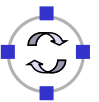


Programmierung und Algorithmen

Kapitel 5 Formale Algorithmenmodelle

P&A (WS 25/26): 05 – Formale Algorithmenmodelle

1



Überblick

Motivation

Formale Algorithmenmodelle

Registermaschine

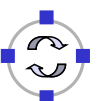
Abstrakte Maschinen

Markov-Algorithmen

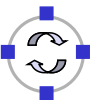
Church'sche These

P&A (WS 25/26): 05 – Formale Algorithmenmodelle

2



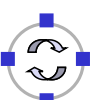
- ❑ Bisher: Ausführung von Algorithmen auf abstrakter Ebene
- ❑ Nun: Entwicklung von **Modellen für Maschinen**, die Algorithmen ausführen
- ❑ Ziel: einfache Modelle
 - ❑ Näher an Computer als abstrakte Ausführung applikativer Algorithmen
 - ❑ Einfacher für mathematische Betrachtungen als bisherige Modelle
- ❑ Konkret: Registermaschine als einfaches Modell realer Computer
- ❑ Abstrakte Maschinen als vereinheitlichender Rahmen
- ❑ Markov-Algorithmen als einfach zu programmierende Maschinen auf Zeichenketten



- ❑ (Maschinennahe) Präzisierung des Algorithmenbegriffs:
Registermaschine
 - ❑ Registermaschine besteht aus Registern $B, C_0, C_1, C_2, \dots, C_n, \dots$ und einem Programm.
 - ❑ B heißt Befehlszähler, C_0 heißt Arbeitsregister oder Akkumulator, $C_n, n \geq 1$ heißen Speicherregister
 - ❑ Jedes der Register enthält als Wert eine natürliche Zahl
 - ❑ **Konfiguration** der Registermaschine:

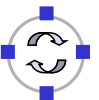
$$(b, c_0, c_1, \dots, c_n, \dots)$$

mit Register B enthält Zahl b und
für $n \geq 0$ gilt, das Register C_n enthält Zahl c_n



- Programm einer Registermaschine
 - Programm = endliche Folge von Befehlen
 - Durch Anwendung eines Befehls wird die Konfiguration der Registermaschine geändert
 - Notation

$$(b, c_0, c_1, \dots, c_n, \dots) \vdash (b', c'_0, c'_1, \dots, c'_n, \dots)$$



LOAD i , $i \in \mathbb{N}_+$ $b' = b + 1$ $c'_0 = c_i$

CLOAD i , $i \in \mathbb{N}$ $b' = b + 1$ $c'_0 = i$

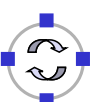
STORE i , $i \in \mathbb{N}_+$ $b' = b + 1$ $c'_i = c_0$

(bei diesen Operationen gilt grundsätzlich zusätzlich zu den spezifischen Erklärungen der folgenden Folien:

$c'_j = c_j$ für $j \neq 0$; bzw.

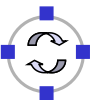
$c'_j = c_j$ für $j \neq 0 \wedge j \neq i$ bei STORE i

d.h.: Befehle haben keine „unspezifische Nebenwirkungen“ auf die aktuelle Konfiguration)



ADD i ,	$i \in \mathbb{N}_+$	$b' = b + 1$	$c'_0 = c_0 + c_i$
CADD i ,	$i \in \mathbb{N}_+$	$b' = b + 1$	$c'_0 = c_0 + i$
SUB i ,	$i \in \mathbb{N}_+$	$b' = b + 1$	$c'_0 = \begin{cases} c_0 - c_i & \text{für } c_0 \geq c_i \\ 0 & \text{sonst} \end{cases}$
CSUB i ,	$i \in \mathbb{N}_+$	$b' = b + 1$	$c'_0 = \begin{cases} c_0 - i & \text{für } c_0 \geq i \\ 0 & \text{sonst} \end{cases}$
MULT i ,	$i \in \mathbb{N}_+$	$b' = b + 1$	$c'_0 = c_0 \cdot c_i$
CMULT i ,	$i \in \mathbb{N}_+$	$b' = b + 1$	$c'_0 = c_0 \cdot i$
DIV i ,	$i \in \mathbb{N}_+$	$b' = b + 1$	$c'_0 = \lfloor c_0 / c_i \rfloor$
CDIV i ,	$i \in \mathbb{N}_+$	$b' = b + 1$	$c'_0 = \lfloor c_0 / i \rfloor$

(bei diesen Operationen gilt grundsätzlich $c'_j = c_j$ für $j \neq 0$)



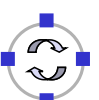
□ Sprungbefehle:

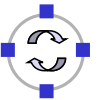
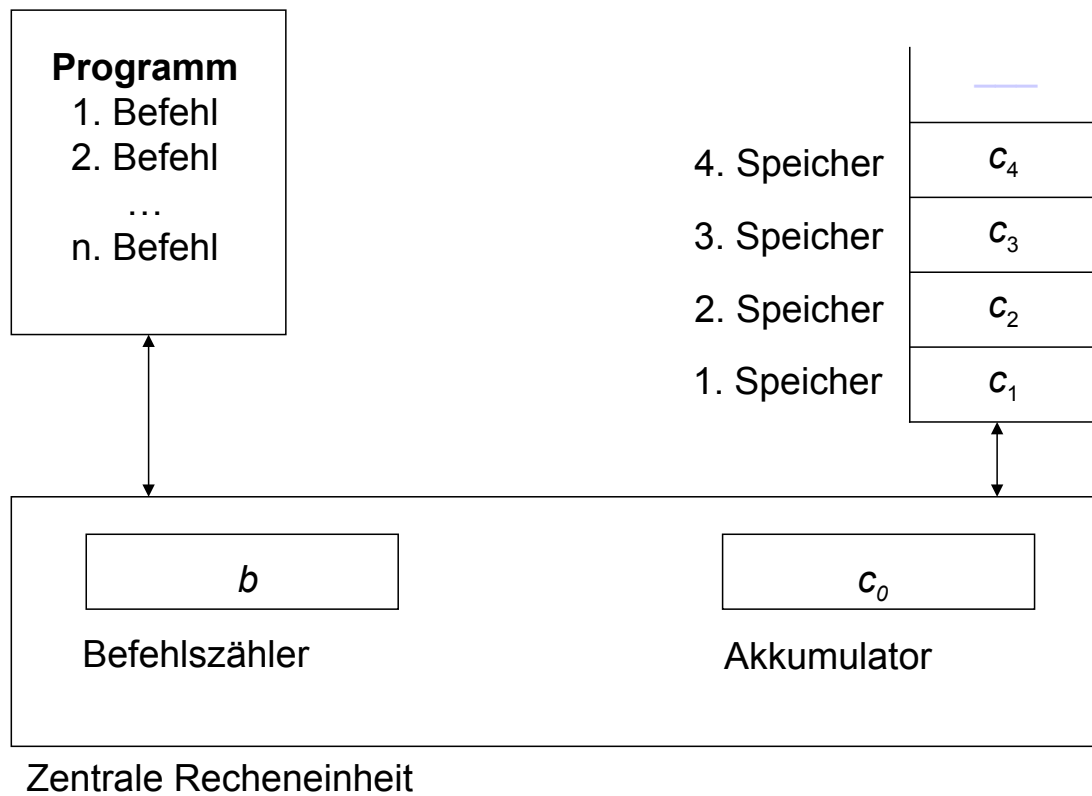
GOTO i , $i \in \mathbb{N}_+$ $b' = i$

IF $c_0 = 0$ GOTO i , $i \in \mathbb{N}_+$ $b' = \begin{cases} i & \text{falls } c_0 = 0 \\ b + 1 & \text{sonst} \end{cases}$

□ Stoppbefehl:

END, $b' = b$ $c'_j = c_j$ für $j \geq 0$





```
1  LOAD 1
2  DIV 2
3  MULT 2
4  STORE 3
5  LOAD 1
6  SUB 3
7  STORE 3
8  END
```

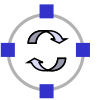


□ Startkonfiguration

$$b = 1, c_0 = 0, c_1 = 32, c_2 = 5, c_3 = 0,$$

□ Folge von Konfigurationen:

$$\begin{aligned} (1, 0, 32, 5, 0, \dots) &\vdash (2, 32, 32, 5, 0, \dots) \vdash (3, 6, 32, 5, 0, \dots) \\ &\vdash (4, 30, 32, 5, 0, \dots) \vdash (5, 30, 32, 5, 30, \dots) \\ &\vdash (6, 32, 32, 5, 30, \dots) \vdash (7, 2, 32, 5, 30, \dots) \\ &\vdash (8, 2, 32, 5, 2, \dots) \end{aligned}$$

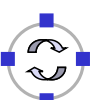


□ Startkonfiguration

$$b = 1, c_0 = 0, c_1 = 100, c_2 = 20, c_3 = 0,$$

□ Folge von Konfigurationen:

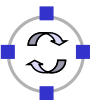
$$\begin{aligned} (1, 0, 100, 20, 0, \dots) &\vdash (2, 100, 100, 20, 0, \dots) \\ &\vdash (3, 5, 100, 20, 0, \dots) \\ &\vdash (4, 100, 100, 20, 0, \dots) \\ &\vdash (5, 100, 100, 20, 100, \dots) \\ &\vdash (6, 100, 100, 20, 100, \dots) \\ &\vdash (7, 0, 100, 20, 100, \dots) \\ &\vdash (8, 0, 100, 20, 0, \dots). \end{aligned}$$



$$b = 1, c_0 = 0, c_1 = n, c_2 = m, c_3 = 0$$

- $n = q \cdot m + r$ mit $0 \leq r < m$
- d. h. $q = \lfloor n/m \rfloor$ ist ganzzahliges Ergebnis der Division von n durch m
- r ist der verbleibende Rest bei dieser Division

$$\begin{aligned}
 (1, 0, n, m, 0, \dots) &\vdash (2, n, n, m, 0, \dots) && \vdash (3, q, n, m, 0, \dots) \\
 &\vdash (4, q \cdot m, n, m, 0, \dots) && \vdash (5, q \cdot m, n, m, q \cdot m, \dots) \\
 &\vdash (6, n, n, m, q \cdot m, \dots) && \vdash (7, r, n, m, q \cdot m, \dots) \\
 &\vdash (8, r, n, m, r, \dots)
 \end{aligned}$$



Eine Registermaschine M **berechnet** die Funktion

$$f: \mathbb{N}^n \longrightarrow \mathbb{N}^m$$

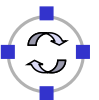
mit $f(x_1, x_2, \dots, x_n) = (y_1, y_2, \dots, y_m)$ wenn es Zahlen $i_1, i_2, \dots, i_m$ so gibt, dass M jede Konfiguration

$$(1, 0, x_1, x_2, \dots, x_n, 0, 0, \dots)$$

in eine Konfiguration $(b, c_0, c_1, \dots)$ überführt, für die b die Nummer einer **END**-Anweisung ist und $c_{i_j} = y_j$ für $1 \leq j \leq m$ gilt.



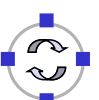
- ❑ Die Registermaschine beginnt die Abarbeitung mit dem ersten Befehl
- ❑ Die Argumente bzw. Eingaben der zu berechnenden Funktion stehen dabei in den ersten n Speicherregistern $C_1, \dots, C_n$
- ❑ Die Registermaschine beendet ihre Arbeit bei Erreichen eines **END**-Befehls
- ❑ Die Resultate bzw. Ausgaben stehen nach beendeter Abarbeitung in den vorab festgelegten Speicherregistern $C_{i_1}, C_{i_2}, \dots, C_{i_m}$



Berechnete Funktion: Beispiel

```
1  LOAD  1
2  ADD   2
3  STORE 3
4  END
```

- ❑ Berechnet aus Werten x und y in Registern C_1 und C_2 die Summe $x + y$ und legt diese im Register C_3 ab
- ❑ Mit $i_1 = 3$ wird die Addition realisiert

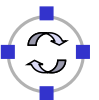


```

1  CLOAD 1
2  STORE 3
3  LOAD 2
4  IF  $c_0 = 0$  GOTO 12
5  LOAD 3
6  MULT 1
7  STORE 3
8  LOAD 2
9  CSUB 1
10 STORE 2
11 GOTO 4
12 END

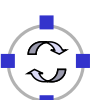
```

□ $f_2 : \mathbb{N}^2 \longrightarrow \mathbb{N}$ mit Ergebnis im dritten Speicherregister c_3 (also $i_1 = 3$)



$(1, 0, 5, 3, 0, \dots)$
 $\vdash (2, 1, 5, 3, 0, \dots) \quad \vdash (3, 1, 5, 3, 1, \dots) \quad \vdash (4, 3, 5, 3, 1, \dots)$
 $\vdash (5, 3, 5, 3, 1, \dots) \quad \vdash (6, 1, 5, 3, 1, \dots) \quad \vdash (7, 5, 5, 3, 1, \dots)$
 $\vdash (8, 5, 5, 3, 5, \dots) \quad \vdash (9, 3, 5, 3, 5, \dots) \quad \vdash (10, 2, 5, 3, 5, \dots)$
 $\vdash (11, 2, 5, 2, 5, \dots) \quad \vdash (4, 2, 5, 2, 5, \dots) \quad \vdash (5, 2, 5, 2, 5, \dots)$
 $\vdash (6, 5, 5, 2, 5, \dots) \quad \vdash (7, 25, 5, 2, 5, \dots) \quad \vdash (8, 25, 5, 2, 25, \dots)$
 $\vdash (9, 2, 5, 2, 25, \dots) \quad \vdash (10, 1, 5, 2, 25, \dots) \quad \vdash (11, 1, 5, 1, 25, \dots)$
 $\vdash (4, 1, 5, 1, 25, \dots) \quad \vdash (5, 1, 5, 1, 25, \dots) \quad \vdash (6, 25, 5, 1, 25, \dots)$
 $\vdash (7, 125, 5, 1, 25, \dots) \quad \vdash (8, 125, 5, 1, 125, \dots)$
 $\vdash (9, 1, 5, 1, 125, \dots) \quad \vdash (10, 0, 5, 1, 125, \dots)$
 $\vdash (11, 0, 5, 0, 125, \dots) \quad \vdash (4, 0, 5, 0, 125, \dots)$
 $\vdash (12, 0, 5, 0, 125, \dots)$

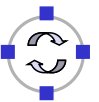
Dieses konkrete Beispiel ergibt $f_1(5, 3) = 125$.



Konfiguration $(1, 0, x, y, 0, \dots)$:

- Nach Abarbeitung der Befehle 1 – 3 ergibt sich $(4, y, x, y, 1, \dots)$.
 - $y = 0$: END-Anweisung; $(12, y, x, y, 1) = (12, 0, x, 0, 1)$ mit Ergebnis 1.
 - Falls $y \neq 0$: Befehle 5 – 11; $(4, y - 1, x, y, 1 \cdot x, \dots)$.
 - $y - 1 = 0$: $(12, y - 1, x, y - 1, x, \dots) = (12, 0, x, 0, x, \dots)$ mit Ergebnis x .
 - $y - 1 \neq 0$: Befehle 5 – 11; $(4, y - 2, x, y - 2, x^2, \dots)$.
 - $y - 2 = 0$: $(12, y - 2, x, y - 2, x^2, \dots) = (12, 0, x, 0, x^2, \dots)$ mit Ergebnis x^2 .
 - $y - 2 \neq 0$: Befehle 5 – 11 etc.
- Abbruch allgemein $(12, y - k, x, y - k, x^k, \dots) = (12, 0, x, 0, x^y, \dots)$ (wegen $y = k$).

Berechnete Funktion: $f_2(x, y) = x^y$



□ Programm

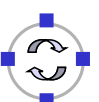
```

1  LOAD 1
2  IF  $c_0 = 0$  GOTO 4
3  GOTO 3
4  END

```

□ Berechnet die Funktion:

$$f_3(x) = \begin{cases} 0 & \text{falls } x = 0 \\ \text{undefiniert} & \text{sonst} \end{cases}$$



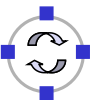
Beispiel für komplexeres Programm:

$$f_4(x) = \begin{cases} 0 & \text{falls } x \text{ keine Primzahl ist} \\ 1 & \text{falls } x \text{ eine Primzahl ist} \end{cases}$$

Wir überprüfen zuerst, ob die Eingabe $x \leq 1$ ist. Sollte dies der Fall sein, so ist x keine Primzahl und wir schreiben in das zweite Speicherregister eine 0 und beenden die Arbeit.

Ist dagegen $x \geq 2$, so testen wir der Reihe nach, ob x durch 2, 3, 4, . . . $x - 1$ teilbar ist. Gibt einer dieser Tests (für $x > 2$) ein positives Resultat, so ist x keine Primzahl.

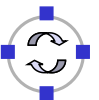
Fallen dagegen alle diese Tests negativ aus (oder ist $x = 2$), so ist x prim.



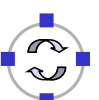
1	LOAD 1	Laden von x
2	CSUB 1	Berechnen von $x - 1$
3	IF $c_0 = 0$ GOTO 19	Test, ob $x \leq 1$
4	CLOAD 2	Laden des ersten Testteilers $t=2$
5	STORE 2	Speichern des Testteilers t
6	LOAD 1	
7	SUB 2	Berechnen von $x - t$
8	IF $c_0 = 0$ GOTO 21	Test, ob $t \leq x$
9	LOAD 1	
10	DIV 2	Befehle 9 – 14 berechnen Rest
11	MULT 2	bei ganzzahliger Teilung x / t
12	STORE 3	(siehe Beispiel M_1)
13	LOAD 1	
14	SUB 3	



15	IF $c_0 = 0$ GOTO 19	Test, ob t Teiler
16	LOAD 2	
17	CADD 1	Erhöhung des Testteilers um 1
18	GOTO 5	Start des nächsten Tests
19	STORE 2	Speichern des Ergebnisses 0
20	GOTO 23	
21	CLOAD 1	
22	STORE 2	Speichern des Ergebnisses 1
23	END	



- Gemeinsamkeit bisheriger Modelle (Registermaschine, applikative und imperative Algorithmen)
 - Kontrollstrukturen + elementare, von einer „Maschine“ ausführbare Einzelschritte
- Allgemeines Modell für deterministische Algorithmen:
abstrakte Maschine
 - Ermöglicht den Vergleich wichtiger Eigenschaften wie *Laufzeit* und *Terminierung* unabhängig von einem konkreten Algorithmenmodell bzw. Programmierparadigma
- Weitere Spezialisierungen abstrakter Maschinen:
Turing-Maschine, Markov-Algorithmen



$$M = (X, Y, K, \alpha, \omega, T, \sigma),$$

X Menge von *Eingabewerten*

Y Menge von *Ausgabewerten*

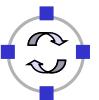
K Menge von *Konfigurationen*

$\alpha: X \rightarrow K$ *Eingabefunktion*

$\omega: K \rightarrow Y$ *Ausgabefunktion*

$T: K \rightarrow K$ *Transitionsfunktion*

$\sigma: K \rightarrow \text{bool}$ *Stoppfunktion* (markiert Endkonfiguration)

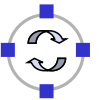
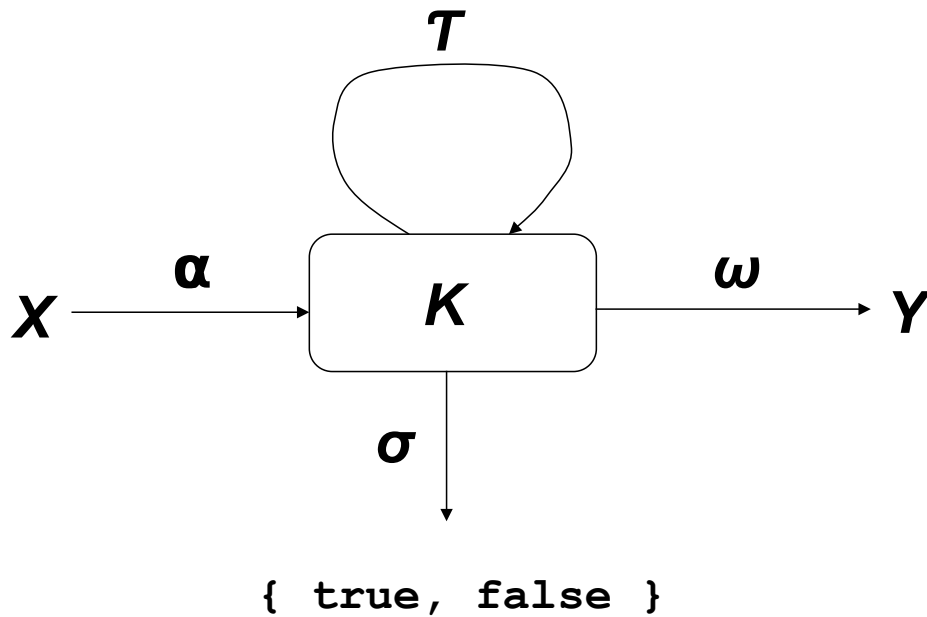


Endkonfigurationen zu M :

$$E = \{ k \in K \mid \sigma(k) = \text{true} \}$$

Endkonfigurationen sind Maschinenzustände, in denen eine Berechnung terminiert.





Eine abstrakte Maschine *arbeitet* folgendermaßen:

1. Ein Eingabewert $x \in X$ bestimmt die Anfangskonfiguration
 $k_0 = \alpha(x) \in K$.
2. Wir überführen mittels T Konfigurationen in Folgekonfigurationen, also

$$k_1 = T(k_0), k_2 = T(k_1), \dots$$

bis zum ersten Mal eine Endkonfiguration $k_i \in E$ erreicht wird. Dies braucht natürlich niemals eintreten.

3. Wird eine Endkonfiguration $k_i \in E$ erreicht, so wird der Ausgabewert $\omega(k_i) \in Y$ ausgegeben.

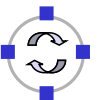


Bei Eingabe von $x \in X$ gibt es also zwei Möglichkeiten:

1. Die Maschine hält nie an.
2. Die Maschine hält und gibt einen eindeutig bestimmten Ausgabewert $y \in Y$ aus.

Auf diese Weise berechnet die Maschine M eine partielle Funktion

$$f_M : X \rightarrow Y$$



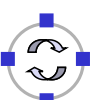
Die **Laufzeit** einer abstrakten Maschine M für die Eingabe $x \in X$ ist

$$t_M(x) = (\mu n) [\sigma (T^n (\alpha(x)))]$$

Hierbei ist $(\mu n) [B]$ die kleinste natürliche Zahl

$n \in \mathbb{N} = \{0, 1, 2, \dots\}$, so dass die Bedingung $B = \text{true}$ wird und B für alle $m \leq n$ definiert ist.

Gibt es *keine* solche Zahl $n \in \mathbb{N}$, so ist $t_M(x) = \perp$ (undefiniert).

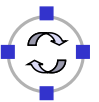


Die von einer abstrakten Maschine M **berechnete Funktion**

$$f_M : X \rightarrow Y$$

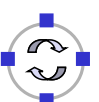
ist gegeben durch

$$f_M(x) = \omega (T^{t_M(x)} (\alpha(x))) ; \quad \text{ist } t_M(x) = \perp, \text{ so ist } f_M(x) = \perp .$$



- ☐ $X = Y = \mathbb{N}$
- ☐ $K = \mathbb{N} \times \mathbb{N}$
- ☐ $\alpha(n) = (1, n)$
- ☐ $T((m, n)) = (m * n, n - 1)$
- ☐ $\omega((m, n)) = m$
- ☐ $\sigma((m, n)) = \begin{cases} \text{true}, & \text{falls } n = 0 \\ \text{false} & \text{sonst} \end{cases}$

Laufzeit? Berechnete Funktion?



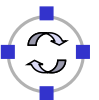
$$f_i(u_{i,1}, \dots, u_{i,n_i}) = t_i(u_{i,1}, \dots, u_{i,n_i}), i = 1, \dots, m.$$

- $X = \mathbb{Z}^{n_1}$
- $Y = \mathbb{Z}$
- K = Terme ohne Unbekannte
- $\alpha(a_1, \dots, a_{n_1})$ = der Term “ $f_1(a_1, \dots, a_{n_1})$ ”
- $\omega(t)$ = der Wert von t
- T : Termauswertung aufgrund der Funktionsdefinitionen.

Durch “Berechnungsvorschrift” deterministisch machen!

Z.B. durch die Forderung, stets das erste Auftreten von links eines Funktionsaufrufes mit Konstanten $f_i(b_1, \dots, b_{n_i})$ mit $b_j \in \mathbb{Z}, j=1, \dots, n_i$, durch die rechte Seite $t_i(b_1, \dots, b_{n_i})$ zu ersetzen.

- $\sigma(t) = \begin{cases} \text{true}, & \text{falls } t = b \in \mathbb{Z} \text{ (Konstante) ist} \\ \text{false} & \text{sonst} \end{cases}$



```

PROG:  var      V, W, ...: int;
        P, Q, ...: bool;
        input    X1, ... , Xn;
         $\bar{\beta}$ 
        output   Y1, . . . , Ym.

```



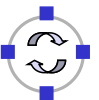
- $X = \mathbb{Z}^n, Y = \mathbb{Z}^m$
- $K = \{ (Z, \beta) \mid Z \text{ Zustand, } \beta \text{ Anweisung} \}$
 Z : aktueller Zustand, β : noch auszuführende Anweisung
- $\alpha(a_1, \dots, a_n) = (Z_0, \bar{\beta})$, wobei

$$Z_0(X_i) = a_i, i = 1, \dots, n, \text{ und}$$

$$Z_0(Y) = \perp \text{ für } Y \neq X_i, i = 1, \dots, n.$$
- $\omega(Z, \beta) = (Z(Y_1), \dots, Z(Y_m))$
- $T(Z, \beta) = (Z', \beta')$, wobei

$$Z' = \text{Zustand nach Ausführung der nächsten Anweisung}$$

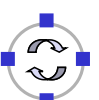
$$\beta' = \text{Rest der noch auszuführenden Anweisung}$$
- $\sigma(t) = \begin{cases} \text{true,} & \text{falls } \beta \text{ leer ist (keine Anweisungen mehr)} \\ \text{false} & \text{sonst} \end{cases}$



- Einfaches mathematisch orientiertes Modell als
Spezialisierung abstrakter Maschinen
- Programmtechnisch einfach in einen Interpretierer für
Markov-Tafeln umzusetzen

$A = (a_1, \dots, a_n)$ Alphabet

A^* die Menge der Worte (Texte) über A



Ein **Markov-Algorithmus** wird definiert durch **Regeln** der Form:

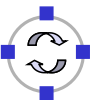
$$\varphi \rightarrow \psi \text{ mit } \varphi, \psi \in A^*$$

Bedeutung der Regel: Wort φ soll durch das Wort ψ ersetzt werden.

Angewendet auf ein Wort $\xi \in A^*$ entsteht durch Regelanwendung auf eindeutige Weise ein neues Wort $g[\varphi \rightarrow \psi](\xi)$:

- Ist φ ein Teilwort von ξ , also $\xi = \mu\varphi\nu$ für $\mu, \nu \in A^*$, und ist φ an dieser Stelle nach μ das erste Auftreten (von links) von φ in ξ , so ist $g[\varphi \rightarrow \psi](\xi) = \mu\psi\nu$, d. h. φ wird (nur!) an dieser Stelle durch ψ ersetzt.
- Ist φ kein Teilwort von ξ , so ist $g[\varphi \rightarrow \psi](\xi) = \xi$, d. h., es passiert nichts.

Achtung: $\varphi, \psi \in A^*$ bedeutet: φ oder ψ können auch ε (leeres Wort) sein!

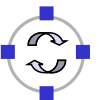


Sei $A = (0, 1)$. Wir wenden die Regel $01 \rightarrow 10$ sukzessive an.

1 1 0 <u>0</u> 1 0 1 0 1 1	→	1 1 <u>0</u> 1 0 0 1 0 1 1
	→	1 1 1 0 0 <u>0</u> 1 0 1 1
	→	1 1 1 0 <u>0</u> 1 0 0 1 1
	→	1 1 1 <u>0</u> 1 0 0 0 1 1
		etc.



- ❑ Originalvorgehensweise:
 1. Gehe alle Regeln *der Reihe* nach durch.
 2. Wenn eine Regel anwendbar ist, wende sie an der *linkesten möglichen* Stelle an.
 3. Starte von vorne.
- ❑ Terminierung:
 - ❑ Einige Regeln als Terminierungsregeln markieren, *oder*
 - ❑ Terminiere wenn keine Regel mehr anwendbar ist.
- ❑ Hier: an Rechnermodelle angepasste Variante, die **Markov-Tafeln**.

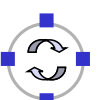


Eine **Markov-Tafel** ist wie folgt definiert:

- ❑ Sie besteht aus fünf Spalten und beliebig vielen (endlich vielen) Zeilen.
- ❑ Eine Zeile ist ein 5-Tupel der Form

$$k \quad \varphi \quad \psi \quad i \quad j$$

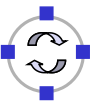
wobei $i, j, k \in \mathbb{N}$ sind, k ist die Zeilennummer, und $\varphi \psi$ stellen die Regel $\varphi \rightarrow \psi$ dar, i ist die Nummer der nächsten Zeile, falls das Teilwort gefunden (die Regel also angewandt) wurde, und j ist die Nummer der nächsten Zeile, falls φ nicht auftrat.



Die **Ausführung** der Markov-Tafel beginnt in der ersten Zeile (Nummer 0) und stoppt sobald zu einer nicht vorhandenen Zeilennummer (i oder j) gegangen werden soll.

Der **Zustand** bei der Ausführung einer Markov-Tafel ist eindeutig festgelegt durch

- die sich in Bearbeitung befindende Zeichenkette, und
- die aktuelle Zeilennummer.


 $X \subseteq A^*$

Eingabemenge

 $Y \subseteq A^*$

Ausgabemenge

 $K \subseteq \{ (z, n) \mid z \in A^*, n \in \mathbb{N} \}$

Konfigurationen

 $\alpha(x) = (x, 0)$

Eingabefunktion

 $\omega(z, n)$

Ausgabefunktion

 $T : K \rightarrow K$

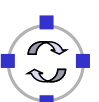
Transitionsfunktion

$T(z, n) = (g[\varphi \rightarrow \psi](z), n')$, wobei
 $\varphi \rightarrow \psi$ die Regel in der Zeile n ist und
 n' die Folgennummer (oben: i bzw. j)

 $\sigma(z, n)$

$$\sigma(z, n) = \begin{cases} \text{true,} & \text{falls } n \text{ keine Zeilennummer} \\ \text{false,} & \text{sonst} \end{cases}$$

Ein Markov-Algorithmus ist damit tatsächlich eine direkte Spezialisierung des Konzepts der abstrakten Maschine.



"Addiere |"

$$A = \{ | \}; \quad X = A^*; \quad Y = A^+ = A^* - \{ \epsilon \}$$

Markov-Tafel:

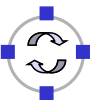
k	φ	ψ	i	j
0	ϵ		1	-

berechnet die Funktion $f(|^n) = |^{n+1}$, $n \in \mathbb{N}$

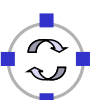
- Formuliert mit per „.“ markierter Terminierungsregel:

$$\epsilon \rightarrow |.$$

(Nach Ausführung einer mit . markierten Regel, terminiert der Algorithmus.)



- Das leere Wort ϵ kommt in jedem Wort vor. Das erste Auftreten ist ganz am Anfang.
- Der Algorithmus 'Addition von Eins' schreibt also ein | vor das Eingabewort $|^n = || \dots |$.
- Der j -Eintrag der Tabelle kann niemals angelaufen werden und ist daher irrelevant. Dies deuten wir durch das Zeichen „-“ in der Tabelle an.



$$A_0 = \{ | \}; A = A_0 \cup \{ + \}; X = A_0^* + A_0^* (= \{ \mu + \nu \mid \mu, \nu \in A_0^* \}); Y = A_0^*$$

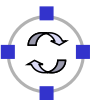
Markov-Tafel:

k	φ	ψ	i	j
0	+	ε	1	-

Der Algorithmus löscht das erste + im Eingabewort, also:

$$f(|^n + |^m) = |^{n+m}$$

- Formuliert mit per „.“ markierter Terminierungsregel:
 $+ \rightarrow \varepsilon$.



$$A_0 = \{ | \}; A = A_0 \cup \{ \# \}; X \stackrel{*}{=} Y = A_0$$

Das Zeichen # spielt die Rolle einer Markierung (gelegentlich auch als „Schiffchen“ bezeichnet), die einmal von links nach rechts durchwandert und dabei die Zeichen | verdoppelt.

k	φ	ψ	i	j	Kommentar (hierbei gilt stets: $p = n - q$)
0		#	1	3	$ ^n \rightarrow \# ^n$
1	#	#	1	2	$ ^{2p} \# ^q \rightarrow ^{2(p+1)} \# ^{q-1}$ wiederholen bis $q = 1$
2	#	ε	3	-	$ ^{2p} \# \rightarrow ^{2n}$ hier gilt $q = 0$

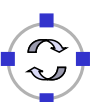
- Formuliert mit per „.“ markierter Terminierungsregel:

$$\# | \rightarrow || \#$$

$$\# \rightarrow \varepsilon$$

$$| \rightarrow \# |$$

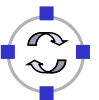
(Man beachte, dass die dritte Regel nur zu Beginn einmal ausgeführt werden kann. Ebenso kann die zweite Regel nur am Ende ausgeführt werden.)



Eine Berechnung mit dem Eingabewort $|||$ ergibt:

$$||| \xrightarrow{0} \# ||| \xrightarrow{1} || \# || \xrightarrow{1} ||| \# | \xrightarrow{1} |||| \# \xrightarrow{2} |||||$$

Allgemein wird die Funktion $f(|^n) = |^{2^n}$ berechnet.



Multiplikation

$$A_0 = \{ | \}; A = A_0 \cup \{ *, \# \}; X = A_0^* * A_0^*; Y = A_0^*$$

Der folgende Markov-Algorithmus berechnet die Funktion

$$f(|^n * |^m) = |^{n * m}:$$

k	φ	ψ	i	j	Kommentar
0	*	**	1	-	$ ^n * ^m \rightarrow ^n * * ^m$
1	ε	*	2	-	$ ^n * * ^m \rightarrow * ^n * * ^m$
2	**	#**	3	6	
3	#	#	4	5	Faktor vorn
4	ε		3	-	kopieren
5	#	ε	2	-	
6	*	*	6	7	1. Faktor löschen
7	***	ε	8	-	Hilfsmarkierung löschen



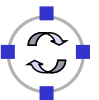
Mit der Eingabe $||| * ||$ ergibt sich z. B. folgende Berechnungsfolge:

$$||| * || \xrightarrow{0} ||| ** || \xrightarrow{1} * ||| ** ||$$

$$\xrightarrow{2} * ||| \# ** | \xrightarrow{3,4} | * || \# | * | \xrightarrow{3,4} || * | \# || ** | \xrightarrow{3,4,5} ||| * ||| ** |$$

$$\xrightarrow{2} ||| * ||| \# ** \xrightarrow{3,4} |||| * || \# | ** \xrightarrow{3,4} |||| * | \# || ** \xrightarrow{3,4,5} ||||| * ||| **$$

$$\xrightarrow{(2)6} ||||| * || ** \xrightarrow{6} ||||| * | ** \xrightarrow{6} ||||| * ** \xrightarrow{7} |||||$$

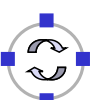


$$A_0 = \{ 0, 1 \}; A = A_0 \cup \{ *, \# \}; X = A_0^*; Y = A_0^* * A_0^*$$

Der folgende Markov-Algorithmus berechnet die Funktion

$$f(\mu) = \mu * \mu, \mu \in A_0$$

k	φ	ψ	i	j	Kommentar
0	ε	$* \#$	1	-	
1	$\# 0$	$0 \#$	2	3	kopiere nächstes Zeichen 0 vor *
2	$*$	$0 *$	1	-	und wiederhole
3	$\# 1$	$1 \#$	4	5	kopiere nächstes Zeichen 1 vor *
4	$*$	$1 *$	1	-	und wiederhole
5	$\#$	ε	6	-	



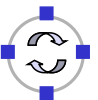
Mit der Eingabe 0 1 0 ergibt sich folgende Berechnung:

$$0\ 1\ 0 \xrightarrow{0} * \# 0\ 1\ 0 \xrightarrow{1} * \ 0 \# 1\ 0 \xrightarrow{2} 0 * 0 \# 1\ 0$$

$$\xrightarrow{(1)3} 0 * 0\ 1 \# 0 \xrightarrow{4} 0\ 1 * 0\ 1 \# 0$$

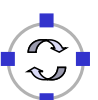
$$\xrightarrow{1} 0\ 1 * 0\ 1\ 0 \# \xrightarrow{2} 0\ 1\ 0 * 0\ 1\ 0 \#$$

$$\xrightarrow{(1,3)5} 0\ 1\ 0 * 0\ 1\ 0$$



Vor- und Nachteile von Markov-Algorithmen

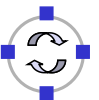
- ❑ Markov-Algorithmen arbeiten auf einem sehr einfachen Niveau:
 - ❑ Nur reine Zeichenketten werden unterstützt
 - ❑ „Höhere“ Datentypen wie Zahlen werden nicht unterstützt, sondern müssen als Zeichenketten kodiert werden
 - ❑ Dadurch wird die Formulierung von Markov-Algorithmen aufwändig
 - ❑ Auch werden Markov-Tafeln für größere Alphabete schnell unhandlich, da in jedem logischen Schritt Fallunterscheidungen für alle Symbole notwendig werden können
- ❑ Andererseits kann der einzelne Bearbeitungsschritt leicht mit Standardoperationen des Datentyps `String` in gängigen Programmiersprachen umgesetzt werden:
 - ❑ Suchen des ersten Auftretens eines Teil-Strings und Ersetzen des Teil-Strings sind elementare String-Operationen
 - ❑ Markov-Interpreter können somit leicht realisiert werden



- ❑ Ausdrucksfähigkeit der verschiedenen Modelle
 - ❑ Leisten imperative Algorithmen mehr als Registermaschinen?
- ❑ Kann man Leistung von Algorithmenmodellen vergleichen?

Die Klasse der **intuitiv berechenbaren** Funktionen stimmt mit den formalen Klassen der (Registermaschinen-, imperativ, applikativ, Markov-, etc.) berechenbaren Funktionen überein.

(Diese These ist prinzipiell nicht beweisbar, da sich allein schon der Begriff „intuitiv“ nicht formal fassen lässt.)



Ein Algorithmenmodell heißt **universell**, wenn damit alle berechenbaren Funktionen beschrieben werden können.

Eigenschaften universeller Sprachen (u. a. alle Programmiersprachen)

- ❑ Der nutzbare Bereich für Daten / Parameterwerte ist nicht beschränkt (hierfür reicht der Datentyp `integer` ohne Begrenzung durch ein `maxint` aus).
- ❑ Konzepte wie Rekursion oder Iteration (**while**) mit bedingten Sprüngen erlauben eine (bedingte und potentiell unendliche) Wiederholung von Teilaufgaben
- ❑ **Satz:** *Die Algorithmenmodelle applikative, imperative, Markov- und Registermaschinen-Algorithmen sind universell.*
(Beweise solcher Aussagen werden in Vorlesungen zur Theoretischen Informatik behandelt)



- ❑ Formale Modelle
 - ❑ Registermaschine
 - ❑ Abstrakte Maschinen
 - ❑ Markov-Algorithmen
- ❑ Ausdrucksfähigkeit: Church'sche These
- ❑ Literatur: Saake/Sattler: *Algorithmen und Datenstrukturen*, Kapitel 6

