Grundzüge der Theoretischen Informatik

Markus Bläser

Universität des Saarlandes

10.12.2021

Kapitel 16: Turingmaschinen

Berechnungen

$C \vdash C'$

- $C = (q, (p_1, x_1), \ldots (p_k, x_k))$
- $C' = (q', (p'_1, x'_1), \ldots (p'_k, x'_k))$
- $x_\kappa = u_\kappa \alpha_\kappa v_\kappa$, wobei $|u_\kappa| = p_\kappa - 1$ und $\alpha_\kappa \in \Gamma$, $1 \leq \kappa \leq k$.

C' heißt *Nachfolgekonfiguration* von C, falls C' durch einen Schritt von M von C erreicht wird.

D.h. falls $\delta(q, \alpha_1, \ldots, \alpha_k) = (q', \beta_1, \ldots, \beta_k, r_1, \ldots, r_k)$, dann ist

$$x'_\kappa = u_\kappa \beta_\kappa v_\kappa, \quad 1 \leq \kappa \leq k$$

und

$$p'_\kappa = \begin{cases} p_\kappa - 1 & \text{falls } r_\kappa = L, \\ p_\kappa & \text{falls } r_\kappa = S, \\ p_\kappa + 1 & \text{falls } r_\kappa = R. \end{cases}$$

Berechnungen (2)

Randfälle:

- Falls $p_\kappa = 1$ und $r_\kappa = L$, dann ist

$$x'_\kappa = \square \beta_\kappa \nu_\kappa$$

und

$$p'_\kappa = 1.$$

- Falls $p_\kappa = |x_\kappa|$ and $r_\kappa = R$, dann ist

$$x'_\kappa = u_\kappa \beta_\kappa \square$$

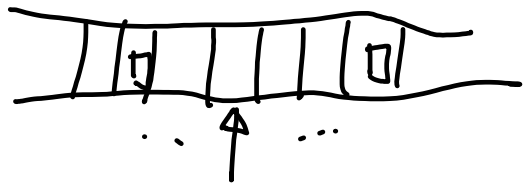
und

$$p'_\kappa = |x_\kappa| + 1.$$

Berechnungen (3)

- Notation: $C \vdash_M C'$
- $\vdash_M^*$ bezeichnet die reflexiv-transitive Hülle
- $C \vdash_M^* C'$ falls es $C_1, \ldots, C_\ell$ gibt mit $C \vdash_M C_1 \vdash_M \ldots \vdash_M C_\ell \vdash_M C'$.
- Eine Konfiguration ohne Nachfolger heißt *haltend*.
- M hält auf w, falls $\mathrm{SC}_M(w) \vdash_M^* C_t$ und C_t ist haltend.
- $\mathrm{SC}_M(w) \vdash_M C_1 \vdash_M C_2 \vdash_M \ldots \vdash_M C_t$ heißt *Berechnung* von M auf w.
- Falls C_t nicht existiert, so hält M nicht auf w.
  Die zugehörige Berechnung ist unendlich.

Berechnungen (4)



- Sei $SC_M(w) \vdash_M^* C_t$, $C_t = (q, (p_1, x_1), \ldots, (p_k, x_k))$ haltend.
- Sei $i \leq p_1$ der größte Index mit $x_1(i) = \square$.
  ($i = 0$ falls der Index nicht existiert.)
- Sei $j \geq p_1$ der kleinste Index mit $x_1(j) = \square$.
  ($j = |x_1| + 1$, falls der Index nicht existiert.)
- $x_1(i+1)x_1(i+2)\ldots x_1(j-1)$ ist die *Ausgabe* von M auf w.
- Berechnete Funktion: $\varphi_M : \Sigma^* \to (\Gamma \setminus \{\square\})^*$

$$\varphi_M(w) = \begin{cases} \text{Ausgabe von } M \text{ auf } w & \text{falls } M \text{ auf } w \text{ hält,} \\ \text{undefiniert} & \text{sonst.} \end{cases}$$

Berechnete Funktionen und Sprachen

Definition (16.3)

$f : \Sigma^* \to \Sigma^*$ ist *Turing-berechenbar*, falls $f = \varphi_M$ für eine Turingmaschine $M = (Q, \Sigma, \Gamma, \delta, q_0)$.

- Wir könnten $L \subseteq \Sigma^*$ Turing-entscheidbar nennen, falls $\chi_L : \Sigma^* \to \{0, 1\}$ Turing-berechenbar ist. (0, 1 als Elemente von Σ aufgefasst.)
- Stattdessen arbeiten wir mit akzeptierenden Zuständen $Q_{acc} \subseteq Q$.
- Eine haltende Konfiguration $(q, (p_1, x_1), \ldots, (p_k, x_k))$ heißt *akzeptierend*, falls $q \in Q_{acc}$. Sonst heißt sie *verwerfend*.

Berechnete Funktionen und Sprachen (2)

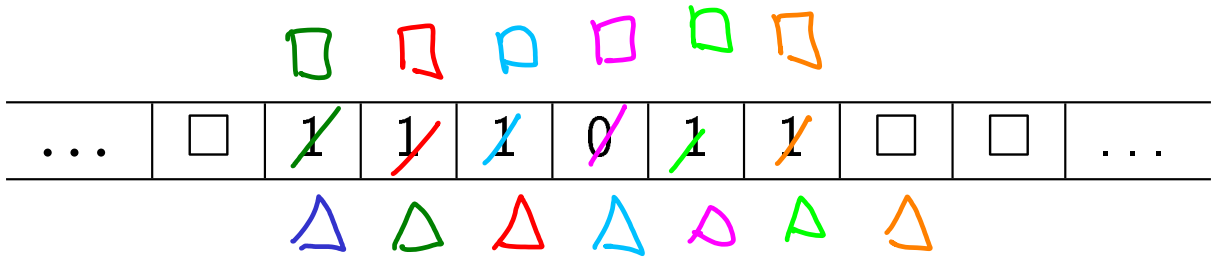
Definition (16.4)

Sei $L \subseteq \Sigma^*$.

1. $M = (Q, \Sigma, \Gamma, \delta, q_0, Q_{acc})$ *erkennt* $L \subseteq \Sigma^*$, falls für alle $w \in L$ die Berechnung von M in einer akzeptierenden Konfiguration endet und für alle $w \notin L$ nicht.
   (D.h. sie endet entweder in einer verwerfenden Konfiguration oder M hält nicht auf w.)
2. M *entscheidet* L, falls zusätzlich M auch auf alle $w \notin L$ hält.
3. $L(M)$ bezeichnet die von M erkannte Sprache.

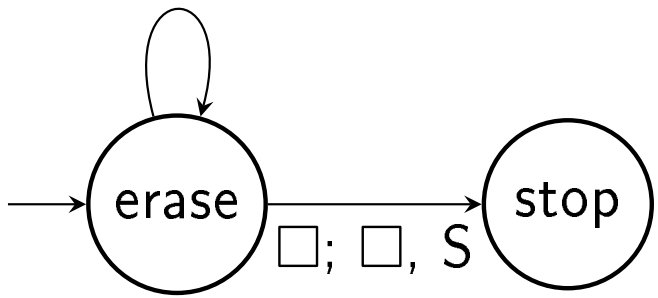
Kapitel 17: Beispiele, Tricks und syntaktischen Zucker

Die Turingmaschine ERASE

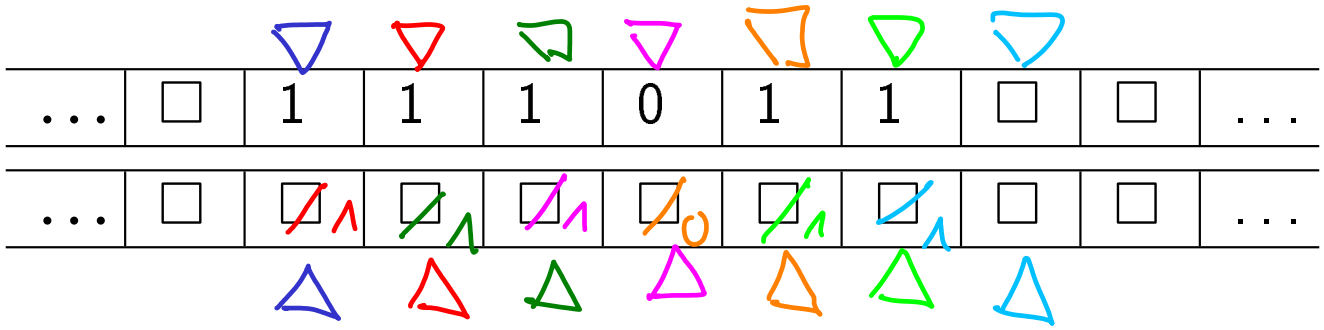


1; □, R

0; □, R



Die Turingmaschine COPY



Band 1 | Band 2

1; 1, R | □; 1, R

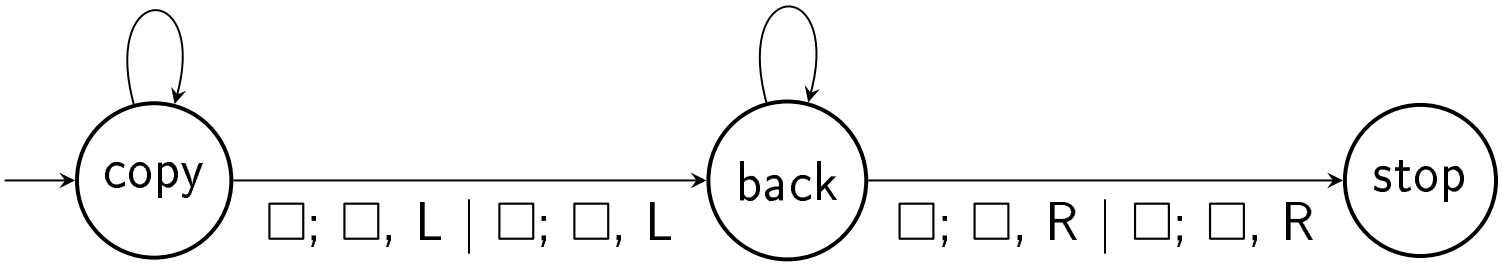
0; 0, R | □; 0, R

0; 0, L | 0; 0, L

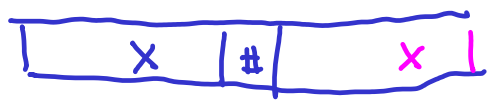
~~0; 0, L~~ | ~~1; 1, L~~

~~1; 1, L~~ | ~~0; 0, L~~

1; 1, L | 1; 1, L

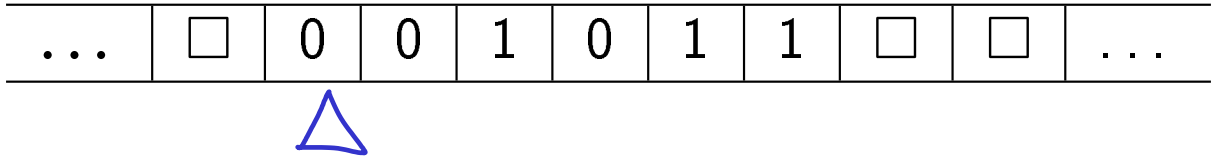


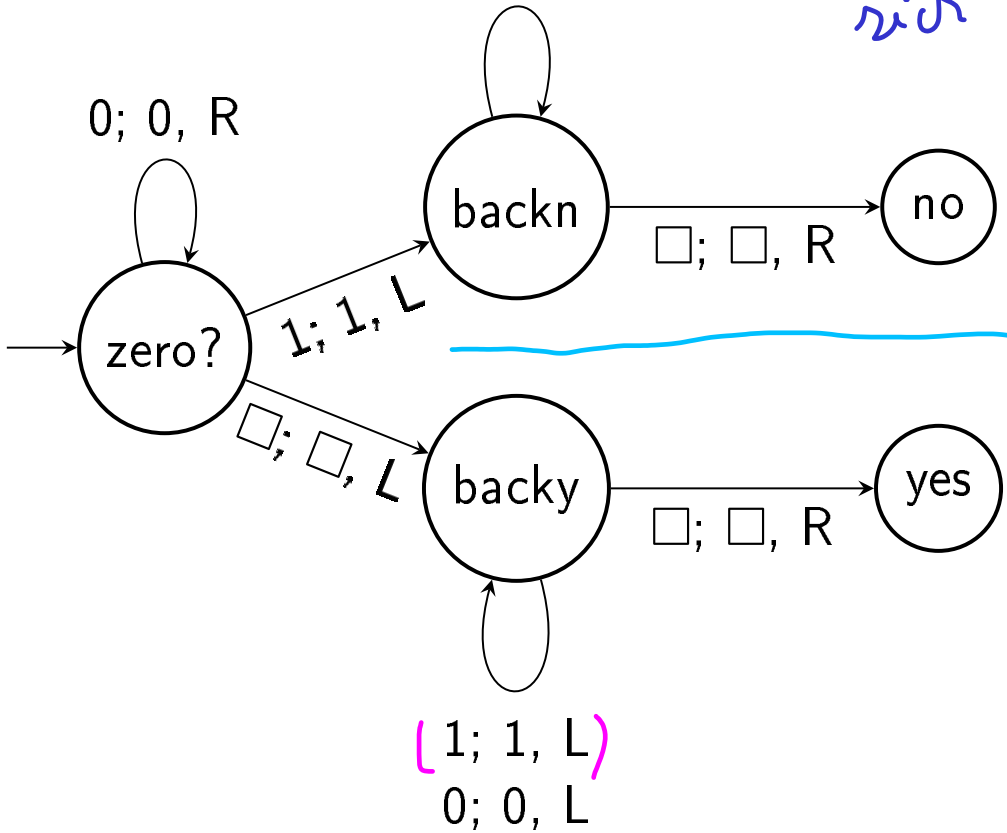
Copy auf einer Band



?

Die Turingmaschine COMPARE

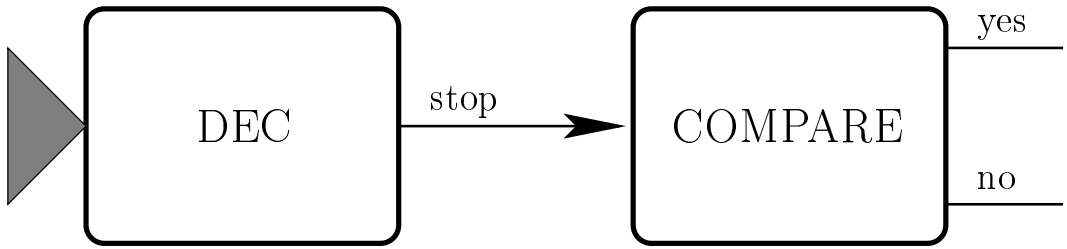


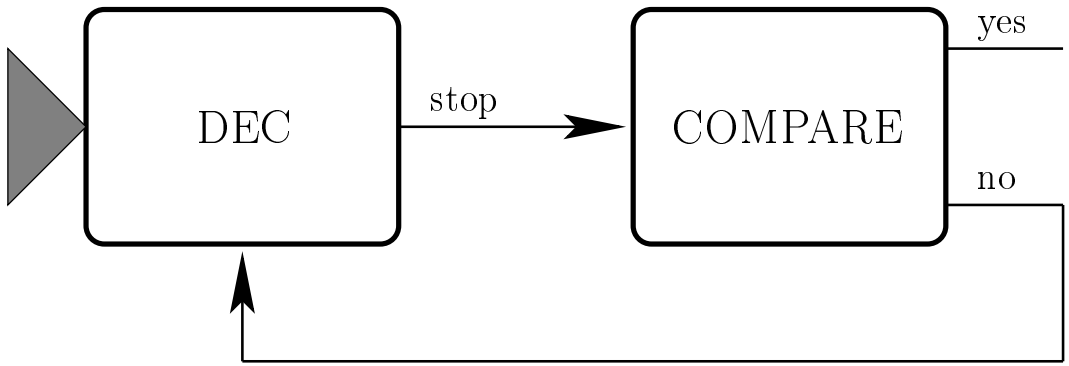


"Die TM merkt sich etwas in Zustand"

Konkatenation von Turingmaschinen

"Starte in Startzustand von DEC. Wenn der Zustand stop erreicht wird, gehe in den Startzustand von COMPARE"





Zeiler verbinden → "Schleife"

Parallele Ausführung

- k-Band-TM $M = (Q, \Sigma, \Gamma, \delta, q_0)$
- k'-Band-TM $M' = (Q', \Sigma, \Gamma, \delta', q_0')$
- $(k + k')$-Band-TM, die M und M' parallel simuliert

Übergangsfunktion:

$$\Delta : (Q \times Q') \times \Gamma^{k+k'} \to (Q \times Q') \times \Gamma^{k+k'} \times \{L, S, R\}^{k+k'},$$

ist definiert durch

$$\begin{aligned}&\Delta((q, q'), \gamma_1, \ldots, \gamma_{k+k'}) \\ &\quad = ((p, p'), \alpha_1, \ldots, \alpha_k, \alpha'_1, \ldots, \alpha'_{k'}, r_1, \ldots, r_k, r'_1, \ldots, r'_{k'})\end{aligned}$$

falls

$$\delta(q, \gamma_1, \ldots, \gamma_k) = (p, \alpha_1, \ldots, \alpha_k, r_1, \ldots, r_k) \text{ und}$$
$$\delta'(q', \gamma_{k+1}, \ldots, \gamma_{k+k'}) = (p', \alpha'_1, \ldots, \alpha'_{k'}, r'_1, \ldots, r'_{k'})$$

WHILE-berechenbar $\quad \mathbb{N} \to \mathbb{N}$

$\Leftrightarrow \quad \Leftrightarrow$

Turing-berechenbar $\quad \Sigma^* \to \Sigma^*$

Kapitel 18: Die Church-Turing-These

While-Berechenbarkeit und Turing-Berechenbarkeit

While-berechenbar = Turing-berechenbar

Identifizieren $\mathbb{N}$ mit $\{0,1\}^*$:

- $\mathrm{cod} : \{0,1\}^* \to \mathbb{N}$
- $\mathrm{cod}(x) = \mathrm{bin}^{-1}(1x) - 1$

Bijektion

Identifizieren $\mathbb{N} \to \mathbb{N}$ mit $\{0,1\}^* \to \{0,1\}^*$:

- Zu $f : \mathbb{N} \to \mathbb{N}$ definiere $\hat{f} : \{0,1\}^* \to \{0,1\}^*$ durch

$$\hat{f}(x) = \mathrm{cod}^{-1}(f(\mathrm{cod}(x))) \quad \text{für alle } x \in \{0,1\}^*.$$

- Zu $g : \{0,1\}^* \to \{0,1\}^*$, definiere $\hat{g} : \mathbb{N} \to \mathbb{N}$ durch

$$\hat{g}(n) = \mathrm{cod}(g(\mathrm{cod}^{-1}(n))) \quad \text{für alle } n \in \mathbb{N}.$$

$$\hat{\hat{f}} = f \text{ und } \hat{\hat{g}} = g$$

$\mathbb{N} \cong \{0,1\}^*$

$n \mapsto bin(n)$ injektiv

00 1100

$\{0,1\}^* \to \mathbb{N}$

$x_1 \dots x_n$

$\to$ < Anzahl führende Nullen, Rest >

$\{0,1\}^* \to \mathbb{N}$

$x_1 \dots x_n \mapsto bin^{-1}(1 x_1 \dots x_n)$

$$
\begin{array}{ccc}
F: & \mathbb{N} & \to & \mathbb{N} \\
 & \uparrow cod & = & \downarrow cod^{-1} \\
\hat{F}: & \{0,1\}^* & \to & \{0,1\}^*
\end{array}
$$

$$
\begin{array}{ccc}
g: & \{0,1\}^* & \to & \{0,1\}^* \\
 & \uparrow cod^{-1} & = & \downarrow cod \\
\hat{g}: & \mathbb{N} & \to & \mathbb{N}
\end{array}
$$

GOTO-Programme

Ein GOTO-Programm ist eine Folge

$$(1, s_1), (2, s_2), \ldots, (m, s_m)$$

wobei jedes s_μ eine Anweisung der Form

1. $x_i = x_j + x_k$ oder
2. $x_i = x_j - x_k$ oder
3. $x_i := c$ oder
4. **if** $x_i \neq 0$ **then goto** λ

ist.

Das Programm terminiert, wenn eine nicht vorhandene Zeile erreicht wird.

Von WHILE nach GOTO

Lemma (18.2)

Für jedes WHILE-Programm P *gibt es ein GOTO-Programm* Q *mit* $\varphi_P = \varphi_Q$.

1: **while** $x_i \neq 0$ **do**
2: $\quad P$
3: **od**

1: **if** $x_i \neq 0$ **then goto** 3
2: **goto** 5 ← syntaktische Zucker
3: P
4: **goto** 1
5: ...

Von GOTO zu Turingmaschinen

Lemma (18.3)

Sei $f : \mathbb{N} \to \mathbb{N}$. *Falls* f *GOTO-berechenbar ist, dann ist* $\hat{f}$ *Turing-berechenbar.*

INC DEC ERASE

- einfache Anweisungen: $x_i{+}{+}$, $x_i{-}{-}$ und $x_i := 0$.
- Jede Variable wird durch ein Band dargestellt.
- Der Inhalt steht in binär von links nach rechts.
- Schrittweise Simulation
- Invariante: Zu Beginn der Simulation eines Schrittes stehen die Köpfe auf der Einerstelle.

$x_i := x_j + x_z$

$x_i := x_j$

while $x_z \neq 0$ do

x_i++

x_z--

od

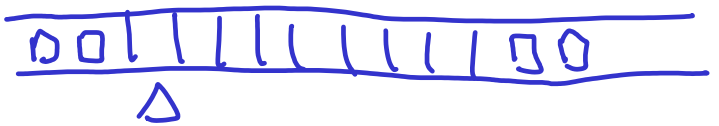
$x_i := c$

$x_i := 0$

x_i++ ; ... ; x_i++

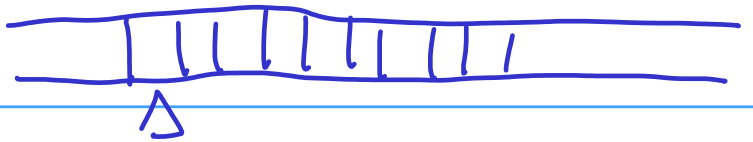
(c)

x_0



Inhalt von x_0
(in binär)

x_0



Beispiel

1: **if** $x_0 \neq 0$ **then goto** 3
2: $x_0{+}{+}$
3: ...