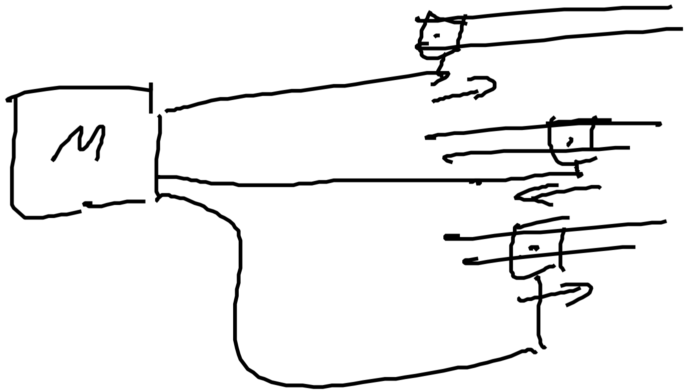
Mehrband - TMs

Eine Mehrband TM hat $k \geq 1$ Bänder mit k unabhängigen Schreib/Lese-Köpfen, d.h.

$$\delta : Z \times \Gamma^k \to Z \times \Gamma^k \times \{L, R, N\}^k$$



Ein- und Ausgabe erfolgt über Band 1

Satz: Zu jeder Mehrband-TM M gibt es eine Einband-TM M', so dass M' die selbe Funktion wie M berechnet bzw. die selbe Sprache wie M akzeptiert.

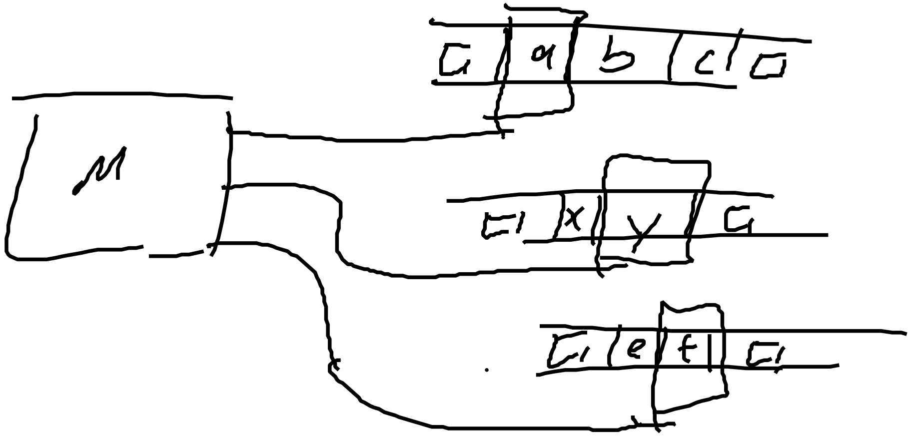
Bem.

2 neue Bandsymbole: * und | mit *, | $\notin \Gamma$

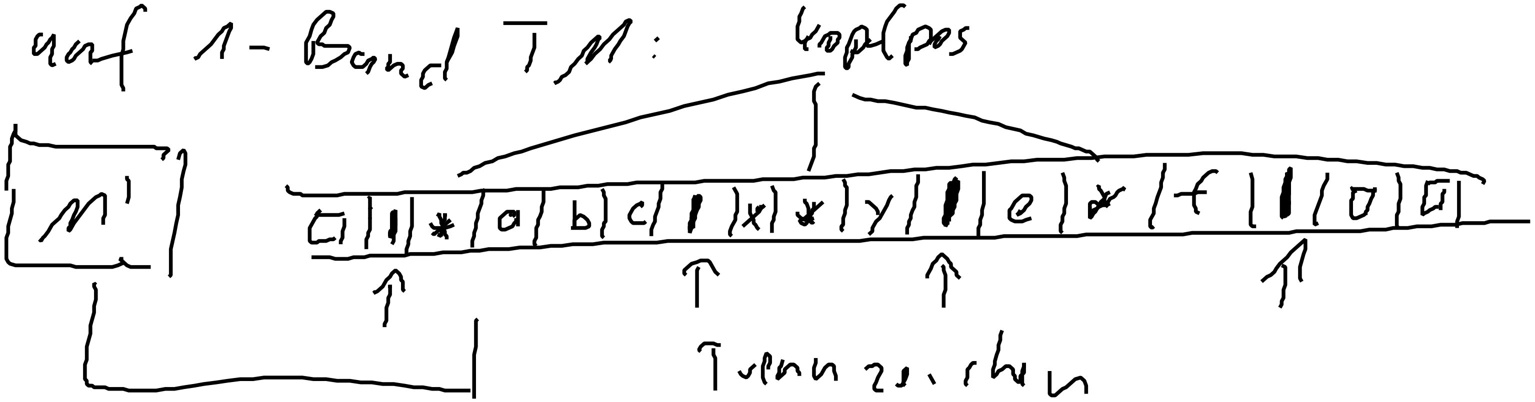
*: Markierung für Kopfposition

|: Trennzeichen zwischen "Bändern"

Darstellung des Inhalts von mehreren Bändern:



auf 1-Band TM:

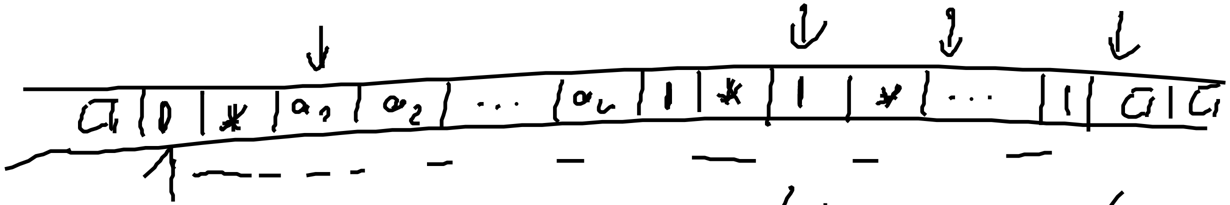


M' simuliert die Ausführung von M

1) Eingabe wird in Anfangskonf. überführt

Bei M steht auf Band 1 Eingabewort: $a_1 a_2 \ldots a_n$

Für M' dann:



2) M' fährt von links nach rechts über das Band.

Daraus ergibt sich, welchen Übergang M machen müsste.

M' fährt von rechts nach links und ändert Kopfpositionen und Zeichen entsprechend. Dabei kann es notwendig werden, dass M' Teile nach links verschieben muss.

3) Endkonfiguration erreicht?

Nein $\rightarrow$ 2

ja $\rightarrow$ 4

4) Endkonfiguration in die Ausgabe überführen.
Alle Bandinhalte der Bänder >2 löschen,
Kopfpositionszeilen löschen und nach links fahren. □

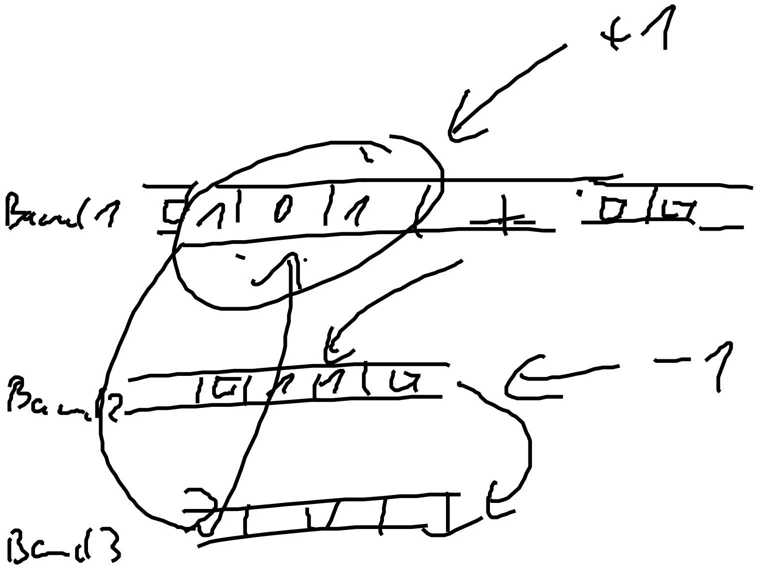
<u>Notation:</u> Falls M ein 1-Band-TM ist, dann sei
$M(i)$ eine Mehrband-TM, die M auf dem
i-ten Band ausführt (wobei $i \leq k$, f.s. ein k-Band TM)

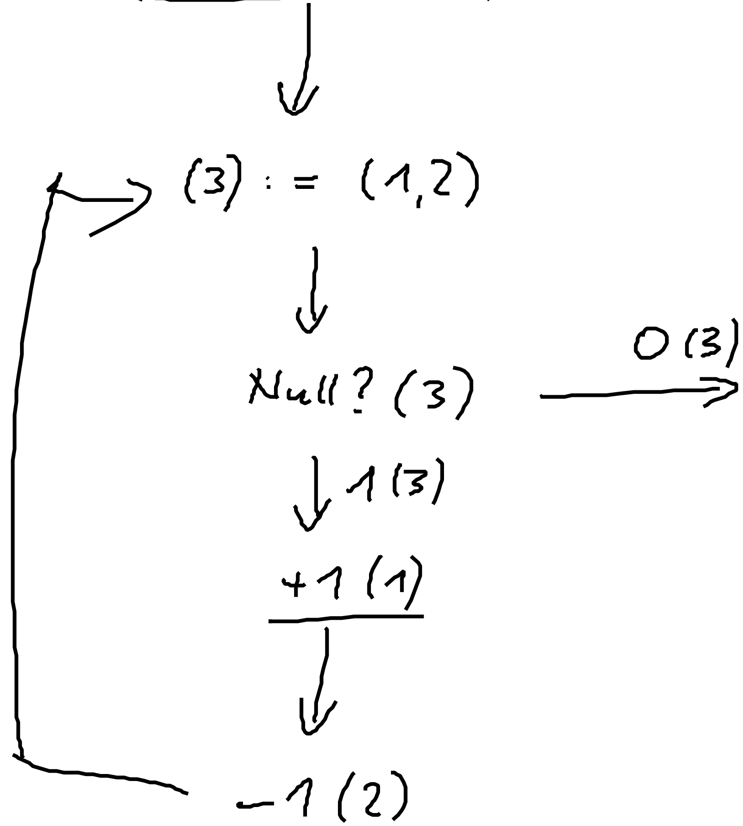
<u>Bsp:</u> <u>+1 (i)</u> : Addiert 1 auf den Inhalt des i-ten Bandes
<u>−1 (i)</u> : Subtrahiert 1 vom Inhalt des i-ten Bandes,
falls > 0.

<u>(i) := (j, k)</u> kopiert den j-ten Teil vom k-ten Band
auf das Band i.

<u>Null? (i)</u> Schreibt 0 auf das i-te Band, falls
das i-te Band die Null enthält oder
leer ist. Sonst 1.

$(3) := (1,1) \rightarrow (2) := (2,1) \rightarrow (1) := (1,3)$





x_1, x_2 Eingaben
x_0 Ausgabe

WHILE $x_2 \neq 0$ DO

$x_2 := x_2 - 1$

$x_1 := x_1 + 1$

END

$x_0 := x_1$

$x_0 := x_1 + x_2$

2.3 LOOP-, WHILE-, GOTO- Berechenbarkeit

2.3.1 LOOP-Berechenbarkeit

Bausteine

Var: $x_0, x_1, x_2, \ldots$

Konst: $0, 1, 2, \ldots$

Trennsymbole: $;$ $:=$

Schlüsselwort: LOOP DO END

Operatoren: $+$, $-$

Loop-Programme

- Wertzuweisung (x_i, x_j Var., c Konstante)

  $x_i := x_j + c$ $x_i := x_j - c$

  wie üblich, wobei für $x_j - c$ die modifizierte Subtraktion gemacht werden soll $= \max(0, x_j - c)$

- Konkatenation (P_1, P_2 sind LOOP-Prog.)

  $P_1 ; P_2$

  offensichtlich

- Schleife (x_i Var, P Prog.)

  LOOP x_i DO P END

  Die Schleife wird so oft durchlaufen, wie der Wert von x_i vor dem ersten Durchlauf war

Resultat eines Programms:

Programme werden mit einer Eingabe $n_1, \dots, n_k$ gestartet. Die Werte stehen dann in den Var. $\underline{x_1, \dots, x_k}$. Alle anderen Variablen haben den Wert 0.
Nach Ausführung des Programms ist der Wert in x_0 das <u>Resultat</u>.

<u>Def</u> Eine Fkt $f: \mathbb{N}^k \to \mathbb{N}$ heißt <u>LOOP-berechenbar</u> (Lb), falls es ein LOOP-Programm gibt, das mit $n_1, \dots, n_k$ in $x_1, \dots, x_k$ gestartet den Wert $f(n_1, \dots, n_k)$ in x_0 zurück gibt.

<u>Bem</u>: Alle Lb Fkt. sind total.

<u>Bsp.</u>:

```
x0 := x1;
LOOP x2 DO
  x0 := x0 + 1
END
```

Summation

```
LOOP x1 DO
  LOOP x2 DO
    x0 := x0 + 1
  END
END
```

$x_1 \cdot x_2$

$y := 1;$
LOOP x DO $y := 0$ END;
LOOP y DO $\underline{A}$ END

bedeutet:

IF $x=0$ THEN A END

D.h. wir können Konstrukte wie $*, +, /,$ MOD, IF $x=0$ THEN .. END, realisieren.

<u>Notation</u>: Alle arith. Operationen über zwei Var und bedingte Ausführung sollen zugelassen sein.

2.3.2 WHILE-Berechenbarkeit.

Ein weiteres Konstrukt:

WHILE $x_i \neq 0$ DO P END   Führt P so lange aus bis $x_i = 0$ wird.

<u>Def.</u> <u>WHILE-Berechenbarkeit</u> (Wb) analog zur LOOP-Berechenbarkeit.

Bem $wb \not\Rightarrow Lb$

z. B. Ω ist wb aber nicht Lb

Satz Alle wb Fkt. sind Tb.

Bew: Offensichtlich mit Hilfe von Hintereinanderschaltung von Mehrband-TMs.

2.3.3 GOTO-Berechenbarkeit

Anweisung

- Wertzuweisungen $x_i := x_j \pm c$
- unbedingte Sprünge GOTO M_i
- bedingte Sprünge IF $x_i = c$ THEN GOTO M_i
- Programmende: HALT

GOTO-Programme

$M_1 : A_1;$ M_i: Marken
$M_2 : A_2;$ A_i Anweisungen.
$\vdots$
$M_k : A_k$

GOTO-Berechenbarkeit (Gb) wird def. analog zur Lb und Wb.

Satz Alle Wb Fkt. sind Gb.

Bew: WHILE $x_i \neq 0$ DO P END

wird übersetzt zu

M_1: IF $x_i = 0$ THEN GOTO M_2;

P;

GOTO M_1;

M_2: ....

$\sqsubset$

Satz Alle Gb Fkt. sind Wb.

Bew: Gegeben GOTO-Prog.

$M_1 : A_1 ; M_2 : A_2 ; \dots , M_k : A_k$

wird übersetzt in WHILE-Prog

pc := 1;

WHILE pc ≠ 0 DO

IF pc = 1 THEN A_1^* END;

IF pc = 2 THEN A_2^* END;

⋮

IF pc = k THEN A_k^* END

END

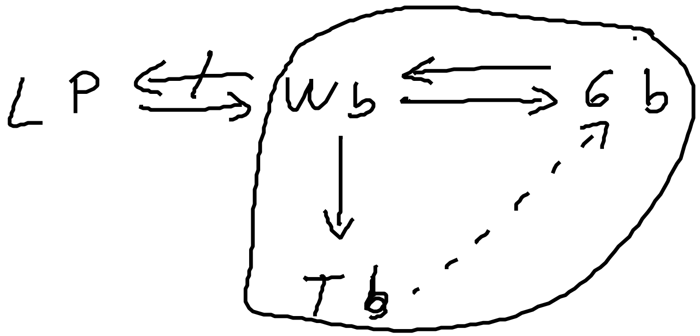
wobei $A_i^* = \begin{cases} x_j := x_k \pm c;\ pc := pc+1; & \text{falls } A_i = \text{"}x_j := x_k \pm c\text{"} \\ pc := n & \text{falls } A_i = \text{"Goto } M_n\text{"} \\ pc := 0 & \text{falls } A_i = \text{"HALT"} \\ y := PC;\ PC := n; & \text{falls } A_i = \text{"IF } x_j = c \\ \text{IF } x_j \neq c \text{ THEN } PC := y+1 \text{ END} & \text{THEN GOTO } M_n\text{"} \end{cases}$

<u>Satz</u> (Kleenesche Normalform für WHILE-Prog)
Jede wb Fkt kann durch ein WHILE-Prog mit nur einer WHILE-Schleife berechnet werden.

<u>Bew</u>: Gegeben ein WHILE-Prog, erzeuge äquivalentes GOTO-Prog. und übersetze das in ein WHILE-Prog. wie im obigen Beweis. □

α

γ



<u>Satz</u> Jede Tb Fkt. ist Gb.