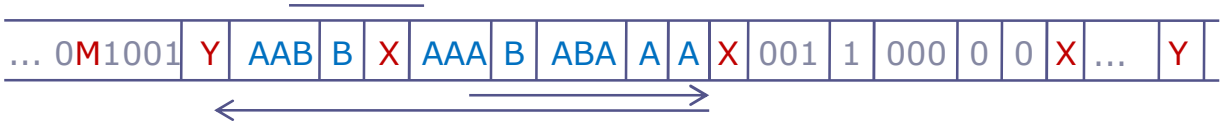


Match

Match

Missmatch

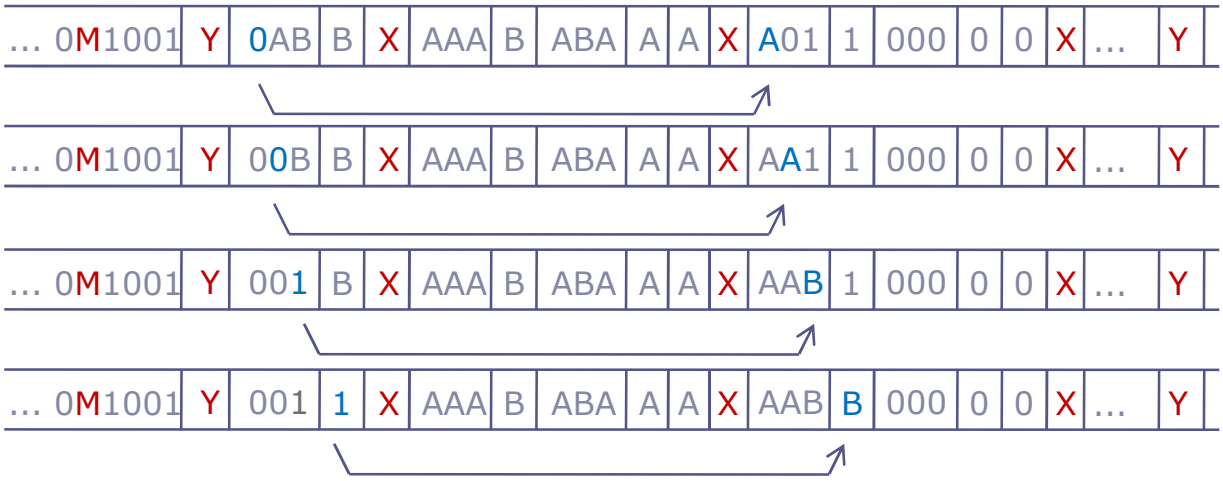
- überspringe alle 0-1 bis zum folgenden X
- laufe rückwärts zu Y und ersetze wieder 0 durch A und 1 durch B



Verarbeitung der UTM III

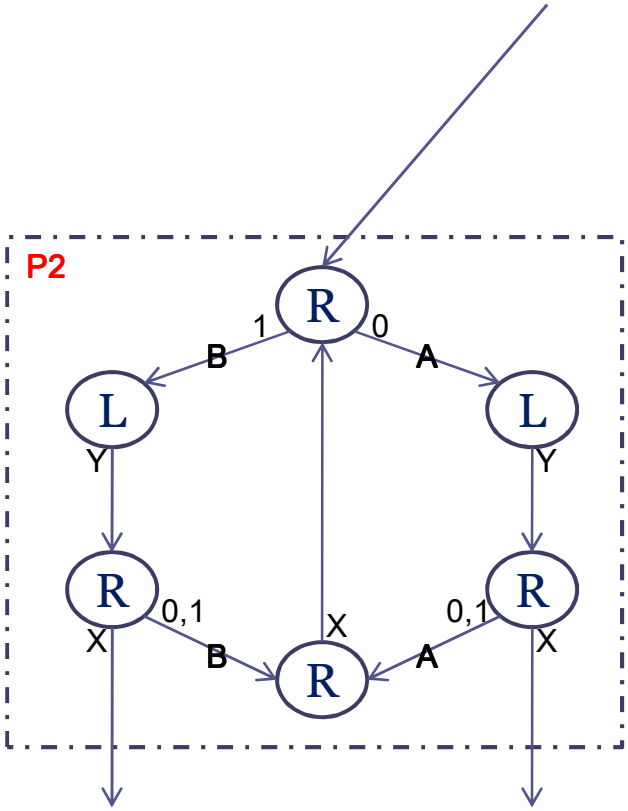
P1:

- gehe nach rechts und überprüfe die Signatur von Zustand/Input mit verfügbaren 0-1-Mustern, ersetze dabei das erste Zeichen A durch 0 oder B durch 1 und prüfe erstes Auftreten



UTM-Architektur (nach Minsky)

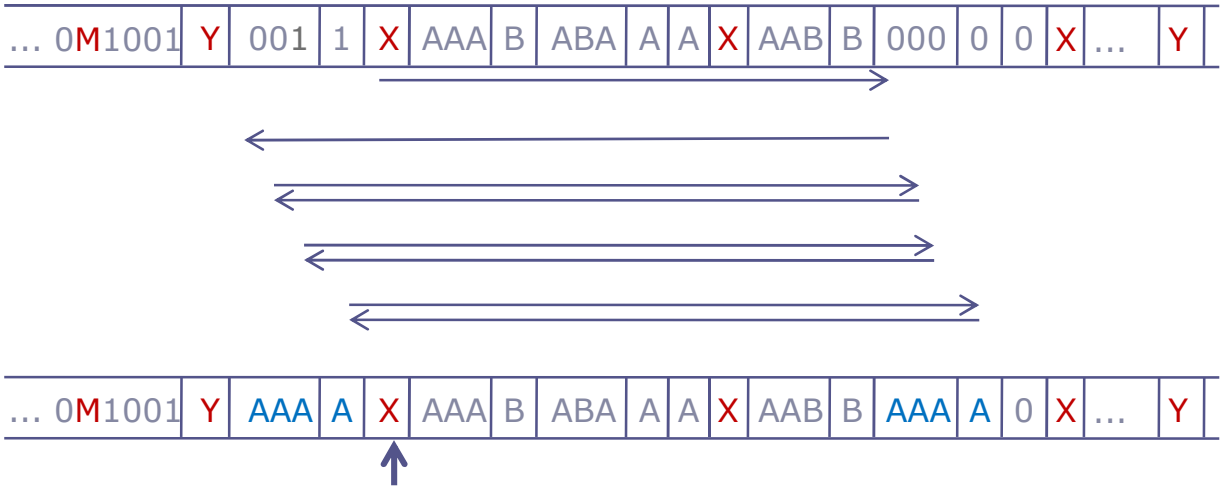
[Minsky 1967]



Verarbeitung der UTM IV

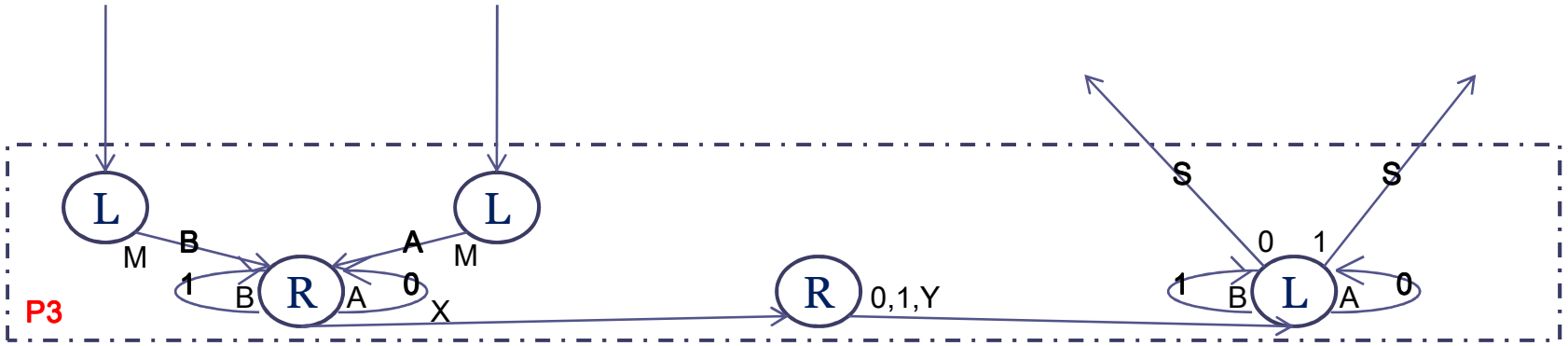
P2:

- Kopieren des Folgezustands und Output



UTM-Architektur (nach Minsky)

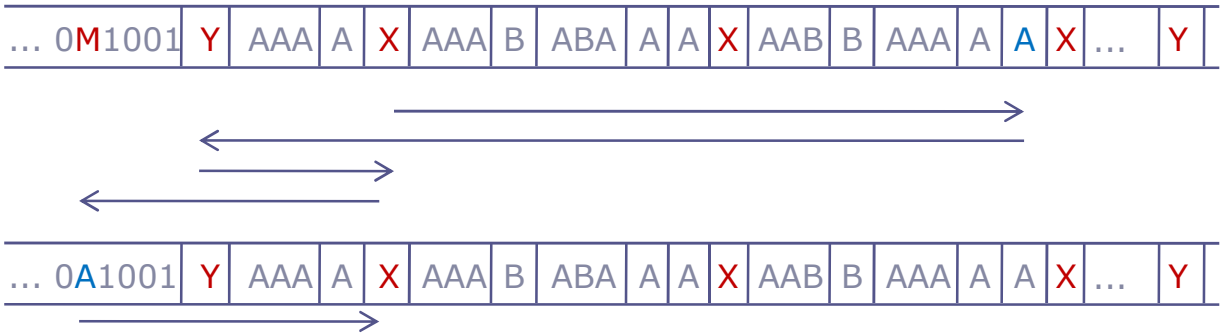
[Minsky 1967]



Verarbeitung der UTM V

P3:

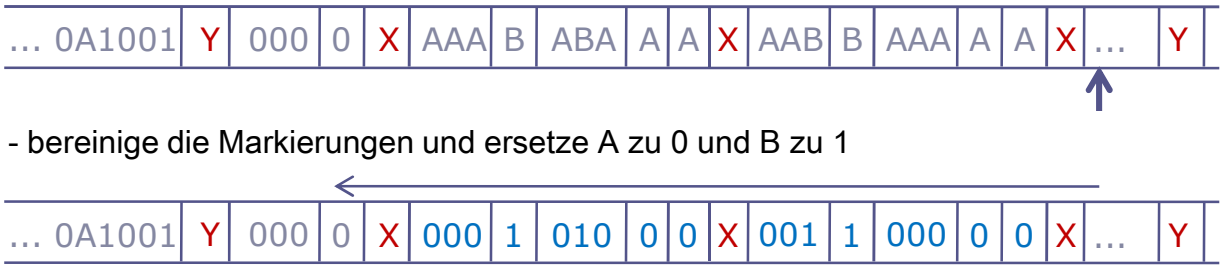
- Folgezustand und Output wurden nach vorn verlagert
- Aktion wird im Speicher gehalten und dieser zu M gebracht



- gehe nach rechts bis X und ersetze 0 A durch 0 und B durch 1

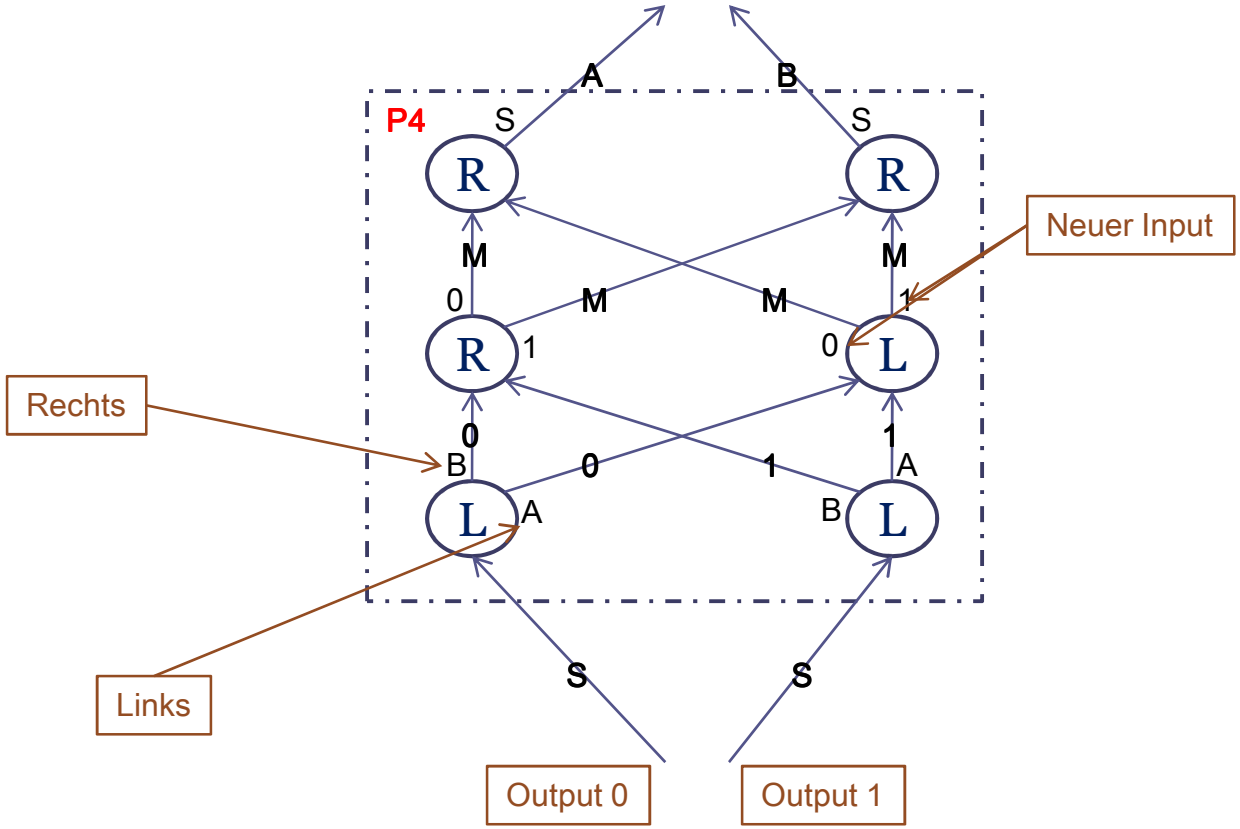
- gehe nach rechts, bis 1, 0 oder Y auftauchen

Verarbeitung der UTM VI



UTM-Architektur (nach Minsky)

[Minsky 1967]



Verarbeitung der UTM VI

P4:

- Output mit S markieren und bis A oder B nach links wandern

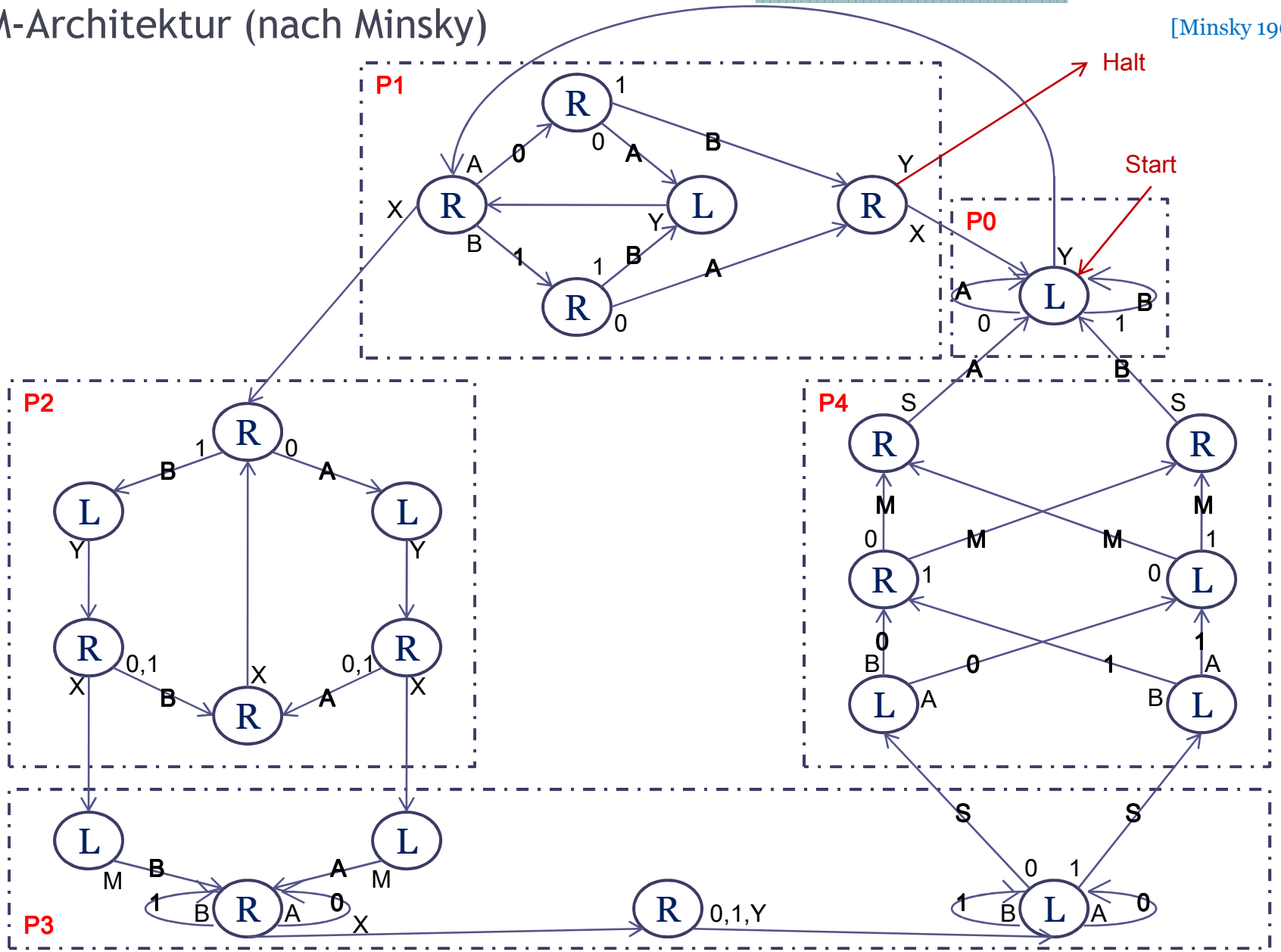
- Ja nach Inhalt wird Output geschrieben und M nach links oder rechts versetzt

- ersetze den Inhalt, der mit M ersetzt wurde mit S (optimiert mit A oder B)

- weiter gehts wieder mit P0

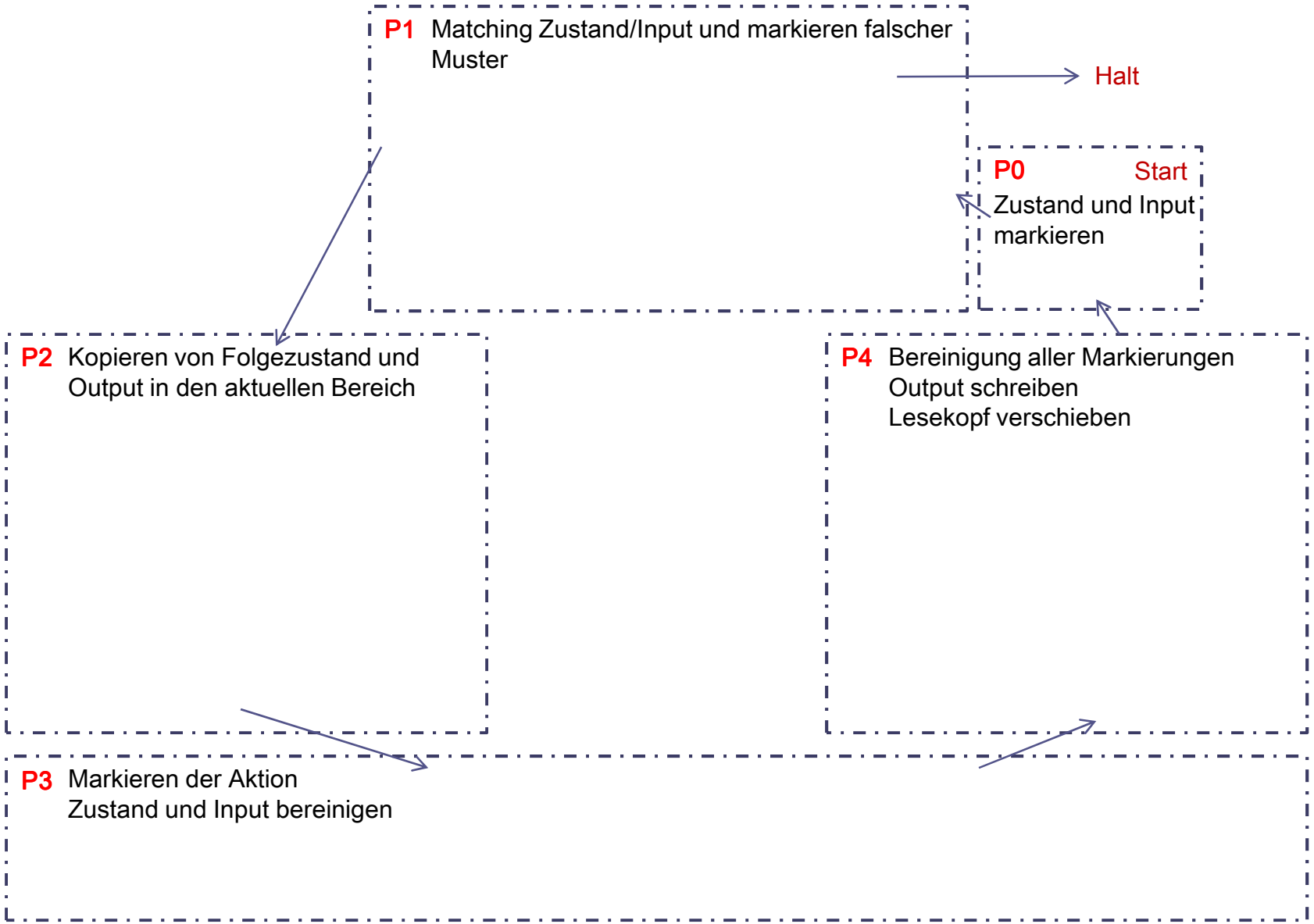
UTM-Architektur (nach Minsky)

[Minsky 1967]



Interpretation der UTM-Architektur

[Minsky 1967]

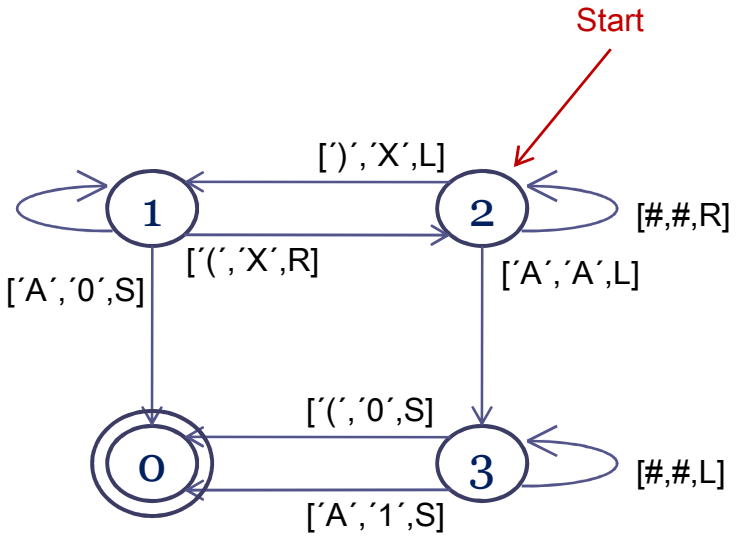


TM-Beispiel

[TM-Code]

Programm testet, ob die Klammern balanciert sind


```
klammer :: [(Int,Char,Int,Char,Char)]
klammer= [(1,'(',2,'X','r'),
          (1,'A',0,'0','s'),
          (1,'#',1,'#','l'),
          (2,')',1,'X','l'),
          (2,'A',3,'A','l'),
          (2,'#',2,'#','r'),
          (3,'(',0,'0','s'),
          (3,'A',0,'1','s'),
          (3,'#',3,'#','l')]
```



[Schöning 1999, Wegener 1993]

Haskell-Implementation I

[TM-Code]

Funktion step

Funktion führt in Abhängigkeit von aktuellem Zustand und der TM-Zustandstabelle zum Folgezustand

```
step::(Int,[Char],Char,[Char])->[(Int,Char,Int,Char,Char)]->(Int,[Char],Char,[Char])
step (q,l,s,r) tabula
    | q==0      = (q,l,s,r)                      -- Haltezustand, es passiert nichts
    | otherwise = (qn,new_l,new_s,new_r)         -- es gibt einen Übergang
    where
        (qn,o,dir) = find_next (q,s) tabula      -- (q,s) wird in der Tabelle gesucht
        (new_l,new_s,new_r)
            | dir=='l' = take_left (l,o,r)       -- gehe nach links
            | dir=='r' = take_right(l,o,r)       -- gehe nach rechts
            | dir=='s' = (l,o,r)                 -- bleibe stehen
```

Haskell-Implementation II

[TM-Code]

Funktionen take_left und take_right

Gehe nach links oder rechts und aktualisiere entsprechend das linke und rechte Band sowie den aktuellen Zustand.

```
take_left, take_right :: ([Char],Char,[Char]) -> ([Char],Char,[Char])
take_left (l,s,r)
                  | head_l == '_' = error "left margin reached"
                  | otherwise     = (tail l,head_l,s:r)
                    where head_l  = head l
take_right(l,s,r)
                  | head_r == '_' = error "right margin reached"
                  | otherwise     = (s:l,head_r,tail r)
                    where head_r  = head r
```

Haskell-Implementation III

[TM-Code]

Funktion find_next

In der Tabelle werden Zustand und Input gesucht und der entsprechende Folgezustand mit Output und Richtung zurückgeliefert.

```
find_next :: (Int, Char) -> [(Int, Char, Int, Char, Char)] -> (Int, Char, Char)
find_next (q,s) ((a,b,qn,out,dir):rest)
    | (a==q && b==s)     = (qn,out,dir)
    | (a==q && b=='#')   = (qn,s,dir)          -- Konvention: # bedeutet beliebiges Symbol
    | otherwise          = find_next (q,s) rest
```

Haskell-Implementation IV

[TM-Code]

Funktion sim

Die Simulation der TM wird mit dieser Funktion vorgenommen. Dabei werden die Zustände der Maschine protokolliert.

```
sim :: (Int, [Char], Char, [Char]) -> [(Int, Char, Int, Char, Char)]
                                   -> [(Int, [Char], Char, [Char])]
                                   -> [(Int, [Char], Char, [Char])]
sim (0,l,s,r) tabula protocol = protocol
sim (q,l,s,r) tabula protocol = sim new tabula (new:protocol)
                              where new = step (q,l,s,r) tabula
```

Haskell-Implementation V

[TM-Code]

Funktion start

Erzeugt ein unendliches (leeres) Band.

```
empty_tape :: [Char]
empty_tape = '_':empty_tape
```

Startet die Simulation mit (q,l,s,r) und einer Tabelle, wobei q der Zustand, l das linke Bank, s das Symbol und r das rechte Band sind.

```
start :: (Int, [Char], Char, [Char]) -> [(Int, Char, Int, Char, Char)] -> IO ()
start (q,l,s,r) tabula = putStr(show_states(sim (q1,l1,s1,r1) tabula [(q1,l1,s1,r1)]))
                         where (q1,l1,s1,r1) = (q,reverse(l)++empty_tape,s,r++empty_tape)
```

Haskell-Implementation VI

[TM-Code]

Funktion show_states (Variante 1)

Ausgabe zeilenweise

Zeilenweises Ausgeben der Zustände und Lesekopfposition.

```
show_states :: [(Int, [Char], Char, [Char])] -> String
show_states []       = ""
show_states (x:xs)   = (show_states xs)++"  "++show_state(x)

show_state :: (Int, [Char], Char, [Char]) -> String
show_state (q,l,s,r) = show(q)++"  "++rev_l++[s]++cut_tape r++"\n“
                       ++ spaces (5+length(rev_l))++"T\n"
                     where rev_l = reverse_finite l []
```

Erstellt einen String bestehend aus n Leerzeichen.

```
spaces :: Int -> [Char]
spaces n = [' '|_<-[1..n]]
```

Reduziert das Band auf die Symbole.

```
cut_tape :: String -> String
cut_tape ('_':_)          = []
cut_tape (x:y)            = x:(cut_tape y)
```

Dreht die Liste um.

```
reverse_finite :: String -> String -> String
reverse_finite ('_':_) y = y
reverse_finite (x:xs)  y = reverse_finite xs (x:y)
```

Haskell-Implementation VII

[TM-Code]

Funktion show_states (Variante 2)

Ausgabe in einer Zeile

Überschreiben des aktuellen Zustands.

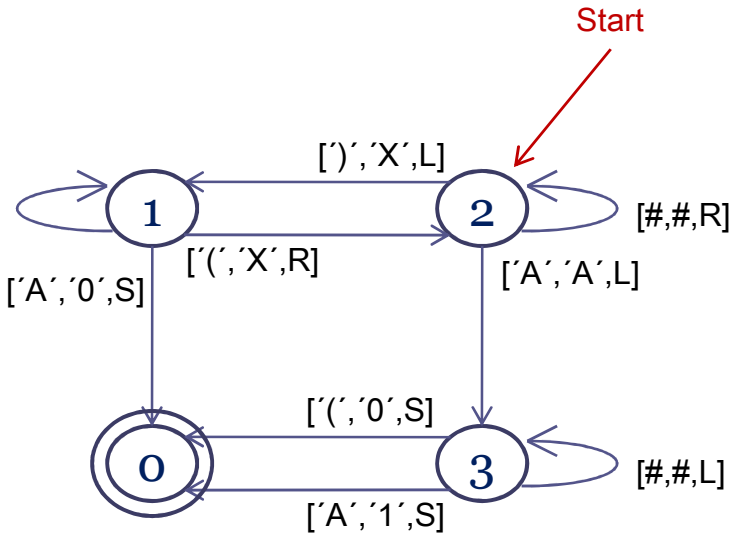
```
show_states [] = ""
show_states (x:xs)    = (show_states xs)++"  "++show_state(x)
show_state (q,l,s,r) = qstr++"  "++rev_l++[s]++cut_t++
                       backspaces(to_right)++"T"++backspaces(100)
                          where
                               qstr | (q<10)    = show(q)++" "
                                    | otherwise = show(q)
                               rev_l   = reverse_finite l []
                               cut_t   = cut_tape r
                               to_left = length(qstr) + length(rev_l)+5
                               to_right= length(cut_t)+1


backspaces n = ['\8'|_<-[1..n]]
```

TM-Beispiel I

[TM-Code]

Programm testet, ob die Klammern balanciert sind


```
klammer :: [(Int,Char,Int,Char,Char)]
klammer= [(1,'(',2,'X','r'),
          (1,'A',0,'0','s'),
          (1,'#',1,'#','l'),
          (2,')',1,'X','l'),
          (2,'A',3,'A','l'),
          (2,'#',2,'#','r'),
          (3,'(',0,'0','s'),
          (3,'A',0,'1','s'),
          (3,'#',3,'#','l')]
```



[Schöning 1999, Wegener 1993]

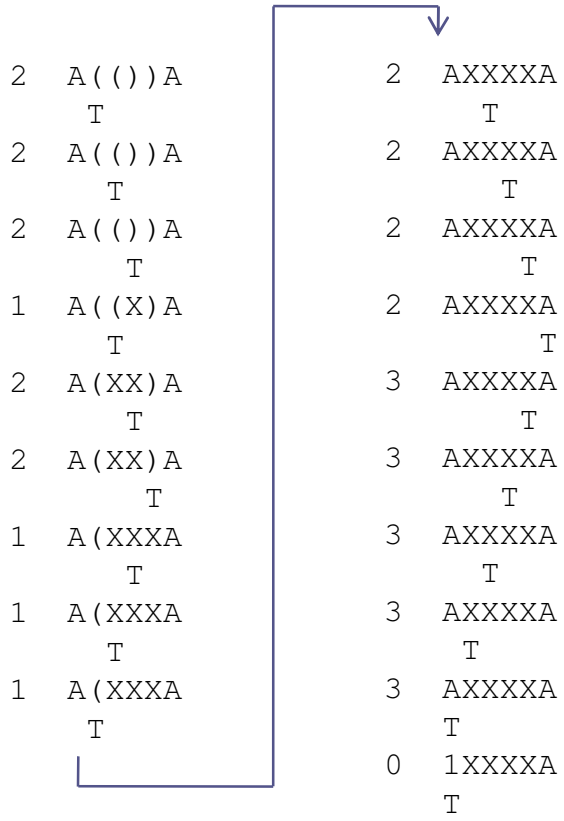
TM-Beispiel II

[TM-Code]

Programm testet, ob die Klammern balanciert sind

Aufruf in Haskell: `start(2,"A",'(',"())A") klammer`

Ausgabe:



Literaturquellen

[Minsky 1967] Minsky M.L.: „*Computation: Finite and Infinite Machines*“, Prentice Hall-Verlag, 1967

[TM-Code] Implementation der Turingmaschine in Haskell, zu finden unter:
http://page.mi.fu-berlin.de/block/haskell/turingInterpreter.hs

[Schöning 1999] Schöning U.: „*Theoretische Informatik*“, 3.Auflage, Spektrum-Verlag 1999

[Wegener 1993] Wegener I.: „*Theoretische Informatik - eine algorithmenorientierte Einführung*“, Teubner-Verlag 1993

Übung: Implementieren Sie die Universelle Turing-Maschine als Programm (Zustandsübergangstabelle) für die vorgestellte TM.