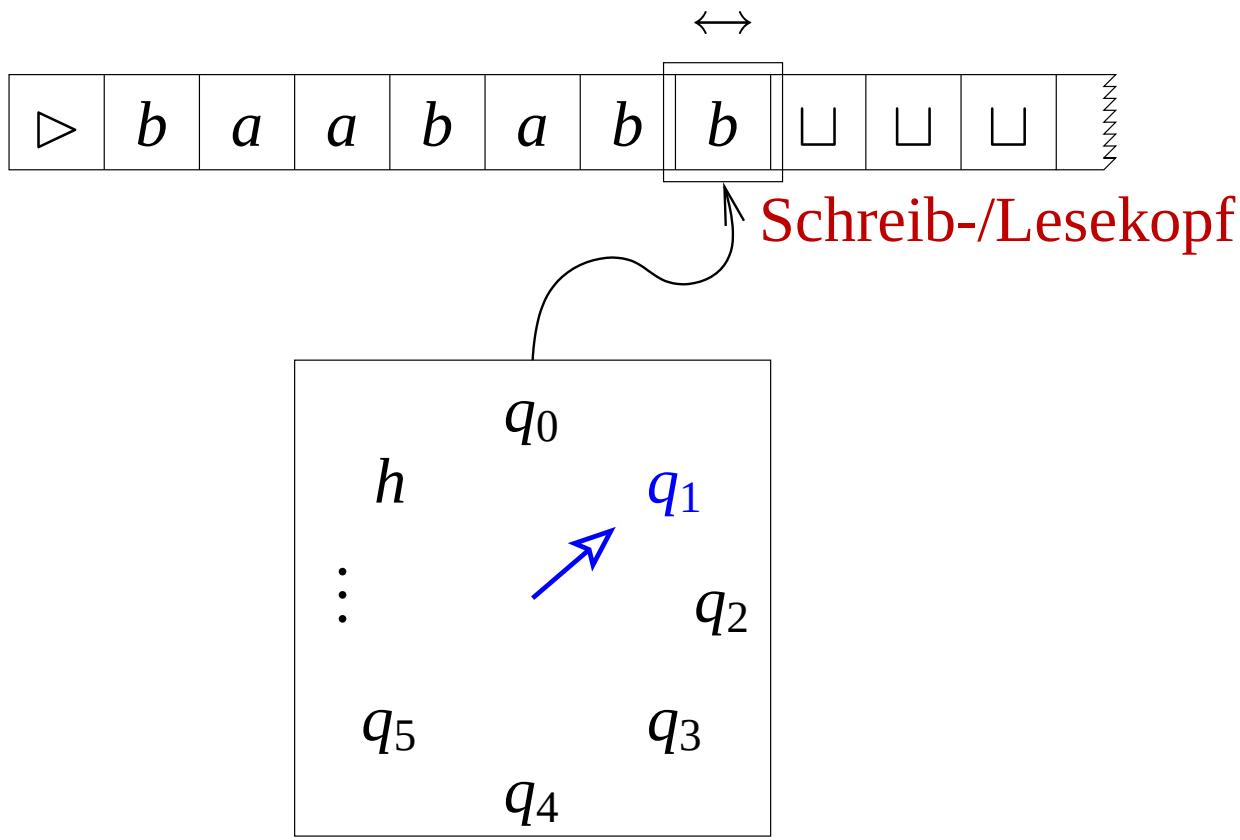
Turingmaschinen



(Deterministische) Turingmaschine

$$(K, \Sigma, \delta, s, H)$$

K endliche Menge von Zuständen

Σ Alphabet, $\sqcup, \triangleright \in \Sigma$, $\leftarrow, \rightarrow \notin \Sigma$

s $s \in K$; Startzustand

H $H \subseteq K$; Menge der Endzustände (Haltezustände)

δ Übergangsfunktion

$\delta : (K - H) \times \Sigma \mapsto K \times (\Sigma \cup \{\leftarrow, \rightarrow\})$,
so dass für alle $q \in K - H$ und $a \in \Sigma$ gilt:
falls $\delta(q, \triangleright) = (p, b)$, dann ist $b = \rightarrow$
falls $\delta(q, a) = (p, b)$, dann ist $b \neq \triangleright$

Sei $M = (K, \Sigma, \delta, s, H)$ eine Turingmaschine. Ein Element von

$$K \times \triangleright\Sigma^* \times (\Sigma^*(\Sigma - \{\sqcup\}) \cup \{\varepsilon\})$$

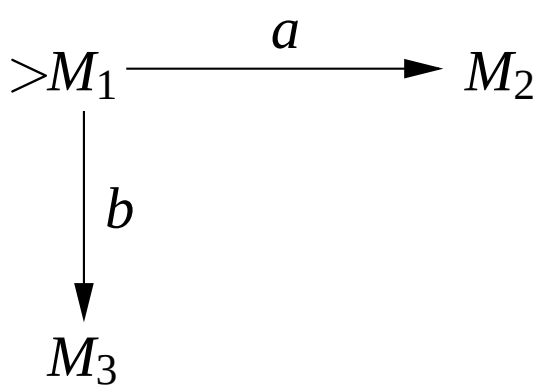
heißt Konfiguration von M. Eine Konfiguration (q, wa, u) heißt Haltekonfiguration, falls $q \in H$.
Abkürzende Schreibweise: $(q, w\underline{a}u)$ statt (q, wa, u)

$$(q_1, w_1\underline{a_1}u_1) \vdash_M (q_2, w_2\underline{a_2}u_2)$$

Diagrammnotation

Kombination von Turingmaschinen

$$M_i = (K_i, \Sigma, \delta_i, s_i, H_i), i = 1,2,3,\ K_i \cap K_j = \emptyset \text{ für } i \neq j$$



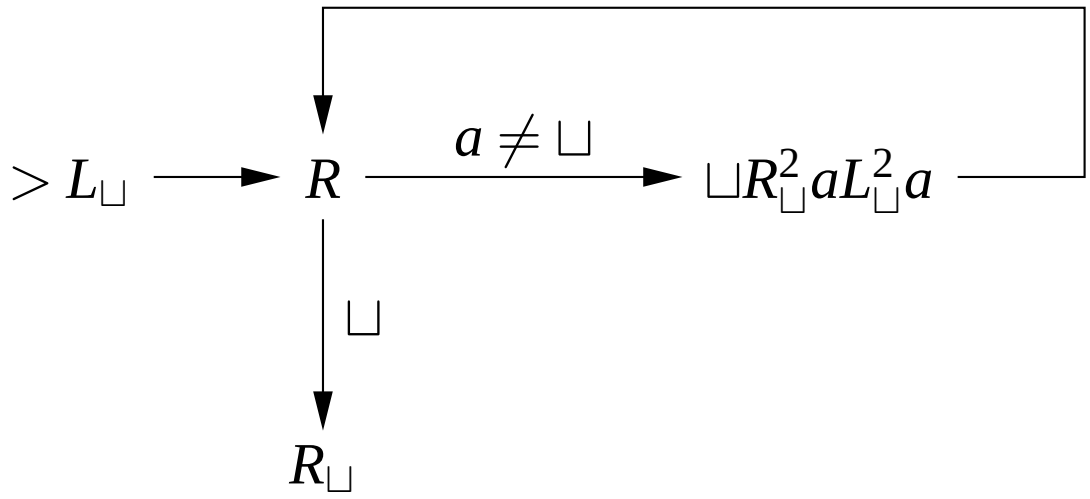
$$M = (K, \Sigma, \delta, s, H)$$

$$K = K_1 \cup K_2 \cup K_3$$
$$s = s_1,\ H = H_2 \cup H_3$$

$$q \in K_i - H_i\text{:}\ \ \delta(q,a) = \delta_i(q,a)$$

$$q \in H_1\text{:}\ \ \delta(q,a) = s_2,\ \ \delta(q,b) = s_3$$

Beispiel:



$$(\sqcup w \underline{\sqcup}) \vdash_M^* (\sqcup w \sqcup w \underline{\sqcup})$$

Turingmaschinen-Berechnungen

Rekursive Sprachen

$$M = (K, \Sigma, \delta, s, H = \{y, n\}) \qquad w \in (\Sigma - \{\sqcup, \triangleright\})^*$$

$(s, \triangleright \underline{\sqcup} w)$	Startkonfiguration
$(y, \ldots)$	akzeptierende Haltekonfiguration
$(n, \ldots)$	verwerfende Haltekonfiguration

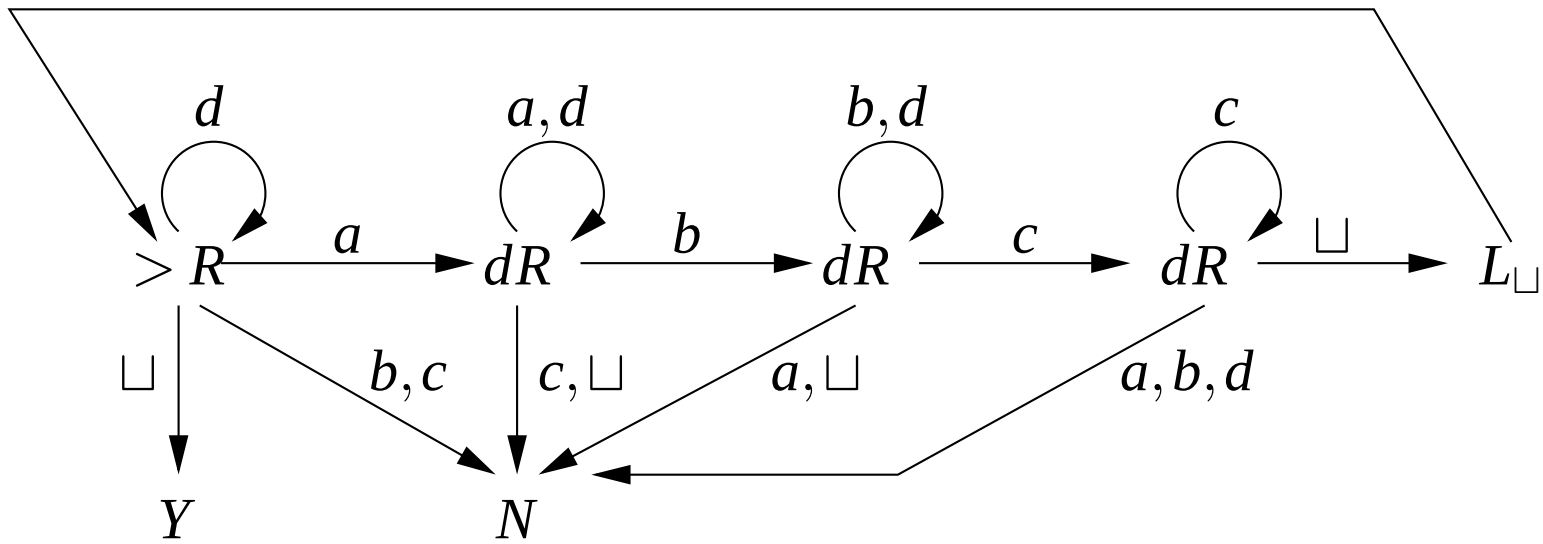
M akzeptiert Eingabe w, falls M aus $(s, \triangleright \underline{\sqcup} w)$ in eine akzeptierende Haltekonfiguration übergeht. M verwirft Eingabe w, falls M aus $(s, \triangleright \underline{\sqcup} w)$ in eine verwerfende Haltekonfiguration übergeht.

$$(s, \triangleright \underline{\sqcup} w) \vdash_M^* \ldots$$

Berechnung ist akzeptierend
oder Berechnung ist verwerfend
oder Berechnung ist unendlich

M entscheidet (decides) eine Sprache $L \subseteq \Sigma_0^*$ mit $\Sigma_0 \subseteq \Sigma - \{\sqcup, \triangleright\}$, falls für jedes Wort w aus Σ_0^* gilt: Falls $w \in L$, so akzeptiert M das Wort w, und falls $w \notin L$, so verwirft M das Wort w. Eine Sprache L heißt rekursiv, falls es eine deterministische Turingmaschine gibt, die L entscheidet.

Die Sprache $a^n b^n c^n$ ist turing-entscheidbar:



Rekursive Funktionen

$$M = (K, \Sigma, \delta, s, H = \{h\}), \quad \Sigma_0 \subseteq \Sigma - \{\sqcup, \triangleright\}, \quad w \in \Sigma_0^*$$

Falls

$$(s, \triangleright \underline{\sqcup} w) \vdash_M^* (h, \triangleright \underline{\sqcup} v)$$

so heißt v die Ausgabe von M bei der Eingabe w und wird als $M(w)$ bezeichnet.

M berechnet eine Funktion $f : \Sigma_0^* \mapsto \Sigma_0^*$, falls $M(w) = f(w)$ für alle $w \in \Sigma_0^*$. Eine Funktion f heißt rekursiv, falls es eine deterministische Turingmaschine M gibt, die f berechnet.

Binärnotation:

$$w = a_1 a_2 \ldots a_n \in \{0,1\}^*$$

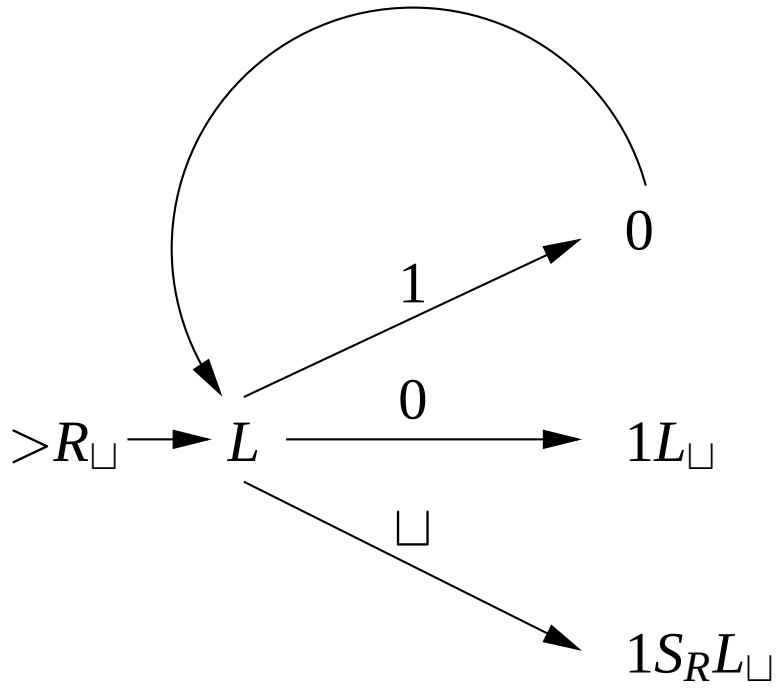
$$\mathsf{num}(w) = a_1 \cdot 2^{n-1} + a_2 \cdot 2^{n-2} + \cdots + a_n$$

$M = (K, \Sigma, \delta, s, H = \{h\})$ berechnet $f : \mathbb{N}^k \mapsto \mathbb{N}$, falls für alle $w_1, \ldots, w_k \in 0 \cup 1(0 \cup 1)^*$ gilt

$$\mathsf{num}(M(w_1; \ldots; w_k)) = f(\mathsf{num}(w_1), \ \ldots, \ \mathsf{num}(w_k))$$

Eine Funktion $f : \mathbb{N}^k \mapsto \mathbb{N}$ heißt rekursiv, falls es eine deterministische Turingmaschine M gibt, die f berechnet.

Beispiel:



$$\mathsf{succ}(n) = n + 1$$

Rekursiv aufzählbare Sprachen

$$M = (K, \Sigma, \delta, s, H), \quad \Sigma_0 \subseteq \Sigma - \{\sqcup, \triangleright\}, \quad L \subseteq \Sigma_0^*$$

M akzeptiert (semi-entscheidet (semidecides)) L, falls gilt:

$$M \text{ hält bei Eingabe } w \iff w \in L$$

Eine Sprache L heißt rekursiv aufzählbar genau dann wenn es eine deterministische Turingmaschine M gibt, die L akzeptiert. Wir sagen dann auch, dass M ein Semi-Entscheidungsverfahren für L ist.

Notation:

$M(w) = \nearrow$ genau dann wenn M bei Eingabe w nicht hält.

Satz: Falls eine Sprache eine rekursive Sprache ist, so ist sie auch rekursiv aufzählbar.

□

Satz: Falls eine Sprache L eine rekursive Sprache ist, so ist auch ihr Komplement $\overline{L}$ eine rekursive Sprache.

□

$$(s, \triangleright \underline{\sqcup} w) \vdash_M^* \ldots$$

Berechnung ist akzeptierend
oder Berechnung ist verwerfend
oder Berechnung ist unendlich

M entscheidet (decides) eine Sprache $L \subseteq \Sigma_0^*$ mit $\Sigma_0 \subseteq \Sigma - \{\sqcup, \triangleright\}$, falls für jedes Wort w aus Σ_0^* gilt: Falls $w \in L$, so akzeptiert M das Wort w, und falls $w \notin L$, so verwirft M das Wort w. Eine Sprache L heißt rekursiv, falls es eine Turingmaschine gibt, die L entscheidet.

Rekursiv aufzählbare Sprachen

$$M = (K, \Sigma, \delta, s, H), \quad \Sigma_0 \subseteq \Sigma - \{\sqcup, \triangleright\}, \quad L \subseteq \Sigma_0^*$$

M akzeptiert (semi-entscheidet (semidecides)) L, falls gilt:

$$M \text{ hält bei Eingabe } w \iff w \in L$$

Eine Sprache L heißt rekursiv aufzählbar genau dann wenn es eine Turingmaschine M gibt, die L akzeptiert. Wir sagen dann auch, dass M ein Semi-Entscheidungsverfahren für L ist.

Notation:

$M(w) = \nearrow$ genau dann wenn M bei Eingabe w nicht hält.

Satz: Falls eine Sprache eine rekursive Sprache ist, so ist sie auch rekursiv aufzählbar.

Beweisidee: $\delta'(n,a) = (n,a)$ □

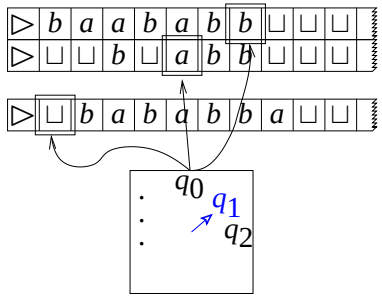
Satz: *Falls eine Sprache L eine rekursive Sprache ist, so ist auch ihr Komplement $\overline{L}$ eine rekursive Sprache.*

Beweisidee:

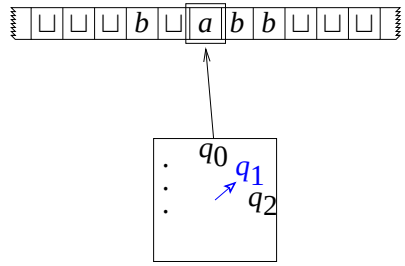
$$\delta'(q,a) = \begin{cases} (n,\ldots) & \text{falls } \delta(q,a) = (y,\ldots) \\ (y,\ldots) & \text{falls } \delta(q,a) = (n,\ldots) \\ \delta(q,a) & \text{sonst} \end{cases}$$

□

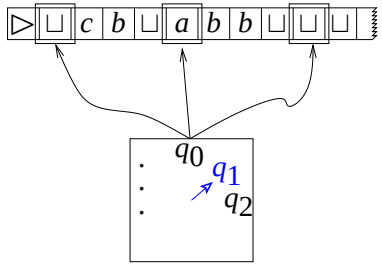
Variationen von Turingmaschinen



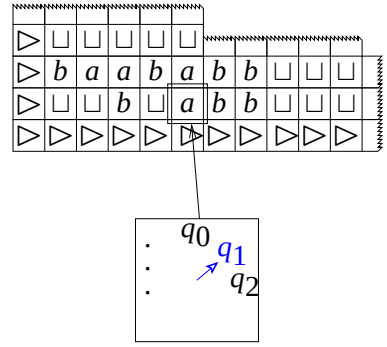
mehrere Bänder



beidseitig unendliches Band

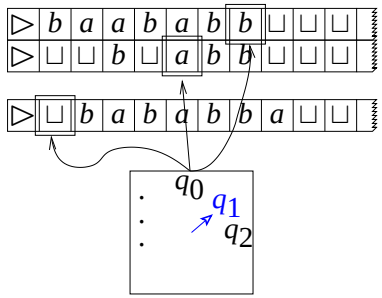


mehrere Köpfe



zweidimensionales Band

k-Band Turingmaschine



$$(K, \Sigma, \delta, s, H)$$

$$\delta : (K - H) \times \Sigma^k \mapsto K \times (\Sigma \cup \{\leftarrow, \rightarrow\})^k$$

Konfiguration: $(q,\ w_1\underline{a_1}u_1,\ \ldots,\ w_k\underline{a_k}u_k)$

Startkonfiguration: Eingabe auf dem ersten Band, übrige Bänder leer: $(s,\ \triangleright\underline{\sqcup}w,\ \triangleright\underline{\sqcup},\ \ldots,\ \triangleright\underline{\sqcup})$

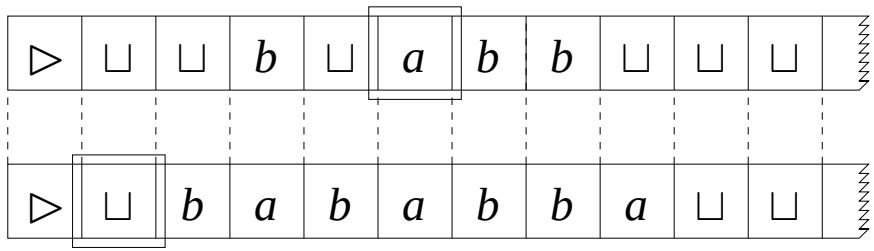
Jede k-Band Turingmaschine kann durch eine Standard-turingmaschine mit nur einem Band simuliert werden:

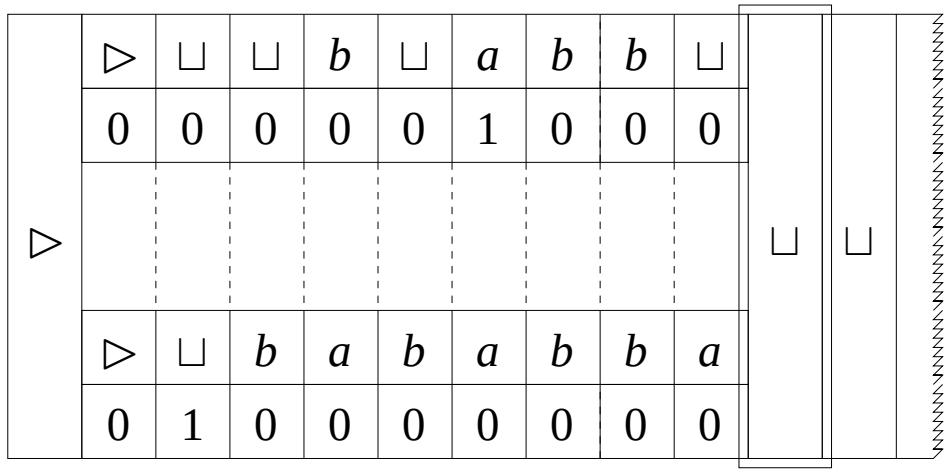
Satz:

(a) Für jede k-Band Turingmaschine $M = (K, \Sigma, \delta, s, H)$ *gibt es eine Turingmaschine* $M' = (K', \Sigma', \delta', s', H)$, *so dass für alle* $x \in \Sigma^*$ *gilt: M hält bei Eingabe x mit Ausgabe v auf ihrem ersten Band genau dann wenn* M' *im gleichen Zustand mit Ausgabe v hält.*

(b) Falls M bei Eingabe x nach t Schritten hält, so hält M' *nach* $O(t \cdot (|x| + t))$ *Schritten.*

Beweisidee: $\Sigma' = \Sigma \cup (\Sigma \times \{0,1\})^k$





Simulationsphasen:

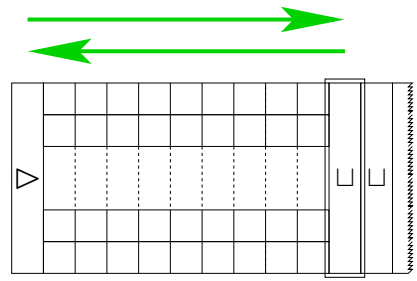
(1) Umbau der Eingabe:

$$(s, \triangleright \underline{\sqcup} w) \vdash_M^* (q, \triangleright \omega \underline{\sqcup})$$

mit $w \in \Sigma^*$ und $\omega \in ((\Sigma \times \{0,1\})^k)^*$

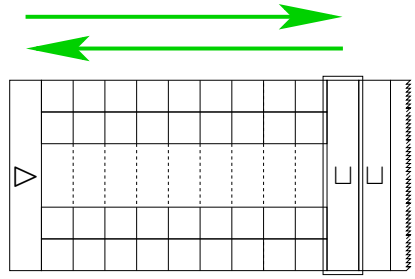
(2) Schritt für Schritt Simulation:

(a) Aufsammeln der Symbole unter den Schreib-/Leseköpfen (Speichern der gelesenen Symbole im Zustand).



(b) Eigentliche Zustandsänderung

(c) Ausführen der Schreib- und Bewegekopfaktionen (gespeichert im Zustand) auf den virtuellen Bändern. Dabei eventuell Ersetzen von $\sqcup$ durch Symbol aus $(\{\sqcup\} \times \{0,1\})^k$.

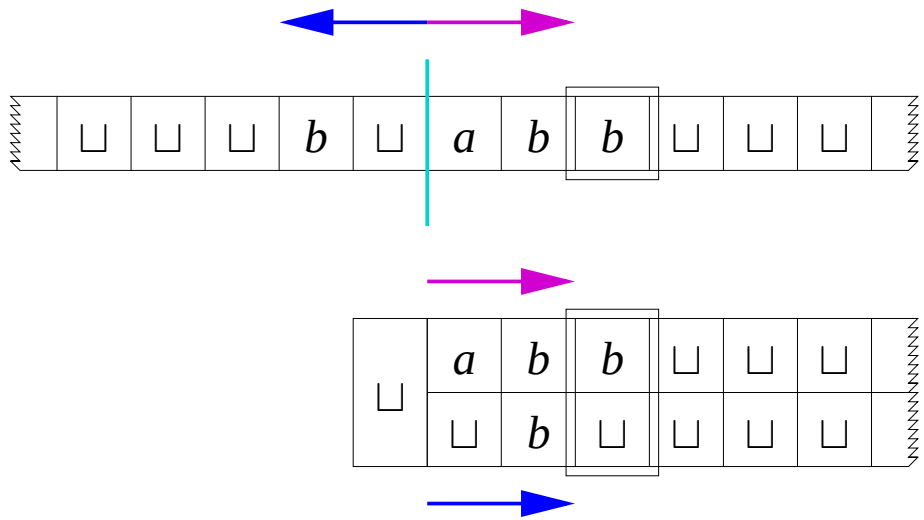


(3) Umbau der Ausgabe.

□

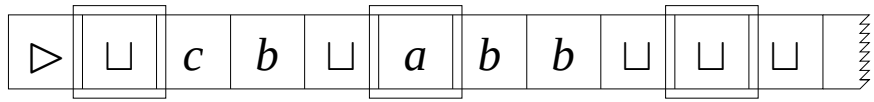
Turingmaschine mit beidseitig unendlichem Band

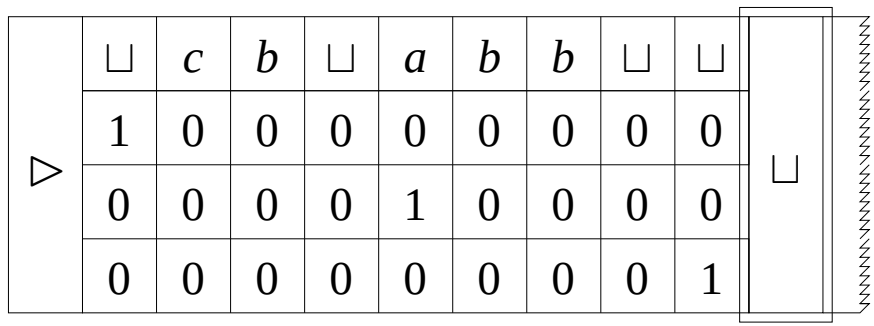
Jede Turingmaschine mit einem beidseitig unendlichem Band kann durch eine Standardturingmaschine mit einem einseitig unendlichen Band simuliert werden:



Turingmaschine mit k Köpfen

Jede Turingmaschine mit mehreren Köpfen kann durch eine Turingmaschine mit $k+1$ Bändern und somit durch eine Standardturingmaschine simuliert werden:

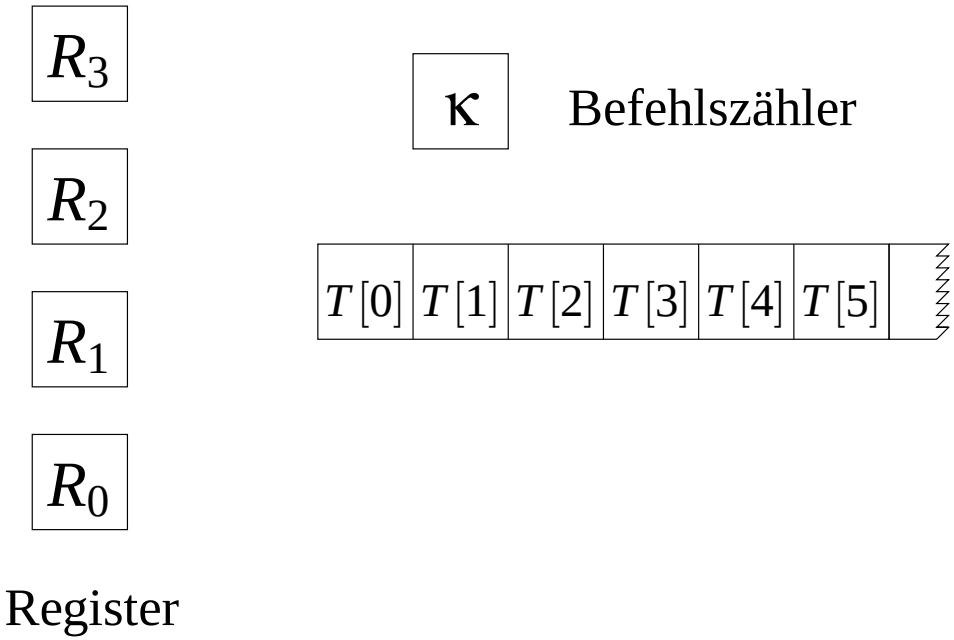




Satz: Jede rekursive Sprache, rekursiv aufzählbare Sprache oder rekursive Funktion, die von einer Turingmaschine mit mehreren einseitig oder beidseitig unendlichen oder auch mehrdimensionalen Bändern und mehreren Köpfen entschieden, akzeptiert beziehungsweise berechnet wird, wird auch von einer Standardturingmaschine mit nur einem einseitig unendlichen Band entschieden, akzeptiert beziehungsweise berechnet.

□

Registerturingmaschinen



Eine Registerturingmaschine ist ein Paar $M = (k, \Pi)$, wobei k die Anzahl der Register ist und $\Pi = (\pi_1, \pi_2, \ldots, \pi_p)$ eine endliche Folge von Befehlen (instructions).

Jeder Befehl π_i ist von einem der folgenden Typen:

Befehl	Operand	Semantik
read	j	$R_0 := T[R_j]$
write	j	$T[R_j] := R_0$
store	j	$R_j := R_0$
load	j	$R_0 := R_j$
load	$= c$	$R_0 := c$
add	j	$R_0 := R_0 + R_j$
add	$= c$	$R_0 := R_0 + c$
sub	j	$R_0 := \max(R_0 - R_j, 0)$
sub	$= c$	$R_0 := \max(R_0 - c, 0)$
half		$R_0 := \lfloor \frac{R_0}{2} \rfloor$
jump	s	$\kappa := s$
jpos	s	if $R_0 > 0$ then $\kappa := s$
jzero	s	if $R_0 = 0$ then $\kappa := s$
halt		$\kappa = 0$

Konfiguration einer Registerturingmaschine

$$(\kappa, R_0, R_1, \ldots, R_{k-1}, T)$$

κ	Befehlszähler; $0 \leq \kappa \leq p$
R_j	aktuelle Wert von Register R_j für $j = 0, \ldots, k-1$
T	Bandinhalt: Endliche Menge von Paaren aus $\mathbb{N} \times (\mathbb{N} - \{0\})$, so dass es für alle $i \geq 0$ höchstens ein Paar der Form (i, m) gibt

Eine Konfiguration heißt Haltekonfiguration, falls $\kappa = 0$.

Übergänge einer Registerturingmaschine

$$M = (k, (\pi_1, \pi_2, \ldots, \pi_p))$$

$$(\kappa, R_0, R_1, \ldots, R_{k-1}, T) \vdash_M (\kappa', R'_0, R'_1, \ldots, R'_{k-1}, T')$$

genau dann wenn

π_κ ist vom Typ read j und

$\kappa' = \kappa + 1$; $R'_0 = m$, falls $(R_j, m) \in T$, und $R'_0 = 0$ sonst;
$T' = T$; $R'_i = R_i$ für $i = 1, \ldots, k-1$.

oder π_κ ist vom Typ write j und

$\kappa' = \kappa + 1$; $R'_i = R_i$ für $i = 0, \ldots, k-1$; $T' = T \cup \tau -$
$\{(R_j, m)\}$, falls es ein $(R_j, m) \in T$ gibt, und $T' = T \cup \tau$
sonst, wobei $\tau = \{(R_j, R_0)\}$, falls $R_0 \neq 0$ und $\tau = \emptyset$ sonst

oder π_κ ist vom Typ store j und ...

$\vdots$

oder π_κ ist vom Typ add $= c$ und

$$\kappa' = \kappa + 1; \quad R'_0 = R_0 + c; \quad T' = T; \quad R'_i = R_i \text{ für } i = 1, \ldots, k-1.$$

$\vdots$

oder π_κ ist vom Typ jzero s und

$$R_0 = 0; \ \kappa' = s; \ T' = T; \ R'_i = R_i \text{ für } i = 0, \ldots, k-1.$$

oder π_κ ist vom Typ jzero s und

$$R_0 \neq 0; \ \kappa' = \kappa + 1; \ T' = T; \ R'_i = R_i \text{ für } i = 0, \ldots, k-1.$$

oder π_κ ist vom Typ halt und

$$\kappa' = 0; \ T' = T; \ R'_i = R_i \text{ für } i = 0, \ldots, k-1.$$

Beispiel:

$$x := y + x - 1$$

(1) load 1
(2) add 2
(3) sub $= 1$
(4) store 1

$$\begin{array}{ccc} x & \leftrightarrow & R_1 \\ y & \leftrightarrow & R_2 \end{array}$$

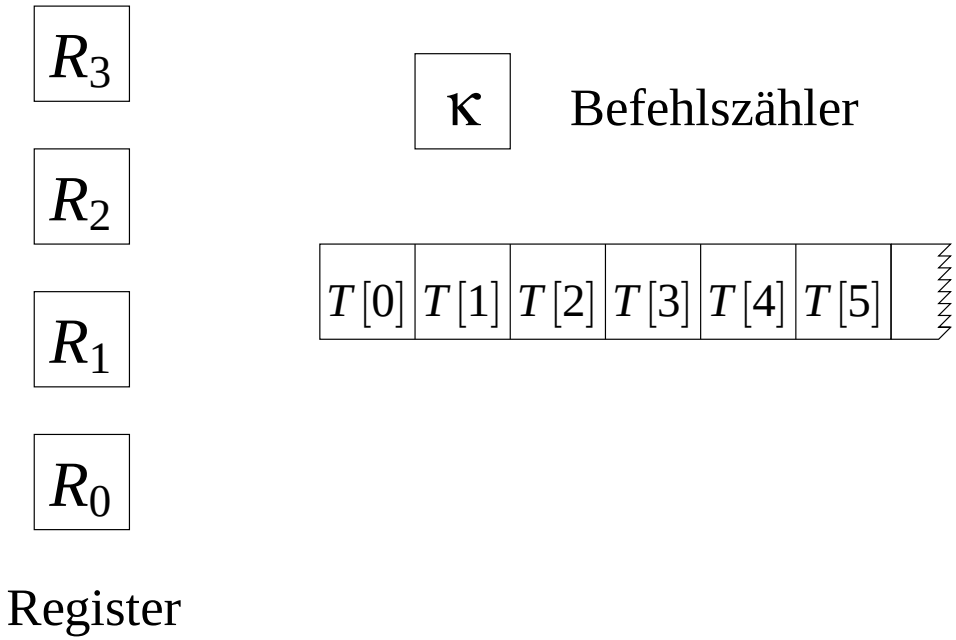
Beispiel:

while $x > 0$ do $x := x - 3$

(1) load 1
(2) jzero 6
(3) sub $= 3$
(4) store 1
(5) jump 1
(6) $\ldots$

$x \quad \leftrightarrow \quad R_1$

Registerturingmaschinen



Eine Registerturingmaschine ist ein Paar $M = (k, \Pi)$, wobei k die Anzahl der Register ist und $\Pi = (\pi_1, \pi_2, \ldots, \pi_p)$ eine endliche Folge von Befehlen (instructions).

Jeder Befehl π_i ist von einem der folgenden Typen:

Befehl	Operand	Semantik
read	j	$R_0 := T[R_j]$
write	j	$T[R_j] := R_0$
store	j	$R_j := R_0$
load	j	$R_0 := R_j$
load	$= c$	$R_0 := c$
add	j	$R_0 := R_0 + R_j$
add	$= c$	$R_0 := R_0 + c$
sub	j	$R_0 := \max(R_0 - R_j, 0)$
sub	$= c$	$R_0 := \max(R_0 - c, 0)$
half		$R_0 := \lfloor \frac{R_0}{2} \rfloor$
jump	s	$\kappa := s$
jpos	s	if $R_0 > 0$ then $\kappa := s$
jzero	s	if $R_0 = 0$ then $\kappa := s$
halt		$\kappa = 0$

Konfiguration einer Registerturingmaschine

$$(\kappa, R_0, R_1, \ldots, R_{k-1}, T)$$

κ	Befehlszähler; $0 \leq \kappa \leq p$
R_j	aktuelle Wert von Register R_j für $j = 0, \ldots, k-1$
T	Bandinhalt: Endliche Menge von Paaren aus $\mathbb{N} \times (\mathbb{N} - \{0\})$, so dass es für alle $i \geq 0$ höchstens ein Paar der Form (i, m) gibt

Eine Konfiguration heißt Haltekonfiguration, falls $\kappa = 0$.

Registermaschinen-Berechnungen

$$\Sigma, \quad \sqcup \in \Sigma, \rhd \notin \Sigma$$

$$\mathbf{E} : \Sigma \mapsto \{0, 1, \ldots, |\Sigma| - 1\} \text{ bijektiv}, \quad \mathbf{E}(\sqcup) = 0$$

Startkonfiguration einer Registerturingmaschine $M = (k, \Pi)$ bei Eingabe $w = a_1 a_2 \ldots a_n \in (\Sigma - \{\sqcup\})^*$:

$$(\kappa, R_0, R_1, \ldots, R_{k-1}, T)$$

mit $\kappa = 1$, $R_i = 0$ für $i = 0, \ldots, k-1$ und

$$T = \{(1, \mathbf{E}(a_1)), (2, \mathbf{E}(a_2)), \ldots, (n, \mathbf{E}(a_n))\}$$

M akzeptiert Eingabe w, falls M aus der Startkonfiguration in eine Haltekonfiguration mit $R_0 = 1$ übergeht. M verwirft Eingabe w, falls M aus der Startkonfiguration in eine Haltekonfiguration mit $R_0 = 0$ übergeht.

Sei $\Sigma_0 \subseteq \Sigma - \{\sqcup\}$ und sei $L \subseteq \Sigma_0^*$. M entscheidet (decides) L, falls für jedes Wort w aus Σ_0^* gilt: Falls $w \in L$, so akzeptiert M das Wort w, und falls $w \notin L$, so verwirft M das Wort w.

M akzeptiert (semi-entscheidet (semidecides)) $L \subseteq \Sigma_0^*$, falls M genau dann in eine Haltekonfiguration übergeht, wenn $w \in L$.

M berechnet eine Funktion

$$f : \Sigma_0^* \mapsto \Sigma_0^*$$

falls M für alle $x \in \Sigma_0^*$ bei Eingabe x in eine Haltekonfiguration mit Bandinhalt $T = \{(1, \mathbf{E}(b_1)), (2, \mathbf{E}(b_2)), \ldots, (m, \mathbf{E}(b_m))\}$ übergeht, wobei $f(x) = b_1 b_2 \ldots b_m$.

Beispiel: $\mathbf{E}(a) = 1;\ \mathbf{E}(b) = 2;\ \mathbf{E}(c) = 3;$

$acount := 0;\ bcount := 0;\ ccount := 0;$
$n := 1;$
while $T[n] = 1$ do
    $n := n + 1;\ acount := acount + 1;$
while $T[n] = 2$ do
    $n := n + 1;\ bcount := bcount + 1;$
while $T[n] = 3$ do
    $n := n + 1;\ ccount := ccount + 1;$
if $acount = bcount = ccount$ und $T[n] = 0$
    then load $= 1$; halt ;
    else load $= 0$; halt ;

Simulation von Registermaschinen

Satz: *Jede rekursive oder rekursiv aufzählbare Sprache und jede rekursive Funktion kann von einer Registerturingmaschine entschieden, akzeptiert beziehungsweise berechnet werden.*

Beweisidee: Position des Schreib-/Lesekopfes der Standardturingmaschine in einem Register verwalten. □

Satz: *Jede Sprache, die von einer Registerturingmaschine entschieden oder akzeptiert wird, und jede Funktion, die von einer Registerturingmaschine berechnet wird, kann auch von einer Standardturingmaschine entschieden, akzeptiert beziehungsweise berechnet werden.*

Beweisskizze:
Simuliere $M = (k, (\pi_1, \ldots, \pi_p))$ durch eine $(k+3)$-Band Turingmaschine M': Band 1 dient nur der Ein-/Ausgabe, Band 2 verwaltet Bandinhalt von M als Wort aus

$$(\sqcup^\star (0 \cup 1(0 \cup 1)^\star, 1(0 \cup 1)^\star))^\star \sqcup^\star \$$$

Bänder $3, \ldots, k+2$ verwalten Inhalte der Register R_0 bis R_{k-1}. Band $k+3$ dient der Aufnahme von Konstanten.

Befehlzähler wird in den Zuständen von M' verwaltet. Zustandsmenge K von M' ist partitioniert in Zustandsklassen $K_1, \ldots, K_p$. Zustandsklasse K_j dient der Simulation von Befehl π_j.

Simulation eines Lesezugriffs auf $T[i]$:
Durchsuchen von Band 2 nach Wort $(x, \ldots)$, wobei x die Binärdarstellung von i ist.

Simulation eines Schreibzugriffs $T[i] := n$:
Durchsuchen von Band 2 nach Paar $(x, \ldots)$ und Löschen desselbigen durch Überschreiben mit $\sqcup$, falls vorhanden, wobei x die Binärdarstellung von i ist; Suche nach \$ auf Band 2; Falls $n \neq 0$, Überschreiben von \$ und nachfolgender $\sqcup$ durch (x,y)\$, wobei y die Binärdarstellung von n ist.

$\ldots$ $\square$

Anzahl der Schritte einer simulierenden Standardturingmaschine ist polynomiell in der Länge der Eingabe und der Anzahl der Schritte der simulierten Registerturingmaschine!

Nichtdeterministische Turingmaschine

$$(K, \Sigma, \Delta, s, H)$$

$\vdots$ $\vdots$

Δ Übergangsrelation

$$\Delta : (K - H) \times \Sigma \times K \times (\Sigma \cup \{\leftarrow, \rightarrow\})$$

$\vdots$ $\vdots$

Relation $\vdash_M$ wird entsprechend erweitert, ebenso $\vdash_M^*$

$$M = (K, \Sigma, \Delta, s, H) \qquad w \in (\Sigma - \{\sqcup, \triangleright\})^*$$

M akzeptiert Eingabe w, falls $(s, \triangleright \underline{\sqcup} w) \vdash_M^* (h, u\underline{a}v)$ für ein $h \in H$, $a \in \Sigma$ und $u, v \in \Sigma^*$.

M akzeptiert eine Sprache $L \subseteq \Sigma_0^*$ mit $\Sigma_0 \subseteq \Sigma - \{\sqcup, \triangleright\}$, falls für alle $w \in \Sigma_0^*$ gilt: M akzeptiert w genau dann wenn $w \in L$.

$$M = (K, \Sigma, \Delta, s, H = \{y, n\}) \qquad w \in (\Sigma - \{\sqcup, \triangleright\})^*$$

M entscheidet eine Sprache $L \subseteq \Sigma_0^*$ mit $\Sigma_0 \subseteq \Sigma - \{\sqcup, \triangleright\}$, falls gilt

1. Es gibt ein $N \in \mathbb{N}$, das von M und w abhängt, so dass es keine Konfiguration C gibt mit

$$(s, \triangleright \underline{\sqcup} w) \vdash_M^N C$$

2. $w \in L$ genau dann wenn $(s, \triangleright \underline{\sqcup} w) \vdash_M^* (y, u \underline{a} v)$ für $a \in \Sigma$ und $u, v \in \Sigma^*$.

Sei $\Sigma_0 \subseteq \Sigma - \{\sqcup, \triangleright\}$. Eine nichtdeterministische Turingmaschine $M = (K, \Sigma, \Delta, s, \{h\})$ berechnet eine Funktion $f : \Sigma_0^* \mapsto \Sigma_0^*$, falls

1. Es gibt ein $N \in \mathbb{N}$, das von M und w abhängt, so dass es keine Konfiguration C gibt mit

$$(s, \triangleright \underline{\sqcup} w) \vdash_M^N C$$

2. $(s, \triangleright \underline{\sqcup} w) \vdash_M^* (h, u\underline{a}v)$ genau dann wenn $ua = \triangleright\sqcup$ und $v = f(w)$.

Satz: Falls eine nichtdeterministische Turingmaschine M eine Sprache akzeptiert oder entscheidet oder eine Funktion berechnet, so gibt es auch eine deterministische Turingmaschine M', die die Sprache akzeptiert beziehungsweise entscheidet beziehungsweise die Funktion berechnet.

Beweisidee:

Alle möglichen Berechnungen durchprobieren.

Ein paar Details: Es gibt $r \in \mathbb{N}$, so dass für alle Paare (q,a) gilt

$$|\{(p,b) : (q,a,p,b) \in \Delta\}| \leq r$$

Für jedes Paar (q,a) 4-Tupel (q,a,p_i,b_i) für $i = 1,2,\ldots,r$.
Sei Σ_1 Menge von r Symbolen. Wort aus Σ_1^d kodiert eine Berechnung bestehend aus d Schritten.

3-Band Turingmaschine M:

Band 1: Eingabe w
Band 2: bearbeitete Kopie von w
Band 3: $i_1 i_2 \ldots i_d \in \Sigma_1^*$.

Das Symbol auf der j-ten Position von Band 3 bestimmt, welche der r möglichen Aktionen auf Band 2 ausgeführt wird. Wird auf Band 3 ein $\sqcup$ erreicht, so wird die aktuelle deterministische Simulation gestoppt, ein neues Wort auf Band 3 generiert und die zum neu erzeugten Wort gehörige Simulation wird gestartet. Auf Band 3 werden der Länge nach alle Wörter aus Σ_1^* generiert.

□

Grammatiken

Grammatik (V, Σ, R, S)

V	Alphabet
Σ	$\Sigma \subseteq V$; Terminalalphabet
R	$R \subseteq (V^+ - \Sigma^+) \times V^*$; endliche Menge von Regeln oder auch Produktionen
S	$S \in V - \Sigma$; Startsymbol
$V - \Sigma$	Nichtterminal(symbol)e oder Variablen

Sei $G = (V, \Sigma, R, S)$ eine Grammatik.

Falls $(u, v) \in R$, so schreiben wir $u \longrightarrow_G v$.

Relation $\Rightarrow_G$ auf $V^* \times V^*$:
$x \Rightarrow_G y$ genau dann wenn es $u, v, w, z \in V^*$ gibt, so dass
$x = uvw,\ \ y = uzw$ und $v \longrightarrow_G z$

Ableitung von y in G aus x:
$x = w_0 \Rightarrow_G w_1 \Rightarrow_G \cdots \Rightarrow_G w_n = y$

Erzeugte Sprache

$$L(G) = \{w \in \Sigma^* : S \Rightarrow_G^* w\}$$

Beispiel:

$$
\begin{aligned}
V &= \{S, a, b, c, A, B, C, T_a, T_b, T_c\} \\
\Sigma &= \{a, b, c\} \\
R &= \{S \to ABCS,\ \ S \to T_c, \\
&\quad CA \to AC,\ \ BA \to AB,\ \ CB \to BC, \\
&\quad CT_c \to T_c c,\ \ CT_c \to T_b c, \\
&\quad BT_b \to T_b b,\ \ BT_b \to T_a b, \\
&\quad AT_a \to T_a a,\ \ T_a \to \varepsilon\}
\end{aligned}
$$

$$L(G) = \{a^n b^n c^n : n \geq 1\}$$

Grammatiken und Turingmaschinen

Satz: *Eine Sprache ist eine von einer Grammatik erzeugte Sprache genau dann wenn sie rekursiv aufzählbar ist.*

Beweisskizze:

$\Rightarrow$: Sei $G = (V, \Sigma, R, S)$ eine Grammatik.

Nichtdeterministische 2-Band Turingmaschine M simuliert Ableitungen:

Band 1: w

Band 2: $x \in V^*$ mit $S \Rightarrow_G^* x$
(anfangs $x = S$)

$|R|+1$ Optionen, von denen eine nichtdeterministisch ausgewählt wird: Eine von $|R|$ Regeln auf Band 2 anwenden oder Ableiten beenden.

Regel $u \to v$ anwenden: Startposition für Suche nach u auf Band 2 nichtdeterministisch auswählen, Bandinhalt ab Startposition mit u vergleichen. Falls Übereinstimmung gefunden wurde, zugehörigen Teil von Band 2 löschen, Platz für v anpassen und v auf Band 2 schreiben.

Ableiten beenden: Bänder 1 und 2 vergleichen.

Falls in einem der Schritte keine Übereinstimmung gefunden wurde, in eine Endlosschleife übergehen.

M hält bei Eingabe w genau dann wenn $w \in L(G)$.

$\Leftarrow$: Sei $(K, \Sigma, \delta, s, \{h\})$ eine Turingmaschine.

Annahmen o.B.d.A.: Falls M hält, dann in der Konfiguration $(h, \underline{\sqcup})$; Ferner $K \cap \Sigma = \emptyset$ und $\triangleleft \notin \Sigma \cup K$.

Wir konstruieren Grammatik $G = (V, \Sigma, R, S)$ mit

$$V = \Sigma \cup K \cup \{S, \triangleleft\}$$

Konfiguration wird durch Wort aus V^* dargestellt:

$$(q, \triangleright u\underline{a}w) \qquad \longleftrightarrow \qquad \triangleright uaqw\triangleleft$$

Übungen vom 1. Mai eine Woche später

Grammatiken

Grammatik (V, Σ, R, S)

V	Alphabet
Σ	$\Sigma \subseteq V$; Terminalalphabet
R	$R \subseteq (V^+ - \Sigma^+) \times V^*$; endliche Menge von Regeln oder auch Produktionen
S	$S \in V - \Sigma$; Startsymbol
$V - \Sigma$	Nichtterminal(symbol)e oder Variablen

Grammatiken und Turingmaschinen

Satz: Eine Sprache ist eine von einer Grammatik erzeugte Sprache genau dann wenn sie rekursiv aufzählbar ist.

Beweisskizze:

$\Rightarrow$: Sei $G = (V, \Sigma, R, S)$ eine Grammatik.

Band 1: w

Band 2: $x \in V^*$ mit $S \Rightarrow_G^* x$ (anfangs $x = S$)

$|R| + 1$ Optionen, von denen eine nichtdeterministisch ausgewählt wird: Eine von $|R|$ Regeln auf Band 2 anwenden oder Ableiten beenden.

M hält bei Eingabe w genau dann wenn $w \in L(G)$.

$\Leftarrow$: Sei $(K, \Sigma, \delta, s, \{h\})$ eine Turingmaschine.

Annahmen o.B.d.A.: Falls M hält, dann in der Konfiguration $(h, \underline{\sqcup})$; Ferner $K \cap \Sigma = \emptyset$ und $\triangleleft \notin \Sigma \cup K$.

Wir konstruieren Grammatik $G = (V, \Sigma, R, S)$ mit

$$V = \Sigma \cup K \cup \{S, \triangleleft\}$$

Konfiguration wird als Wort aus V^* dargestellt:

$$(q, \triangleright u\underline{a}w) \qquad \longleftrightarrow \qquad \triangleright uaqw\triangleleft$$

Regeln in R entsprechen rückwärts Übergängen von M:

M	G
$\delta(q,a) = (p,b)$	$bp \to_G aq$
$\delta(q,a) = (p,\to)$	$abp \to_G aqb,\ b \neq \triangleleft$ $a \sqcup p\triangleleft \to_G aq\triangleleft$
$\delta(q,a \neq \sqcup) = (p,\leftarrow)$	$pa \to_G aq$
$\delta(q,\sqcup) = (p,\leftarrow)$	$p \sqcup b \to_G \sqcup qb,\ b \neq \triangleleft$ $p\triangleleft \to_G \sqcup q\triangleleft$
	$S \to_G \triangleright \sqcup h\triangleleft$
	$\triangleleft \to_G \varepsilon,\ \triangleright \sqcup s \to_G \varepsilon$

Lemma: $(q_1, u_1\underline{a_1}w_1) \vdash_M (q_2, u_2\underline{a_2}w_2)$

g.d.w.

$u_2 a_2 q_2 w_2 \Rightarrow_G u_1 a_1 q_1 w_1$

□

Nun gilt

$w \in L(G)$

g.d.w. $S \Rightarrow_G \rhd \sqcup h \lhd \Rightarrow_G^* \rhd \sqcup sw \lhd \Rightarrow_G w \lhd \Rightarrow_G w$

g.d.w. $\rhd \sqcup h \lhd \Rightarrow_G^* \rhd \sqcup sw \lhd$

g.d.w. $(s, \rhd\underline{\sqcup}w) \vdash_M^* (h, \rhd\underline{\sqcup})$

g.d.w. M hält bei Eingabe w

□

Sei $G = (V, \Sigma, R, S)$ eine Grammatik und sei $f : \Sigma^* \mapsto \Sigma^*$ eine Funktion. G berechnet f, falls für alle $v, w \in \Sigma^*$ gilt:

$$SwS \Rightarrow_G^* v \quad \text{g.d.w.} \quad v = f(w)$$

Satz: Eine Funktion ist durch eine Grammatik berechnenbar genau dann wenn sie durch eine Turingmaschine berechenbar ist.

(Beweis Übungsaufgabe) □

Primitiv rekursive und μ-rekursive Funktionen

$$f(n,m) = m \cdot n^2 + 6 \cdot m^{3n+15}$$

Primitiv rekursive Funktionen

Basisfunktionen

1. k-stellige Nullfunktion:

$$\mathsf{zero}(n_1, \ldots, n_k) = 0 \text{ für alle } n_1, \ldots, n_k \in \mathbb{N}$$

2. j-te k-stellige Identität:

$$\mathsf{id}_{k,j}(n_1, \ldots, n_k) = n_j \text{ für alle } n_1, \ldots, n_k \in \mathbb{N}$$

3. Nachfolgerfunktion:

$$\mathsf{succ}(n) = n + 1 \text{ für alle } n \in \mathbb{N}$$

Verknüpfung von Funktionen

1. Komposition: Seien g eine k-stellige Funktion, $h_1, \ldots, h_k$ jeweils l-stellige Funktionen.

$$f(n_1, \ldots, n_l) = g(h_1(n_1, \ldots, n_l), \ldots, h_k(n_1, \ldots, n_l))$$

2. Primitive Rekursion: Seien g eine k-stellige Funktion und h eine $(k+2)$-stellige Funktion.

$$\begin{aligned} f(n_1, \ldots, n_k, 0) &= g(n_1, \ldots, n_k) \\ f(n_1, \ldots, n_k, m+1) &= h(n_1, \ldots, n_k, m, f(n_1, \ldots, n_k, m)) \end{aligned}$$

Eine Funktion heißt primitiv rekursiv, falls sie zu den Basisfunktionen gehört oder aus diesen durch die Operationen Komposition und primitive Rekursion erzeugt werden kann.

Beispiele:

$$\begin{aligned} \mathsf{plus}(n,0) &= \mathsf{id}_{1,1}(n) \\ \mathsf{plus}(n,m+1) &= \mathsf{succ}(\mathsf{id}_{3,3}(n,m,\mathsf{plus}(n,m))) \end{aligned}$$

Vereinfacht:

$$\begin{aligned} \mathsf{plus}(n,0) &= n \\ \mathsf{plus}(n,m+1) &= \mathsf{succ}(\mathsf{plus}(n,m)) \end{aligned}$$

$$\begin{aligned} \mathsf{mult}(n,0) &= \mathsf{zero}(n) \\ \mathsf{mult}(n,m+1) &= \mathsf{plus}(n,\mathsf{mult}(n,m)) \end{aligned}$$

$$\begin{aligned} \exp(n,0) &= \mathsf{succ}(\mathsf{zero}(n)) \\ \exp(n,m+1) &= \mathsf{mult}(n,\exp(n,m)) \end{aligned}$$

$$c(n_1,\ldots,n_k) = 516$$

$$\begin{aligned} \mathsf{pred}(0) &= 0 \\ \mathsf{pred}(m+1) &= m \end{aligned}$$

$$\begin{aligned} \mathsf{minus}(n,0) &= n \\ \mathsf{minus}(n,m+1) &= \mathsf{pred}(\mathsf{minus}(n,m)) \end{aligned}$$

Prädikate

Eine Funktion, die nur die Werte 0 und 1 annimmt, heißt Prädikat.

$$\begin{aligned} \mathsf{iszero}(0) &= 1 \\ \mathsf{iszero}(m+1) &= 0 \end{aligned}$$

$$\begin{aligned} \mathsf{positive}(0) &= 0 \\ \mathsf{positive}(m+1) &= 1 \end{aligned}$$

$$\mathsf{greater\text{–}than\text{–}or\text{–}equal}(n,m) = \mathsf{iszero}(\mathsf{minus}(m,n))$$

Primitiv rekursive und μ-rekursive Funktionen

$$f(n,m) = m \cdot n^2 + 6 \cdot m^{3n+15}$$

Prädikate

Eine Funktion, die nur die Werte 0 und 1 annimmt, heißt Prädikat.

$$\begin{aligned} \mathsf{iszero}(0) &= 1 \\ \mathsf{iszero}(m+1) &= 0 \end{aligned}$$

$$\begin{aligned} \mathsf{positive}(0) &= 0 \\ \mathsf{positive}(m+1) &= 1 \end{aligned}$$

$$\mathsf{greater\text{–}than\text{–}or\text{–}equal}(n,m) = \mathsf{iszero}(\mathsf{minus}(m,n))$$

Operationen auf Prädikaten

Negation:

$$\mathsf{minus}(1,\ p(n_1,\ldots,n_k))$$

Disjunktion:

$$\mathsf{positive}(p(n_1,\ldots,n_k)+q(n_1,\ldots,n_l))$$

Konjunktion:

$$p(n_1,\ldots,n_k)\cdot q(n_1,\ldots,n_l)$$

Beispiel:

$\mathsf{equal}(m,n) = \mathsf{greater\text{–}than\text{–}or\text{–}equal}(n,m)$ und
$\mathsf{greater\text{–}than\text{–}or\text{–}equal}(m,n)$

Funktionsdefinition durch endliche Fallunterscheidung

$$f(n_1,\ldots,n_k) = \begin{cases} g(n_1,\ldots,n_k) & \text{falls } p(n_1,\ldots,n_k) \\ h(n_1,\ldots,n_k) & \text{sonst} \end{cases}$$

$$\begin{aligned} f(n_1,\ldots,n_k) \quad = \quad & p(n_1,\ldots,n_k)\cdot g(n_1,\ldots,n_k) \\ & +(\mathsf{minus}(1,\ p(n_1,\ldots,n_k))\cdot h(n_1,\ldots,n_k)) \end{aligned}$$

Beispiel: Division (div) mit Rest (rem)

$$\begin{aligned}
\mathsf{rem}(0,n) &= 0 \\
\mathsf{rem}(m+1,n) &= \begin{cases} 0 & \text{f. equal}(\mathsf{rem}(m,n),\mathsf{pred}(n)) \\ \mathsf{rem}(m,n)+1 & \text{sonst} \end{cases}
\end{aligned}$$

$$\begin{aligned}
\mathsf{div}(0,n) &= 0 \\
\mathsf{div}(m+1,n) &= \begin{cases} \mathsf{div}(m,n)+1 & \text{f. equal}(\mathsf{rem}(m,n),\mathsf{pred}(n)) \\ \mathsf{div}(m,n) & \text{sonst} \end{cases}
\end{aligned}$$

m-t niederwertigste Ziffer der Darstellung von n zur Basis p:

$$\mathsf{digit}(m, n, p) = \mathsf{div}(\mathsf{rem}(n, p^m), p^{\mathsf{minus}(m,1)})$$

$$\mathsf{odd}(n) = \mathsf{digit}(1, n, 2)$$

Beschränkte Quantifizierung

Endliche Konjunktion und Disjunktion von Prädikaten:

$$p(n_1,\ldots,n_l,0) \text{ und } p(n_1,\ldots,n_l,1) \text{ und } \cdots \text{ und } p(n_1,\ldots,n_l,k)$$
$$p(n_1,\ldots,n_l,0) \text{ oder } p(n_1,\ldots,n_l,1) \text{ oder } \cdots \text{ oder } p(n_1,\ldots,n_l,k)$$

$$\begin{aligned} q(n,k) &= p(n,0) \text{ und } p(n,1) \text{ und } \cdots \text{ und } p(n,k) \\ q(n,k+1) &= q(n,k) \text{ und } p(n,k+1) \\ &= h(n,k,q(n,k)) \\ h(x,y,z) &= p(x,\mathsf{succ}(y)) \text{ und } z \end{aligned}$$

Beschränkte Existenzquantifizierung:

$$q(n_1,\ldots,n_l,n) = \begin{cases} 1 & \exists k \leq n : p(n_1,\ldots,n_l,k) = 1 \\ 0 & \text{sonst} \end{cases}$$

Beschränkte Allquantifizierung:

$$q(n_1,\ldots,n_l,n) = \begin{cases} 1 & \forall 0 \leq j \leq n : p(n_1,\ldots,n_l,j) = 1 \\ 0 & \text{sonst} \end{cases}$$

Beschränkte Minimalisierung

$$f(n_1,\ldots,n_l,b) = \mu\,(m \leq b)[g(n_1,\ldots,n_l,m) = 1]$$

$$= \begin{cases} \text{das kleinste } m \leq b, \text{ so dass } g(n_1,\ldots,n_l,m) = 1 \\ \qquad \text{falls ein solches } m \leq b \text{ existiert} \\ 0 \quad \text{sonst} \end{cases}$$

$$f(n_1,\ldots,n_l,0) \quad = \quad 0$$

$$f(n_1,\ldots,n_l,b+1)$$

$$= \begin{cases} b+1 & \text{falls } g(n_1,\ldots,n_l,0) \neq 1 \\ & \text{und } f(n_1,\ldots,n_l,b) = 0 \\ & \text{und } g(n_1,\ldots,n_l,b+1) = 1 \\ f(n_1,\ldots,n_l,b) & \text{sonst} \end{cases}$$

Nicht primitiv rekursive Funktionen

Es gibt abzählbar unendlich viele primitiv rekursive Funktionen

$$f_0, f_1, f_2, f_3, \ldots$$

Es gibt berechenbare, nicht primitiv rekursive Funktionen:

$$g(n) = f_n(n) + 1$$

Ackermann-Funktion

$$
\begin{aligned}
A(0,y) &= y+1 \\
A(x,0) &= A(x-1,1) \qquad x>0 \\
A(x,y) &= A(x-1,A(x,y-1)) \qquad x,y>0
\end{aligned}
$$

Satz: Die Ackermann-Funktion ist nicht primitiv rekursiv.

(ohne Beweis) □

$$
\begin{aligned}
A(0,y) &= y+1 \\
A(1,y) &= y+2 \\
A(2,y) &= 2y+3 \\
A(3,y) &= 2^{y+3}-3 \\
A(4,y) &= \underbrace{2^{2^{\cdot^{\cdot^{\cdot^{2}}}}}}_{y+3}-3
\end{aligned}
$$

μ-rekursive Funktionen

Unbeschränkte Minimalisierung

Sei g eine $(k+1)$-stellige Funktion. Wir definieren eine k-stellige Funktion durch

$$\mu\, m[g(n_1,\ldots,n_k,m)=1]$$

$$= \begin{cases} \text{das kleinste } m\text{, so dass } g(n_1,\ldots,n_k,m)=1 \\ \qquad \text{falls ein solches } m \text{ existiert} \\ 0 \quad \text{sonst} \end{cases}$$

Eine Funktion $g : \mathbb{N}^k \mapsto \mathbb{N}$ heißt minimalisierbar, falls das folgende Verfahren für alle $n_1, \ldots, n_k \in \mathbb{N}$ terminiert:

$m := 0;$
while $g(n_1, \ldots, n_k, m) \neq 1$ do $m := m + 1;$
return $m;$

Eine Funktion heißt μ-rekursiv, falls sie zu den Basisfunktionen gehört oder aus diesen durch Komposition und primitive Rekursion und Anwendung der unbeschränkten Minimalisierung auf minimalisierbare Funktionen erzeugt werden kann.

Satz: Eine Funktion $f : \mathbb{N}^k \mapsto \mathbb{N}$ ist μ-rekursiv genau dann wenn sie durch eine Turingmaschine berechenbar ist.

Beweisskizze:

$\Rightarrow$: Jede μ-rekursive Funktion $f : \mathbb{N}^k \mapsto \mathbb{N}$ kann auch durch eine Turingmaschine berechnet werden.

Es gibt Registerturingmaschinen, die die Basisfunktionen berechnen.

Wir geben Registerturingmaschinen an, die Komposition, primitive Rekursion und Anwendung des μ-Operators auf minimalisierbare Funktionen realisieren.

Komposition:

Wenn es Registerturingmaschinen gibt, die $h_1, h_2, \ldots, h_l$ und g berechnen, dann berechnet die zu folgendem Programm gehörige Registerturingmaschine die Komposition dieser Funktionen:

$$
\begin{aligned}
m_1 &:= h_1(n_1, \ldots, n_k);\\
m_2 &:= h_2(n_1, \ldots, n_k);\\
&\vdots\\
m_l &:= h_l(n_1, \ldots, n_k);\\
R_0 &:= g(m_1, \ldots, m_l);
\end{aligned}
$$

Primitive Rekursion:

$v := g(n_1, \ldots, n_k);$
if $(m = 0)$ then $R_0 = v$
else for $i := 1, 2, \ldots, m$ do
$\quad v := h(n_1, \ldots, n_k, i-1, v);$
$R_0 := v;$

Unbeschränkte Minimalisierung:

$m := 0;$
while $g(n_1, \ldots, n_k, m) \neq 1$ do
$\quad m := m + 1;$
$R_0 := m;$

Satz: Eine Funktion $f : \mathbb{N}^k \mapsto \mathbb{N}$ ist μ-rekursiv genau dann wenn sie durch eine Turingmaschine berechenbar ist.

Beweisskizze:

$\Rightarrow$: Jede μ-rekursive Funktion $f : \mathbb{N}^k \mapsto \mathbb{N}$ kann auch durch eine Turingmaschine berechnet werden.

Es gibt Registerturingmaschinen, die die Basisfunktionen berechnen.

Wir geben Registerturingmaschinen an, die Komposition, primitive Rekursion und Anwendung des μ-Operators auf minimalisierbare Funktionen realisieren.

Komposition:

Wenn es Registerturingmaschinen gibt, die $h_1, h_2, \ldots, h_l$ und g berechnen, dann berechnet die zu folgendem Programm gehörige Registerturingmaschine die Komposition dieser Funktionen:

$$
\begin{aligned}
m_1 &:= h_1(n_1, \ldots, n_k);\\
m_2 &:= h_2(n_1, \ldots, n_k);\\
&\vdots\\
m_l &:= h_l(n_1, \ldots, n_k);\\
R_0 &:= g(m_1, \ldots, m_l);
\end{aligned}
$$

Primitive Rekursion:

$v := g(n_1, \ldots, n_k);$
if $(m = 0)$ then $R_0 = v$
else for $i := 1, 2, \ldots, m$ do
$\quad v := h(n_1, \ldots, n_k, i-1, v);$
$R_0 := v;$

Unbeschränkte Minimalisierung:

$m := 0;$
while $g(n_1, \ldots, n_k, m) \neq 1$ do
$\quad m := m + 1;$
$R_0 := m;$

$\Leftarrow$: Jede turing-berechenbare Funktion $f : \mathbb{N}^k \mapsto \mathbb{N}$ ist μ-rekursiv.

Sei $M = (K, \Sigma, \delta, s, \{h\})$ eine Turingmaschine, die f berechnet, $\Sigma \cap K = \emptyset$. Sei $\Gamma = \Sigma \cup K$ und sei $b = |\Gamma|$.

$$\mathbf{E} : \Gamma \mapsto \{0, 1, \ldots, b-1\}$$
$$\mathbf{E}(0) = 0, \quad \mathbf{E}(1) = 1$$

Darstellung einer Konfiguration als Wort über Γ:

$$(q, a_1 a_2 \ldots \underline{a_k} \ldots a_n) \quad \leftrightarrow \quad a_1 a_2 \ldots a_{k-1} a_k q a_{k+1} \ldots a_n$$

Interpretation von Wörtern aus $(\Gamma - \{0\})\Gamma^* \cup \{0\}$ als Zahlen zur Basis b:

$$u_1 u_2 \ldots u_m \quad \leftrightarrow \quad \mathbf{E}(u_1) b^{m-1} + \mathbf{E}(u_2) b^{m-2} + \cdots + \mathbf{E}(u_m)$$

Wir bilden so Konfigurationen injektiv auf natürliche Zahlen ab:

$$(q, a_1 a_2 \ldots \underline{a_k} \ldots a_n)$$

$$\leftrightarrow$$

$$\mathbf{E}(a_1) b^n + \cdots + \mathbf{E}(a_k) b^{n-k+1} + \mathbf{E}(q) b^{n-k} + \mathbf{E}(a_{k+1}) b^{n-k-1} + \cdots + \mathbf{E}(a_n)$$

$$f : \mathbb{N} \mapsto \mathbb{N}$$

$$f'(n) = \mathsf{num}(\mathsf{output}(\mathsf{last}(\mathsf{comp}(n))))$$

comp berechnet für n diejenige Zahl, die zur Basis b geschrieben der Hintereinanderreihung der Konfigurationen der Berechnung von M bei Eingabe n entspricht, wobei die Hintereinanderreihung von Konfigurationen mit $\triangleright \sqcup sw$ startet, wobei w die Binärdarstellung von n ist, und mit $\triangleright \sqcup hw'$ ended, wobei w' die Binärdarstellung von $f(n)$ ist.

last — berechnet für eine Zahl, die zur Basis b geschrieben einer Hintereinanderreihung von Konfigurationen entspricht, die Zahl, die zur Basis b geschrieben der letzten Konfiguration aus der Hintereinanderreihung entspricht, und 0 sonst.

output — berechnet für eine Zahl, die zur Basis b geschrieben einer Konfiguration der Form $\triangleright \sqcup hv$ entspricht, die Zahl, die zur Basis b geschrieben dem Wort v entspricht, und 0 sonst.

num — berechnet für eine Zahl, die zur Basis b geschrieben ein Wort aus $1\{0,1\}^* \cup \{0\}$ ist, diejenige Zahl, deren Binärdarstellung dieses Wort ist, und 0 sonst.

$$\mathsf{lastpos}(n) = \mu\,(m \leq n)[\mathsf{equal}(\mathsf{digit}(m,n,b),\mathbf{E}(\triangleright)) \text{ oder } \mathsf{equal}(m,n)]$$

$$\mathsf{last}(n) = \mathsf{rem}(n, b^{\mathsf{lastpos}(n)})$$

$$\mathsf{comp}(n) = \mu\, m[\mathsf{iscomp}(m,n) \text{ und } \mathsf{halted}(\mathsf{last}(n))]$$

iscomp Prädikat ist erfüllt, genau dann wenn m zur Basis b geschrieben eine Hintereinanderreihung von Konfigurationen ist, die (dem Anfang) einer Berechnung von M bei Eingabe n entspricht.

$$\mathsf{iscomp}(m,n) = \mathsf{equal}(n, \mathsf{num}(\mathsf{output}(\mathsf{kth}(m,1)))) \text{ und } \mathsf{yields}(m,n)$$

$$\mathsf{yields}(m,n) = \begin{cases} 1 & \forall 0 \leq k < \mathsf{length}(m): \\ & \mathsf{onestep}(\mathsf{kth}(m,k), \mathsf{kth}(m,k+1)) \\ 0 & \text{sonst} \end{cases}$$

kth berechnet für ein erstes Argument, das zur Basis b geschrieben einer Hintereinanderreihung von Konfigurationen entspricht, die Zahl, die zur Basis b geschrieben der k-ten Konfiguration in dieser Hintereinanderreihung entspricht, wobei k das zweite Argument ist, falls es so viele Konfiguration in der Hintereinanderreihung gibt, und 0 sonst.

length — berechnet für eine Zahl, die zur Basis b geschrieben einer Hintereinanderreihung von Konfigurationen entspricht, die Anzahl der Konfigurationen, aus denen die Hintereinanderreihung besteht, und 0 sonst.

onestep — Prädikat ist erfüllt, genau dann wenn beide Argumente zur Basis b geschrieben Konfigurationen entsprechen und die zweite Konfiguration aus der ersten durch einen Übergang von M hervorgeht.

halted — Prädikat ist erfüllt, genau dann wenn die Zahl zur Basis b geschrieben einer Haltekonfiguration entspricht.

$$f' \equiv f$$

□

LOOP-, WHILE- und GOTO-Berechenbarkeit

Variablen:	$x_0, x_1, x_2, \ldots$
Konstanten:	$0, 1, 2, \ldots$
Trennsymbole:	$;\ :=$
Operationszeichen:	$+, -$
Schlüsselwörter:	LOOP, WHILE, GOTO, DO, END

LOOP-Berechenbarkeit

$$x_i := x_j + c \quad \text{und} \quad x_i := x_j - c$$

sind LOOP-Programme.
Falls Π_1 und Π_2 LOOP-Programme sind, dann ist auch

$$\Pi_1; \Pi_2$$

ein LOOP-Programm.
Falls Π ein LOOP-Programm ist und x_i eine Variable, dann ist auch

$$\text{LOOP } x_i \text{ DO } \Pi \text{ END}$$

ein LOOP-Programm.

Eine Funktion $f : \mathbb{N}^k \mapsto \mathbb{N}$ heißt LOOP-berechenbar, falls es ein LOOP-Programm Π gibt, das gestartet mit $n_1, \ldots, n_k$ in den Variablen $x_1, \ldots, x_k$ (und 0 in den restlichen Variablen) mit dem Wert $f(n_1, \ldots, n_k)$ in der Variablen x_0 hält.

Beispiel:

Addition: $x_1 + x_2$

$x_0 := x_1;$
LOOP x_2 DO $x_0 := x_0 + 1$ END

If-then-else:

$$\text{IF } x = 0 \text{ THEN } A \text{ END}$$

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

Satz: Eine Funktion ist LOOP*-berechenbar genau dann wenn sie primitiv rekursiv ist.*

o.B. □

Beispiel: succ

$$M = (K, \Sigma, \delta, s, \{h\})$$

$$\Sigma = \{0, 1, \sqcup, \triangleright\}$$

$$K = \{s, R, L, a, c, d, e, f, g, h\}$$

$$\Gamma = K \cup \Sigma \qquad |\Gamma| = b = 14$$

		δ
s	$\sqcup$	$(R, \rightarrow)$
s	0	$(s, 0)$
s	1	$(s, 1)$
s	$\triangleright$	$(s, \rightarrow)$
R	$\sqcup$	$(a, \leftarrow)$
R	0	$(R, \rightarrow)$
R	1	$(R, \rightarrow)$
R	$\triangleright$	$(R, \rightarrow)$
a	$\sqcup$	$(e, \leftarrow)$
a	0	$(L, 1)$
a	1	$(c, 0)$
a	$\triangleright$	$(a, \rightarrow)$

		δ
c	$\sqcup$	$(c, \sqcup)$
c	0	$(a, \leftarrow)$
c	1	$(c, 1)$
c	$\triangleright$	$(c, \rightarrow)$
e	$\sqcup$	$(L, 1)$
e	0	$(f, 1)$
e	1	$(g, 1)$
e	$\triangleright$	$(e, \rightarrow)$
d	$\sqcup$	$(L, 0)$
d	0	$(f, 0)$
d	1	$(g, 0)$
d	$\triangleright$	$(d, \rightarrow)$

		δ
f	$\sqcup$	$(f, \sqcup)$
f	0	$(d, \rightarrow)$
f	1	$(d, \rightarrow)$
f	$\triangleright$	$(f, \rightarrow)$
g	$\sqcup$	$(g, \sqcup)$
g	0	$(e, \rightarrow)$
g	1	$(e, \rightarrow)$
g	$\triangleright$	$(g, \rightarrow)$
L	$\sqcup$	$(h, \sqcup)$
L	0	$(L, \leftarrow)$
L	1	$(L, \leftarrow)$
L	$\triangleright$	$(L, \rightarrow)$

	0	1	$\sqcup$	$\triangleright$	s	R	L	a	c	d	e	f	g	h
E	0	1	2	3	4	5	6	7	8	9	10	11	12	13

$(s, \triangleright\underline{\sqcup}101)$ $\triangleright \sqcup s101$

$3 \cdot 14^5 + 2 \cdot 14^4 + 4 \cdot 14^3 + 1 \cdot 14^2 + 0 \cdot 14 + 1 = 1701477$

$(s, \triangleright \underline{\sqcup} 101)$

$\triangleright \sqcup s101$

1701477

$\vdash_M$ $(R, \triangleright \sqcup \underline{1}01)$

$\triangleright \sqcup s101 \triangleright \sqcup 1R01$

12811334018701

$\vdash_M$ $(R, \triangleright \sqcup 1\underline{0}1)$

$\triangleright \sqcup s101 \triangleright \sqcup 1R01 \triangleright \sqcup 10R1$

96463400701835545855

$\vdash_M$ $(R, \triangleright \sqcup 10\underline{1})$

$\triangleright \sqcup s101 \triangleright \sqcup 1R01 \triangleright \sqcup 10R1 \triangleright \sqcup 101R$

726324648266896008596566347

$\vdash_M$ $(R, \triangleright \sqcup 101\underline{\sqcup})$

$\triangleright \sqcup s101 \triangleright \sqcup 1R01 \triangleright \sqcup 10R1 \triangleright \sqcup 101R \triangleright \sqcup 101 \sqcup R$

765644262153810354697781810294527 89

$\vdash_M$ $(a, \triangleright \sqcup 101\underline{1})$

$\triangleright \sqcup s101 \triangleright \sqcup 1R01 \triangleright \sqcup 10R1 \triangleright \sqcup 101R \triangleright \sqcup 101 \sqcup R \triangleright \sqcup 101a$

576494603508055260286971726075781836768973

$\vdash_M$ $(c, \triangleright \sqcup 100\underline{0})$

$\triangleright \sqcup s101 \triangleright \sqcup 1R01 \triangleright \sqcup 10R1 \triangleright \sqcup 101R \triangleright \sqcup 101 \sqcup R \triangleright \sqcup 101a$

$\triangleright \sqcup 100c$

607703161928747972124817351945763329533735061141 76

$\vdash_M$ $(a, \triangleright \sqcup 10\underline{0}0)$

$\triangleright \sqcup s101 \triangleright \sqcup 1R01 \triangleright \sqcup 10R1 \triangleright \sqcup 101R \triangleright \sqcup 101 \sqcup R \triangleright \sqcup 101a$

$\triangleright \sqcup 100c \triangleright \sqcup 10a0$

326837345361166949360057767492878216943151525523 52286170

$\vdash_M$ $(L, \triangleright \sqcup 1\underline{1}0)$

$\triangleright \sqcup s101 \triangleright \sqcup 1R01 \triangleright \sqcup 10R1 \triangleright \sqcup 101R \triangleright \sqcup 101 \sqcup R \triangleright \sqcup 101a$

$\triangleright \sqcup 100c \triangleright \sqcup 10a0 \triangleright \sqcup 11L0$

246093355804133954721673192241725627808926936488 428423401010448

$\vdash_M \quad (L, \triangleright \sqcup \underline{1}10)$

$\triangleright \sqcup s101 \triangleright \sqcup 1R01 \triangleright \sqcup 10R1 \triangleright \sqcup 101R \triangleright \sqcup 101 \sqcup R \triangleright \sqcup 101a$
$\triangleright \sqcup 100c \triangleright \sqcup 10a0 \triangleright \sqcup 11L0 \triangleright \sqcup 1L10$
$185296878188803556089920828121899381670991648965$
9335397421150606286366

$\vdash_M \quad (L, \triangleright \underline{\sqcup} 110)$

$\triangleright \sqcup s101 \triangleright \sqcup 1R01 \triangleright \sqcup 10R1 \triangleright \sqcup 101R \triangleright \sqcup 101 \sqcup R \triangleright \sqcup 101a$
$\triangleright \sqcup 100c \triangleright \sqcup 10a0 \triangleright \sqcup 11L0 \triangleright \sqcup 1L10 \triangleright \sqcup L110$
$139519951501021117250707811249365378266947177658$
$835936109568606514550208131 54$

$\vdash_M \quad (h, \triangleright \underline{\sqcup} 110)$

$\triangleright \sqcup s101 \triangleright \sqcup 1R01 \triangleright \sqcup 10R1 \triangleright \sqcup 101R \triangleright \sqcup 101 \sqcup R \triangleright \sqcup 101a$
$\triangleright \sqcup 100c \triangleright \sqcup 10a0 \triangleright \sqcup 11L0 \triangleright \sqcup 1L10 \triangleright \sqcup L110 \triangleright \sqcup h110$
$105052049754519253909942549028330159281459638428$
$060089903069676722114031593394042730$

$$\mathrm{num}(\mathrm{output}(\mathrm{last}(\mathrm{comp}(5)))) = 6$$

$\mathrm{comp}(5) = \triangleright \sqcup s101 \triangleright \sqcup 1R01 \triangleright \sqcup 10R1 \triangleright \sqcup 101R \triangleright \sqcup 101 \sqcup R \triangleright \sqcup 101$
$a \triangleright \sqcup 100c \triangleright \sqcup 10a0 \triangleright \sqcup 11L0 \triangleright \sqcup 1L10 \triangleright \sqcup L110 \triangleright \sqcup h110 = 1050520$
$4975451925390994254902833015928145963842806008990306967$
6722114031593394042730

$\mathrm{last}(\mathrm{comp}(5)) = \mathrm{last}(\triangleright \sqcup s101 \triangleright \sqcup 1R01 \triangleright \sqcup 10R1 \triangleright \sqcup 101R \triangleright \sqcup 101 \sqcup$
$R \triangleright \sqcup 101a \triangleright \sqcup 100c \triangleright \sqcup 10a0 \triangleright \sqcup 11L0 \triangleright \sqcup 1L10 \triangleright \sqcup L110 \triangleright \sqcup h110)$
$= \triangleright \sqcup h110 = 1726186$

$\mathrm{output}(\mathrm{last}(\mathrm{comp}(5))) = \mathrm{output}(1726186) = \mathrm{output}(\triangleright \sqcup h110)$
$= 110 = 210$

$\mathrm{num}(\mathrm{output}(\mathrm{last}(\mathrm{comp}(5)))) = \mathrm{num}(210) = \mathrm{num}(110) = 6$

LOOP-, WHILE- und GOTO-Berechenbarkeit

Variablen:	$x_0, x_1, x_2, \ldots$
Konstanten:	$0, 1, 2, \ldots$
Trennsymbole:	$;\ :=$
Operationszeichen:	$+, -$
Schlüsselwörter:	LOOP, WHILE, GOTO, DO, END

LOOP-Berechenbarkeit

$$x_i := x_j + c \quad \text{und} \quad x_i := x_j - c$$

sind LOOP-Programme.
Falls Π_1 und Π_2 LOOP-Programme sind, dann ist auch

$$\Pi_1; \Pi_2$$

ein LOOP-Programm.
Falls Π ein LOOP-Programm ist und x_i eine Variable, dann ist auch

$$\text{LOOP } x_i \text{ DO } \Pi \text{ END}$$

ein LOOP-Programm.

Eine Funktion $f : \mathbb{N}^k \mapsto \mathbb{N}$ heißt LOOP-berechenbar, falls es ein LOOP-Programm Π gibt, das gestartet mit $n_1, \ldots, n_k$ in den Variablen $x_1, \ldots, x_k$ (und 0 in den restlichen Variablen) mit dem Wert $f(n_1, \ldots, n_k)$ in der Variablen x_0 hält.

Beispiel:

Addition: $x_1 + x_2$

$x_0 := x_1;$
LOOP x_2 DO $x_0 := x_0 + 1$ END

If-then-else:

$$\text{IF } x = 0 \text{ THEN } A \text{ END}$$

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

Satz: Eine Funktion ist LOOP*-berechenbar genau dann wenn sie primitiv rekursiv ist.*

o.B. □

WHILE-Berechenbarkeit

Jedes LOOP-Programm ist auch ein WHILE-Programm. Falls Π ein WHILE-Programm ist und x_i eine Variable, dann ist auch

$$\text{WHILE } x_i \neq 0 \text{ DO } \Pi \text{ END}$$

ein WHILE-Programm.

Eine Funktion $f : \mathbb{N}^k \mapsto \mathbb{N}$ heißt WHILE-berechenbar, falls es ein WHILE-Programm Π gibt, das gestartet mit $n_1, n_2, \ldots, n_k$ in den Variablen $x_1, \ldots, x_k$ (und 0 in den restlichen Variablen) mit dem Wert $f(n_1, \ldots, n_k)$ in der Variablen x_0 hält.

GOTO-Berechenbarkeit

Ein GOTO-Programm ist eine Folge von Paaren $M_i : A_i$ bestehend aus einer Markierung M_i und einer Anweisung A_i, getrennt durch ';' .

$$M_1 : A_1;\ M_2 : A_2;\ \ldots;\ M_k : A_k$$

Mögliche Anweisungen:

Wertzuweisungen:	$x_i := x_j \pm c$
unbedingter Sprung:	GOTO M_i
bedingter Sprung:	IF $x_i = c$ THEN GOTO M_j
Stopanweisung:	HALT

Eine Funktion $f : \mathbb{N}^k \mapsto \mathbb{N}$ heißt GOTO-berechenbar, falls es ein GOTO-Programm Π gibt, das gestartet mit $n_1, \ldots, n_k$ in den Variablen $x_1, \ldots, x_k$ (und 0 in den restlichen Variablen) mit dem Wert $f(n_1, \ldots, n_k)$ in der Variablen x_0 hält.

WHILE und GOTO

WHILE $x_i \neq 0$ DO Π END

M_1 : IF $x_i = 0$ THEN GOTO M_4;
M_2 : Π;
M_3 : GOTO M_1;
M_4 : ...

$$M_1 : A_1;\ M_2 : A_2;\ \ldots;\ M_k : A_k$$

$count := 1;$
WHILE $count \neq 0$ DO
    IF $count = 1$ THEN A'_1 END;
    IF $count = 2$ THEN A'_2 END;
    $\vdots$
    IF $count = k$ THEN A'_k END;
END

$A'_i \simeq A_i$ plus Modifikation von $count$

Satz: Eine Funktion ist WHILE*-berechenbar genau dann wenn sie* GOTO*-berechenbar ist.* □

Satz: Jede WHILE*-berechenbare Funktion kann durch ein* WHILE*-Programm mit nur einer* WHILE*-Schleife berechnet werden.* □

Satz: Eine Funktion ist WHILE*-berechenbar genau dann wenn sie μ-rekursiv ist.*

o.B. □

Churchsche These

Jede im intuitiven Sinn berechenbare Funktion ist auch durch eine Turing-maschine berechenbar.

Churchsche These

Jede im intuitiven Sinn berechenbare Funktion ist auch durch eine Turing-maschine berechenbar.

Berechenbarkeit, Unentscheidbarkeit

Eine Menge $A \subseteq \Sigma^*$ heißt entscheidbar, falls ihre charakteristische Funktion $\chi_A : \Sigma^* \mapsto \{0,1\}$ berechenbar ist.

$$\chi_A(w) = \begin{cases} 1 & w \in A \\ 0 & w \notin A \end{cases}$$

Universelle Turingmaschine

Eine Universelle Turingmaschine erhält als Eingabe eine Beschreibung einer Turingmaschine “M” und eine zugehörige kodierte Eingabe “w”. Die Universelle Turingmaschine U simuliert dann die Berechnung von M bei Eingabe w:

$$U(\text{“}M\text{”}\,\text{“}w\text{”}) = \text{“}M(w)\text{”}$$

Kodierung einer Turingmachine

$M = (K, \Sigma, \delta, s, H)$:

i, j so dass $\quad 2^i \geq |K|,\ 2^j \geq |\Sigma| + 2$

Zustand $\simeq$ Wort aus $q\{0,1\}^i$

Symbol $\simeq$ Wort aus $a\{0,1\}^j$

Übergang $\simeq$ Wort aus

$$(q\{0,1\}^i, a\{0,1\}^j, q\{0,1\}^i, a\{0,1\}^j)$$

"M" $\simeq$ Folge von Übergängen

"w" $\simeq$ Wort aus $(a\{0,1\}^j)^*$

Konvention:

Symbol	Kodierung
$\sqcup$	$a0^j$
$\triangleright$	$a0^{j-1}1$
$\leftarrow$	$a0^{j-2}10$
$\rightarrow$	$a0^{j-2}11$

$$s \simeq q0^i$$

Eine Universelle Turingmaschine kann leicht als 3-Band Turingmaschine realisiert werden:

Band 1: “v” = kodierter Bandinhalt der simulierten Maschine
Band 2: “M”
Band 3: kodierter Zustand der simulierten Maschine

Halteproblem

Prädikat $\mathsf{halts}(P, X)$ ist erfüllt, falls das Programm P bei Eingabe X terminiert.

Sei $\mathsf{diagonal}(X)$ gegeben durch:

M_1 : $\mathsf{if\ halts}(X, X)\ \mathsf{then\ goto}\ M_1\ \mathsf{else\ halt}$

$\mathsf{diagonal}(\mathsf{diagonal})?$

$$H = \{\text{“}M\text{”}\,\text{“}w\text{”} : \text{Turingmaschine } M \text{ hält bei Eingabe } w\}$$

Satz: *Die Sprache H ist rekursiv aufzählbar.*

Beweis: Eine Universelle Turingmaschine ist ein Semi-Entscheidungsverfahren für H. □

Satz: *Die Sprache H ist nicht rekursiv.*

Beweis:

$$H_1 = \{\text{“}M\text{”} : \text{Turingmaschine } M \text{ hält bei Eingabe “}M\text{”}\}$$

$$H \text{ rekursiv} \Rightarrow H_1 \text{ rekursiv}$$

$$\overline{H_1} = \{w : w \text{ ist keine Kodierung einer Turingmaschine oder es ist die Kodierung einer Turingmaschine } M\text{, die bei Eingabe “}M\text{” nicht hält.}\}$$

$$H_1 \text{ rekursiv} \Rightarrow \overline{H_1} \text{ rekursiv}$$

Angenommen, es gäbe eine Turingmaschine M^*, die ein Semi-Entscheidungsverfahren für $\overline{H_1}$ ist.

Gehört "M^*" zu $\overline{H_1}$?

Nach Definition von $\overline{H_1}$ gehört "M^*" zu $\overline{H_1}$ genau dann wenn M^* bei Eingabe "M^*" nicht hält, also die Eingabe "M^*" nicht akzeptiert. M^* ist aber ein Semi-Entscheidungsverfahren für $\overline{H_1}$. Das bedeutet, M^* akzeptiert "M^*" genau dann wenn "M^*" zu $\overline{H_1}$ gehört. M^* akzeptiert "M^*" also genau dann, wenn M^* die Eingabe "M^*" nicht akzeptiert. Widerspruch.

Es kann M^* nicht geben! □

Satz: Die Klasse der rekursiven Sprachen ist eine echte Teilmenge der Klasse der rekursiv aufzählbaren Sprachen. □

Satz: Die Klasse der rekursiv aufzählbaren Sprachen ist nicht abgeschlossen unter Komplementbildung. □

Reduktion

Seien $L_1, L_2 \subseteq \Sigma^*$ Sprachen. Eine rekursive Funktion $\tau : \Sigma^* \mapsto \Sigma^*$ heißt Reduktion von L_1 auf L_2, falls gilt

$$x \in L_1 \iff \tau(x) \in L_2$$

Satz: Falls L_1 nicht rekursiv ist und es eine Reduktion von L_1 auf L_2 gibt, so ist auch L_2 nicht rekursiv.

□

Unentscheidbare Probleme bei Turingmaschinen

Satz: Die folgenden Probleme sind unentscheidbar:

(a) Gegeben eine Turingmaschine M und ein Wort w, hält M bei Eingabe w?

(b) Gegeben eine Turingmaschine M, hält M bei leerem Eingabewort?

(c) Gegeben eine Turingmaschine M, gibt es ein Eingabewort, bei dem M hält?

(d) Gegeben eine Turingmaschine M, hält M bei jeder Eingabe?

(e) *Gegeben Turingmaschinen M_1 und M_2, halten M_1 und M_2 bei den gleichen Eingabewörtern?*

(f) *Gegeben eine Turingmaschine M, ist die Sprache, für die M ein Semi-Entscheidungsverfahren ist, regulär (kontextfrei, rekursiv)?*

Beweis:

(a): bereits gezeigt

(b): Reduktion von H auf

$$L_{(b)} = \{\text{“}M\text{”} : M \text{ hält bei Eingabe } \varepsilon\}$$

Zu gegebenem M und w konstruieren wir eine Turingmaschine M_w: Wenn M_w mit leerem Band gestartet wird, schreibt M_w zunächst w auf das Band und simuliert dann M.

Die Abbildung τ mit “M”“w” $\mapsto$ “M_w” ist rekursiv.

(c): Reduktion von $L_{(b)}$ auf

$$L_{(c)} = \{\text{“}M\text{”} : \text{ es gibt Eingabe, bei der } M \text{ hält}\}$$

Zu gegebenem M konstruieren wir eine Turingmaschine M': Die Maschine M' löscht zunächst jegliche Eingabe und simuliert dann M.

Die Abbildung τ mit “M” $\mapsto$ “M'” ist rekursiv.

(d): Reduktion von $L_{(b)}$ auf

$$L_{(d)} = \{\text{“}M\text{”} : \ M \text{ hält bei allen Eingaben}\}$$

siehe (c)

(e): Reduktion von $L_{(d)}$ auf

$$L_{(e)} = \{\text{“}M_1\text{”}\,\text{“}M_2\text{”} : M_1 \text{ und } M_2 \text{ halten bei den gleichen Eingaben}\}$$

Wir betrachten die Abbildung τ mit “M” $\mapsto$ “M”“Y”, wobei Y eine Turingmaschine ist, die jede Eingabe sofort akzeptiert.

(f): Reduktion von $L_{(b)}$ auf

$$L_{\text{reg}} = \{\text{“}M\text{”} : L(M) \text{ ist regulär}\}$$

Zu gegebenem M konstruieren wir eine Turingmaschine M': Die Maschine M' sichert die Eingabe (beispielsweise auf einem zweiten Band) und simuliert dann die Berechnung von M bei leerer Eingabe. Falls der Haltezustand von M erreicht wird, so wird die Eingabe wiederhergestellt und die Berechnung der Universellen Turingmaschine bei dieser Eingabe simuliert. $L(M')$ ist also entweder gleich der nicht rekursiven Sprache H, nämlich genau dann, wenn M bei leerer Eingabe hält, oder gleich der regulären (kontextfreien, rekursiven) Sprache $\emptyset$.

Die Abbildung τ mit “M” $\mapsto$ “M'” ist rekursiv. $\square$

Satz: Es gibt eine Turingmaschine M_0, für die das folgende Problem unentscheidbar ist:

Gegeben ein Wort w, hält M_0 bei Eingabe w?

Beweis:

Sei U eine Universelle Turingmaschine.

$$M_0 = U$$

Unentscheidbare Probleme bei Grammatiken

Satz: *Die folgenden Probleme sind unentscheidbar:*

(a) *Gegeben eine Grammatik G und ein Wort w, gilt $w \in L(G)$?*

(b) *Gegeben eine Grammatik G, gilt $\varepsilon \in L(G)$?*

(c) *Gegeben Grammatiken G_1 und G_2, gilt $L(G_1) = L(G_2)$?*

(d) *Gegeben eine Grammatik G, gilt $L(G) = \emptyset$?*

Ferner gibt es eine Grammatik G_0, für die das folgende Problem unentscheidbar ist:

Gegeben ein Wort w, gilt $w \in L(G_0)$?

Beweisskizze:

Wir reduzieren entsprechende Probleme bei Turingmaschinen auf diese Probleme. Zu einer gegebenen Maschine M konstruieren wir eine Grammatik G, so dass $L(G) = L(M)$. Diese Konstruktion haben wir bereits kennengelernt.
Für (a) ergibt sich somit beispielsweise, dass M bei Eingabe w hält genau dann wenn $w \in L(M) = L(G)$. □

Unentscheidbare Probleme bei kontextfreien Grammatiken

Satz: *Die folgenden Probleme sind unentscheidbar:*

(a) *Gegeben eine kontextfreie Grammatik G, gilt $L(G) = \Sigma^*$?*

(b) *Gegeben kontextfreie Grammatiken G_1 und G_2, gilt $L(G_1) = L(G_2)$?*

(c) *Gegeben Kellerautomaten M_1 und M_2, akzeptieren M_1 und M_2 die gleiche Sprache?*

Beweisskizze:

Sei $G_1 = (V_1, \Sigma_1, R_1, S_1)$ eine (uneingeschränkte) Grammatik. $G_1' = (V_1', \Sigma_1, R_1', S_1)$ entsteht aus G_1, indem jede Regel $\alpha_i \to \beta_i$ ersetzt wird durch zwei Regeln $\alpha_i \to A_i$ und $A_i \to \beta_i$, wobei die A_i neue Nichtterminalsymbole sind. Es gilt $L(G_1') = L(G_1)$.

Wir konstruieren eine kontextfreie Grammatik G_2, so dass gilt

$$L(G_2) = \Gamma^* \quad \text{g.d.w.} \quad L(G_1') = \emptyset$$

Zu einer Ableitung $S_1 \Rightarrow x_1 \Rightarrow x_2 \Rightarrow x_3 \Rightarrow \cdots \Rightarrow x_{n-1} \Rightarrow x_n$ von $x_n \in \Sigma_1^*$ in G_1' aus S_1 betrachten wir folgendes Wort über dem Alphabet $\Gamma = V_1' \cup \{\Rightarrow\}$:

$$S_1 \Rightarrow x_1^R \Rightarrow x_2 \Rightarrow x_3^R \Rightarrow \cdots \Rightarrow x_{n-1}^R \Rightarrow x_n$$

Sei $D_{G_1'} \subseteq \Gamma^*$ die Sprache, die aus allen Wörtern besteht, die zu Ableitungen von Wörtern aus Σ_1^* korrespondieren.

Es gilt $D_{G_1'} = \emptyset$ genau dann wenn $L(G_1') = \emptyset$. Oder anders gesagt,

$$\overline{D_{G_1'}} = \Gamma^* \quad \text{g.d.w.} \quad L(G_1') = \emptyset$$

Man kann nun zeigen, dass die Komplementsprache $\overline{D_{G_1'}}$ eine kontextfreie Sprache ist.

(b) und (c) lassen sich mit Hilfe von (a) beweisen. $\square$

Satz: Es gibt keinen Algorithmus, mit dem man zu einem beliebigen Kellerautomaten einen äquivalenten Kellerautomaten mit minimal vielen Zuständen konstruieren kann.

Beweisskizze:

Es gibt einen Kellerautomaten mit nur einem Zustand, der Σ^* akzeptiert. Für einen Kellerautomaten M_1 mit nur einem Zustand kann man entscheiden, ob die von M_1 akzeptierte Sprache Σ^* ist. Gäbe es einen solchen Algorithmus, so könnte man Problem (a) des vorangegangenen Satzes dadurch entscheiden, dass man einen zu G äquivalenten Kellerautomaten und zu diesem mit Hilfe des Algorithmus einen äquivalenten Automaten mit minimal vielen Zuständen konstruiert. Hat der Minimalautomat mehr als einen Zustand, so ist die akzeptierte Sprache nicht Σ^*. Hat er genau einen Zustand, so kann man entscheiden, ob die akzeptierte Sprache Σ^* ist. □

Weiteres zu rekursiv aufzählbaren Sprachen

Satz: Eine Sprache L ist rekursiv genau dann wenn sowohl L als auch ihr Komplement $\overline{L}$ rekursiv aufzählbar sind.

□

Eine Turingmaschine $M = (K, \Sigma, \delta, s, H)$ zählt eine Sprache $L \subseteq \Sigma^*$ auf, genau dann wenn es einen ausgezeichneten Zustand $q \in K$ gibt, so dass

$$L = \{w : (s, \triangleright \sqcup) \vdash_M^* (q, \triangleright \sqcup w)\}$$

Eine Sprache L heißt Turing-aufzählbar, wenn es eine Turingmaschine gibt, die L aufzählt.

Satz: Eine Sprache ist rekursiv aufzählbar genau dann wenn sie Turing-aufzählbar ist.

□

Sei M eine Turingmaschine, die die Sprache L aufzählt, und sei q der ausgezeichnete Zustand.
M zählt L lexikographisch auf, falls aus

$$(q, \triangleright \sqcup w) \vdash_M^+ (q, \triangleright \sqcup w')$$

folgt, dass w lexikographisch kleiner ist als w'.

Eine Sprache L heißt lexikographisch Turing-aufzählbar wenn es eine Turingmaschine M gibt, die L lexikographisch aufzählt.

Satz: Eine Sprache ist rekursiv genau dann wenn sie lexikographisch Turing-aufzählbar ist.

□

Satz von Rice

Satz: Sei $\mathcal{C}$ eine echte, nicht-leere Teilmenge der Klasse aller rekursiv aufzählbaren Sprachen. Dann ist das folgende Problem unentscheidbar:
Gegeben eine Turingmaschine M, gilt

$$L(M) \in \mathcal{C}$$

Beweisskizze: O.B.d.A. $\emptyset \notin \mathcal{C}$. Sei L eine Sprache in $\mathcal{C}$ und M_L eine Turingmaschine, die L semi-entscheidet. Wir reduzieren das Halteproblem auf obiges Problem.

Sei also M eine Turingmaschine und w ein Eingabewort. Wir basteln eine Turingmaschine $T_{M,w}$, die folgendes Verhalten zeigt:

$$T_{M,w}(x) = \begin{cases} M_L(x) & \text{falls } U(\text{“}M\text{”}\,\text{“}w\text{”}) \neq \nearrow \\ \nearrow & \text{sonst} \end{cases}$$

Die Sprache, für die $T_{M,w}$ ein Semi-Entscheidungsverfahren ist, gehört zu $\mathcal{C}$ genau dann wenn M bei Eingabe w hält.

Die Funktion, die zu gegebenen M und w die Turingmaschine $T_{M,w}$ liefert, ist rekursiv. □

24. Juni
Probeklausur

Postsches Korrespondenzproblem

Sei Σ ein Alphabet. Eine endliche Folge von geordneten Wortpaaren

$$(x_1, y_1), (x_2, y_2), \cdots, (x_k, y_k)$$

mit $x_i, y_i \in \Sigma^+$ heißt Postsches Korrespondenzsystem.

Eine Folge von Indizes $i_1, i_2, \ldots, i_n \in \{1, 2, \ldots, k\}$ heißt Lösung des Korrespondenzsystems, falls gilt

$$x_{i_1} x_{i_2} \ldots x_{i_n} = y_{i_1} y_{i_2} \ldots y_{i_n}$$

Das Postsche Korrespondenzproblem ist, zu entscheiden, ob ein gegebenes Postsches Korrespondenzsystem eine Lösung hat.

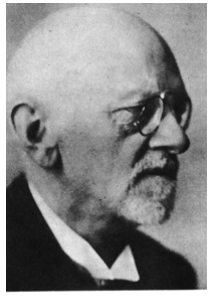
Satz: Das Postsche Korrespondenzproblem ist unentscheidbar.

(o.B.) □

Satz: Das Postsche Korrespondenzproblem ist semi-entscheidbar.

□

10. Hilbertsches Problem



10. Entscheidung der Lösbarkeit einer diophantischen Gleichung. *Eine diophantische Gleichung mit irgendwelchen Unbekannten und mit ganzen rationalen Zahlkoefficienten sei vorgelegt: man soll ein Verfahren angeben, nach welchem sich mittels einer endlichen Anzahl von Operationen entscheiden lässt, ob die Gleichung in ganzen rationalen Zahlen lösbar ist.*

Satz: Das 10. Hilbertsche Problem ist unentscheidbar.

Probleme versus Sprachen

Entscheidungsprobleme

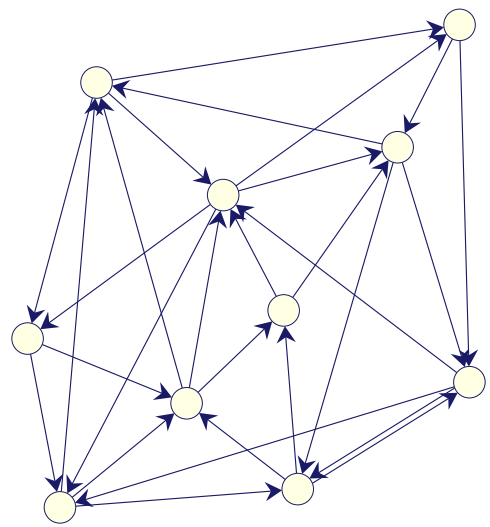
HALTEPROBLEM (HALTING PROBLEM):
Gegeben eine Turingmaschine M und ein Eingabewort w, hält M bei Eingabe w?

$H = \{\text{“}M\text{”}\,\text{“}w\text{”} : \text{Turingmaschine } M \text{ hält bei Eingabe } w\}$

ERREICHBARKEIT (REACHABILITY):

Gegeben ein gerichteter Graph $G = (V, E)$, $E \subseteq V \times V$, und zwei Knoten u und v aus V, gibt es in G einen Pfad von u nach v?

$L = \{\text{“}G\text{”}\,\text{“}u\text{”}\,\text{“}v\text{”} : \text{ es gibt in } G \text{ einen Pfad von } u \text{ nach } v\}$



"G": Kodierung des Graphen: $V = \{v_1, \ldots, v_n\}$.
Darstellung des Graphen durch $n \times n$ Adjazenzmatrix (a_{ij}) mit

$$a_{ij} = \begin{cases} 1 & \text{falls } (v_i, v_j) \in E \\ 0 & \text{sonst} \end{cases}$$

Eine Zeile der Adjazenzmatrix kann dargestellt werden als Folge von n Nullen und Einsen, umrahmt von \$-Zeichen, eine Adjazenzmatrix als Folge von n solchen Zeilendarstellungen.

"u": Binärdarstellung von k, falls $u = v_k$,
umrahmt von \$-Zeichen.

"v": Binärdarstellung von l, falls $v = v_l$,
umrahmt von \$-Zeichen.

$$
\begin{aligned}
L &= \{\kappa(G)\$\mathbf{b}(i)\$\$\mathbf{b}(j)\$: \text{ es gibt in } G \text{ Pfad von } v_i \text{ nach } v_j\} \\
&\subseteq \{0,1,\$\}^*
\end{aligned}
$$

Optimierungsprobleme

PROBLEM DES HANDLUNGSREISENDEN (TRAVELING SALESPERSON PROBLEM):
Gegeben sind n Städte und eine Distanzmatrix (d_{ij}) mit nichtnegativen ganzzahligen Einträgen und $d_{ii} = 0$ und $d_{ij} = d_{ji}$. Finde die kürzeste Rundreise, d.h., finde die Permutation π von $\{1, 2, \ldots, n\}$, für die die zurückgelegte Wegstrecke

$$c(\pi) = d_{\pi(1)\pi(2)} + d_{\pi(2)\pi(3)} + \cdots d_{\pi(n-1)\pi(n)} + d_{\pi(n)\pi(1)}$$

mimimal ist.

Formulierung als Entscheidungsproblem

PROBLEM DES HANDLUNGSREISENDEN (TRAVELING SALESPERSON PROBLEM):

Gegeben sind n Städte und eine Distanzmatrix (d_{ij}) mit nichtnegativen ganzzahligen Einträgen und $d_{ii} = 0$ und $d_{ij} = d_{ji}$ und eine ganze Zahl B. Gibt es eine Permutation π, so dass

$$c(\pi) \leq B \, ?$$

Ein Optimierungsproblem kann nicht effizient lösbar sein, wenn das zugehörige Entscheidungsproblem nicht effizient lösbar ist!

Wir werden von nun an meist nicht mehr zwischen Problemen und Sprachen unterscheiden.

Komplexität

Eine Turingmaschine $M = (K, \Sigma, \delta, s, H)$ heißt polynomiell (zeit)beschränkt, falls es ein Polynom $p(n)$ gibt, so dass für alle $x \in \Sigma^*$ gilt: Es gibt keine Konfiguration C, so dass

$$(s, \triangleright \sqcup x) \vdash_M^{p(|x|)+1} C$$

$\mathcal{P}$

Eine Sprache L heißt mit polynomiellem Zeitaufwand entscheidbar oder auch in Polynomialzeit entscheidbar, falls es eine polynomiell zeitbeschränkte Turingmaschine gibt, die L entscheidet.

Die Klasse der mit polynomiellem Zeitaufwand entscheidbaren Sprachen wird mit $\mathcal{P}$ bezeichnet.

Satz: $\mathcal{P}$ ist abgeschlossen unter Komplementbildung, Vereinigung, Schnitt, Konkatenation und Kleene star.

Satz: ERREICHBARKEIT $\in \mathcal{P}$.

Beweis: $G = (V,E)$, $v_i, v_j \in V$.

forall $w \in V$ do
    color[w] := weiß;
color[v_i] := grau;
while (es gibt graue Knoten) do
    w := ein grauer Knoten;
    forall $(w,z) \in E$ do
        if color[z] = weiß then color[z] := grau;
    color[w] := schwarz;

Alle schwarzen Knoten sind von $v_i \in V$ aus über Kanten in G erreichbar. Die Anzahl der Schritte des Algorithmus ist polynomiell in $|\kappa(G)\$\mathbf{b}(i)\$\$\mathbf{b}(j)\$|$. □

$\mathcal{NP}$

Eine nichtdeterministische Turingmaschine $M = (K, \Sigma, \Delta, s, H)$ heißt polynomiell (zeit)beschränkt, falls es ein Polynom $p(n)$ gibt, so dass für alle $x \in \Sigma^*$ gilt: Es gibt keine Konfiguration C, so dass

$$(s, \triangleright \sqcup x) \vdash_M^{p(|x|)+1} C$$

Die Klasse aller Sprachen, die durch eine polynomiell (zeit)beschränkte nichtdeterministische Turingmaschine entschieden werden können, wird mit $\mathcal{NP}$ bezeichnet.

Erinnerung: Falls eine nichtdeterministische Turingmaschine M eine Sprache L entscheidet, so gilt $w \in L$ genau dann wenn es eine Berechnung gibt, die im Haltezustand y endet:

$$(s, \triangleright \underline{\sqcup} w) \vdash_M^* (y, u\underline{a}v)$$

Es müssen also nicht alle Berechnungen für $w \in L$ im Haltezustand y enden, sondern nur mindestens eine!

$$\mathcal{P} \overset{?}{=} \mathcal{NP}$$

Boolesche Formeln

Boolesche Variablen: $x_1, x_2, \ldots, x_n$

Sei x_i eine Variable. Dann sind x_i und $\overline{x_i}$ Literale.

Seien $y_1, \ldots, y_k$ Literale. Dann ist $(y_1 \vee y_2 \vee \cdots \vee y_k)$ eine Klausel vom Grad k.

Seien $c_1, \ldots, c_m$ Klauseln. Dann ist $c_1 \wedge c_2 \wedge \cdots \wedge c_m$ eine Boolesche Formel in konjunktiver Normalform.

$$(x_1 \vee x_3) \wedge (\overline{x_2} \vee x_3 \vee x_4 \vee \overline{x_5}) \wedge (\overline{x_1} \vee \overline{x_4} \vee x_5) \wedge (x_2 \vee \overline{x_3} \vee x_5)$$

Belegung der Booleschen Variablen:

$$\Psi : \{x_1, x_2, \ldots\} \mapsto \{0, 1\}$$

Erweiterung auf Literale, Klauseln und Formeln in konjunktiver Normalform:

$$\Psi(y) = \begin{cases} \Psi(x_i) & \text{falls } y = x_i \\ 1 - \Psi(x_i) & \text{falls } y = \overline{x_i} \end{cases}$$

$$\Psi(y_1 \vee y_2 \vee \cdots \vee y_k) = \max_{1 \le i \le k} \Psi(y_i)$$

$$\Psi(c_1 \wedge c_2 \wedge \cdots \wedge c_m) = \min_{1 \le i \le m} \Psi(c_i)$$

Eine Boolesche Formel (in konjunktiver Normalform) α heißt erfüllbar, wenn es eine Belegung Ψ der Variablen gibt, so dass

$$\Psi(\alpha) = 1$$

ERFÜLLBARKEIT (SATISFIABILITY) (SAT):
Gegeben eine Boolesche Formel in konjunktiver Normalform, ist die Formel erfüllbar?

k-SAT:
Gegeben eine Boolesche Formel in konjunktiver Normalform, bei der alle Klauseln vom Grad (höchstens) *k* sind, ist die Formel erfüllbar?

Beispielkodierung:

Variable x_i:

$$\kappa(x_i) = \mathbf{b}(i)$$

Literal y:

$$\begin{cases} \$1\$\kappa(x_i)\$ & \text{falls } y = x_i \\ \$0\$\kappa(x_i)\$ & \text{falls } y = \overline{x_i} \end{cases}$$

Klausel $C = (y_1 \vee y_2 \vee \cdots \vee y_k)$:

$$\$\$\kappa(y_1)\kappa(y_2)\ldots\kappa(y_k)\$\$$$

Formel $\alpha = (c_1 \wedge c_2 \wedge \cdots \wedge c_m)$:

$$\kappa(c_1)\ldots\kappa(c_m)$$

Satz: SAT *ist entscheidbar.*

Beweisskizze:
Sei α eine Boolesche Formel in konjunktiver Normalform und sei n die Anzahl der Booleschen Variablen in α.

$$w = a_1 a_2 \dots a_n \in \{0,1\}^n$$

korrespondiert zu einer Belegung der Variablen: $\Psi(x_i) := a_i$.

Ob eine Belegung eine Klausel c erfüllt, kann in Zeit polynomiell in $|\kappa(c)|$ entschieden werden, ob eine Belegung eine Formel α erfüllt, in Zeit polynomiell in $|\kappa(\alpha)|$.

Erzeuge Wort für Wort alle Wörter aus $\{0,1\}^n$ und teste jeweils, ob die zugehörige Belegung die Formel erfüllt. $\square$

Satz: $\text{SAT} \in \mathcal{NP}$.

Beweisskizze:
Sei α eine Boolesche Formel in konjunktiver Normalform und sei n die Anzahl der Booleschen Variablen in α.

Sei M eine nichtdeterministische Turingmaschine mit folgenden Übergängen:

$$(s, \sqcup, q, \rightarrow),\ (q, \#, q, 0),$$
$$(q, \#, q, 1),\ (q, 0, q, \rightarrow),$$
$$(q, 1, q, \rightarrow),\ (q, \sqcup, h, \sqcup)$$

M wandelt $\#^n$ nichtdeterministisch in ein Wort $w \in \{0,1\}^n$.

Wir bestimmen aus α zunächst in Zeit polynomiell in $\kappa(\alpha)$ die Anzahl der Variablen n und erzeugen $\#^n$. Aus $\#^n$ erzeugen wir nichtdeterministisch ein $w \in \{0,1\}^n$ und testen in polynomieller Zeit, ob die zugehörige Belegung α erfüllt. $\square$

Satz: $2\text{-SAT} \in \mathcal{P}$.

Beweisskizze:

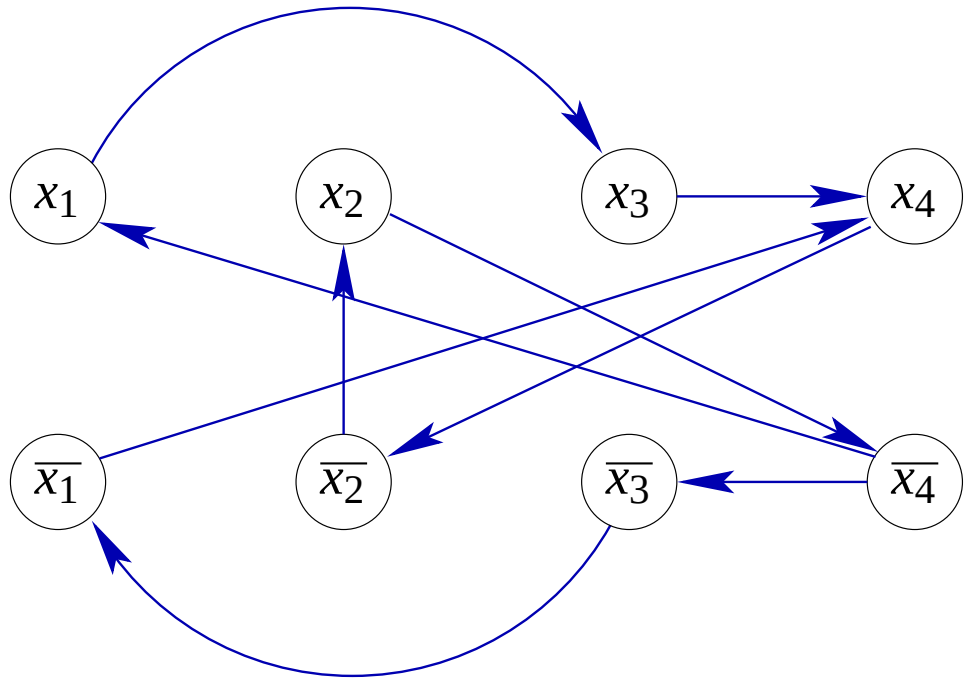
$(x \vee y)$ äquivalent zu $(\overline{x} \Rightarrow y)$ und zu $(\overline{y} \Rightarrow x)$

Zu einer Formel α konstruieren wir einen gerichteten Graphen $G_\alpha = (V, E)$, dessen Knotenmenge V aus allen Literalen besteht (positiven und negativen) und fügen eine Kante von u nach w zu E hinzu, genau dann wenn es in der Formel eine zur Implikation $(u \Rightarrow w)$ äquivalente Klausel gibt. G_α kann in polynomieller Zeit konstruiert werden.

α:

$$(x_1 \vee x_4) \wedge (\overline{x_1} \vee x_3) \wedge (x_2 \vee x_2) \wedge (x_4 \vee \overline{x_3}) \wedge (\overline{x_4} \vee \overline{x_2})$$

G_α:



Satz: α *ist genau dann unerfüllbar, wenn eine Variable* x *existiert, so dass es in* G_α *sowohl ein Pfad von* x *nach* $\bar{x}$ *als auch ein Pfad von* $\bar{x}$ *nach* x *gibt.*

Beweis:

$\Leftarrow$: Wenn es in G_α einen Pfad

$$x \Rightarrow \cdots \Rightarrow y$$

gibt, so bedeutet dies, dass bei einer die Formel erfüllenden Belegung der Variablen die Variable y mit 1 belegt werden muss, falls x mit 1 belegt wird. Wenn es also einen Kreis

$$\bar{x} \Rightarrow \cdots \Rightarrow \bar{x} \Rightarrow \cdots \Rightarrow x$$

in G_α gibt, kann es keine Belegung geben, die α erfüllt.

$\Rightarrow$: Beweis durch Induktion über die Anzahl n der Variablen:

$n = 1$:

$$(x \vee x) \wedge (\bar{x} \vee \bar{x})$$

$n > 1$: Falls α mehr als eine Variable enthält, wählen wir eine Variable x und betrachten die Formeln $\alpha_{x=0}$ und $\alpha_{x=1}$. Nach Induktionsvoraussetzung gibt es in $G_{\alpha_{x=0}}$ als auch in $G_{\alpha_{x=1}}$ Kreise der Form

$$y \Rightarrow \cdots \Rightarrow \bar{y} \Rightarrow \cdots \Rightarrow y$$

Kommt einer dieser Kreise auch in α vor, sind wir fertig. Wenn nicht, muss es im Kreis in $G_{\alpha_{x=0}}$ mindestens eine Kante $\bar{z} \Rightarrow z$ geben, die aus einer Klausel der Form $(x \vee z)$ aus α entstanden ist. In G_α gibt es dann den Pfad

$$\bar{x} \Rightarrow z \Rightarrow \cdots \Rightarrow \bar{u} \Rightarrow x$$

wobei u gleich z ist oder u eine andere Variable ist, für die die Klausel $(x \vee u)$ in α vorkommt.

Analog folgt aus der Existenz eines Kreises in $G_{\alpha_{x=1}}$, dass es einen Pfad $x \Rightarrow \cdots \Rightarrow \overline{x}$ in G_α geben muss. □

Ob es entsprechende Pfade gibt kann durch das Lösen von $2n$ Erreichbarkeitsproblemen in G_α überprüft werden, also in polynomieller Zeit. □

$\mathcal{EXP}$

Eine Turingmaschine $M = (K, \Sigma, \delta, s, H)$ heißt exponentiell (zeit)beschränkt, falls es ein Polynom $p(n)$ gibt, so dass für alle $x \in \Sigma^*$ gilt: Es gibt keine Konfiguration C, so dass

$$(s, \triangleright \sqcup x) \vdash_M^{2^{p(|x|)}+1} C$$

Die Klasse der mit exponentiellem Zeitaufwand entscheidbaren Sprachen wird mit $\mathcal{EXP}$ bezeichnet.

Satz: *Es gilt*

$$\mathcal{P} \subseteq \mathcal{NP} \subseteq \mathcal{EXP}$$

Beweis:

Jede deterministische Turingmaschine ist auch eine nichtdeterministische Turingmaschine. Also gilt $\mathcal{P} \subseteq \mathcal{NP}$.

Sei $L \in \mathcal{NP}$ und M eine polynomiell beschränkte nichtdeterministische Turingmaschine mit Zeitschranke $p(n)$, die L entscheidet. O.B.d.A. gelte $p(n) \geq n$.

Für $M = (K, \Sigma, \Delta, s, H)$ gibt es ein $r \in \mathbb{N}$, so dass für alle Paare (q, a) gilt $|\{(p, b) : (q, a, p, b) \in \Delta\}| \leq r$.

Wie wir bereits wissen, gibt es eine zu M äquivalente deterministische 3-Band Turingmaschine M', die zunächst alle Berechnungen von M der Länge 1 simuliert, dann alle der Länge 2, usw. Da es für eine Eingabe der Länge n keine Berechnung von M der Länge $p(n)+1$ gibt, werden nur Berechnungen der Länge ℓ, $\ell = 1, 2, \ldots, p(n)$, simuliert. Jede Berechnung von M der Länge ℓ kann durch M' in $O(\ell)$ Schritten simuliert werden.

M' simuliert r^ℓ potentielle Berechnungen von M der Länge ℓ. M' führt also insgesamt

$$O\left(\sum_{\ell=1}^{p(n)} r^\ell \ell\right) = O\left(p(n) r^{p(n)}\right)$$

Schritte aus.

M' wiederum kann durch eine äquivalente 1-Band Turingmaschine simuliert werden, die dazu

$$O(p(n)r^{p(n)}(n+p(n)r^{p(n)})) = O(r^{4p(n)})$$

Schritte ausführt. D.h., es gibt eine Konstante c_0, so dass die Anzahl der Schritte von M'' bei Eingaben der Länge n durch $c_0 r^{4p(n)}$ beschränkt ist. Für alle $x \in \Sigma^*$ gilt also, dass es keine Konfiguration C gibt, so dass

$$(s, \triangleright \sqcup x) \vdash_{M''}^{2^{\hat{p}(|x|)+1}} C$$

wobei $\hat{p}(n) = \lceil \log_2 c_0 \rceil + 4\lceil \log_2 r \rceil p(n)$. Somit gilt $L \in \mathcal{EXP}$.

□

Weitere Probleme auf Graphen

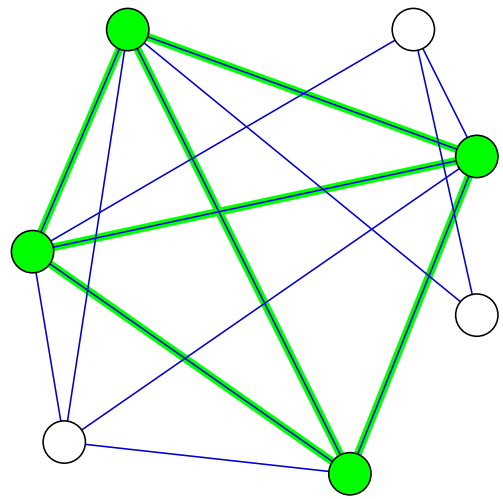
EULER-TOUR (EULER CYCLE):
Gegeben ein gerichteter Graph $G = (V, E)$, gibt es einen geschlossenen Pfad, der jede Kante genau einmal benutzt?

Satz: EULER CYCLE $\in \mathcal{P}$.

HAMILTON-KREIS (HAMILTONIAN CYCLE):
Gegeben ein gerichteter Graph $G = (V, E)$, gibt es einen geschlossenen Pfad, der jeden Knoten genau einmal besucht?

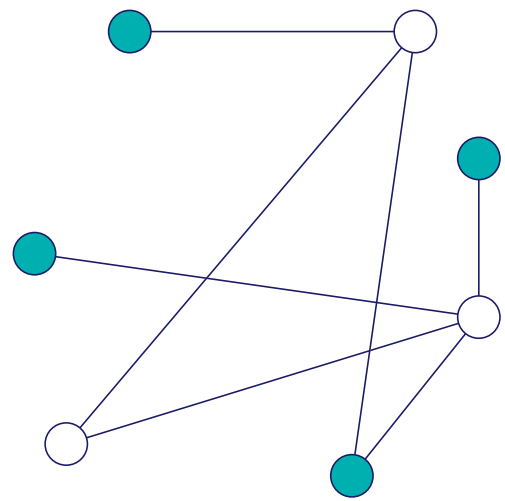
CLIQUE:

Gegeben ein ungerichteter Graph $G = (V, E)$ und $k \in \mathbb{N}$, gibt es eine Teilmenge $V' \subseteq V$ mit $|V'| = k$, so dass für alle $v_i, v_j \in V'$, $v_i \neq v_j$, gilt, dass die Kante zwischen v_i und v_j zu E gehört?



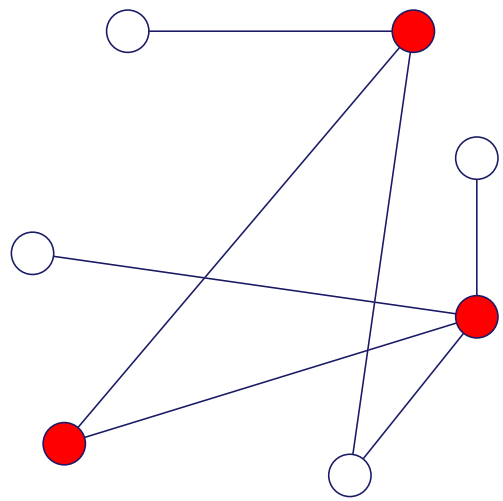
INDEPENDENT SET:

Gegeben ein ungerichteter Graph $G = (V, E)$ und $k \in \mathbb{N}$, gibt es eine Teilmenge $V' \subseteq V$ mit $|V'| = k$, so dass für alle $v_i, v_j \in V'$, $v_i \neq v_j$, gilt, dass es keine Kante zwischen v_i und v_j in G gibt?



KNOTENÜBERDECKUNG (VERTEX COVER):

Gegeben ein ungerichteter Graph $G = (V, E)$ und $k \in \mathbb{N}$, gibt es eine Teilmenge $V' \subseteq V$ mit $|V'| = k$, so dass für alle Kanten $\{u, v\} \in E$ gilt [$u \in V'$ oder $v \in V'$], d.h., mindestens einer der Endpunkte zu V' gehört?



Packungsprobleme

TWO-MACHINE SCHEDULING:

Gegeben $a_1, a_2, \ldots, a_n, D \in \mathbb{N}$ in Binärdarstellung, gibt es $I \subseteq \{1, 2, \ldots, n\}$, so dass

$$\sum_{i \in I} a_i \leq D \text{ und } \sum_{i \notin I} a_i \leq D \quad ?$$

PARTITION:

Gegeben $a_1, a_2, \ldots, a_n \in \mathbb{N}$ in Binärdarstellung, gibt es $I \subseteq \{1, 2, \ldots, n\}$, so dass

$$\sum_{i \in I} a_i = \sum_{i \notin I} a_i \quad ?$$

$H = \frac{1}{2}\sum_{j=1}^{n} a_j.$

$$B(i) = \{b \leq H : \exists I_i \subseteq \{1,2,\ldots,i\} \text{ so dass } \sum_{j \in I_i} a_j = b\}$$

$B(0) := \{0\}$
for $i = 1,2,\ldots,n$ do
    $B(i) := B(i-1)$
    for $j = a_i, a_i+1, a_i+2, \ldots, H$ do
        if $j - a_i \in B(i-1)$ then $B(i) := B(i) \cup \{j\}$

$O(nH)$ Schritte. Aber: nH ist nicht polynomiell in der Länge der Eingabe, in der die ganzzahligen Größen binär dargestellt sind!

Zertifikate

Sei Σ ein Alphabet und **;** ein weiteres Symbol, das nicht zu Σ gehört. Eine Sprache $L' \subseteq \Sigma^*;\Sigma^*$ heißt polynomiell balanciert, falls es ein Polynom $p(n)$ gibt, so dass aus $x;y \in L'$ folgt

$$|y| \leq p(|x|)$$

Satz: Sei $L \subseteq \Sigma^*$ und sei $; \notin \Sigma$. Dann gilt $L \in \mathcal{NP}$ genau dann wenn es eine polynomiell balancierte Sprache $L' \subseteq \Sigma^*;\Sigma^*$ gibt, so dass $L' \in \mathcal{P}$ und

$$L = \{x : \text{ es gibt } y \in \Sigma^*, \text{ so dass } x;y \in L'\}$$

$\mathcal{NP}$-Vollständigkeit

Eine Funktion $f : \Sigma^* \mapsto \Sigma^*$ heißt in polynomieller Zeit berechenbar, falls es eine polynomiell zeitbeschränkte deterministische Turingmaschine gibt, die f berechnet.

Polynomialzeitreduktion

Seien $L_1, L_2 \subseteq \Sigma^*$ Sprachen. Eine in polynomieller Zeit berechenbare Funktion $\tau : \Sigma^* \mapsto \Sigma^*$ heißt Polynomialzeitreduktion von L_1 auf L_2, falls für alle $x \in \Sigma^*$ gilt

$$x \in L_1 \iff \tau(x) \in L_2$$

L_1 heißt in Polynomialzeit auf L_2 reduzierbar, falls es eine Polynomialzeitreduktion von L_1 auf L_2 gibt. Notation:

$$L_1 \preceq_{\mathcal{P}} L_2$$

Satz: *Seien* $L_1, L_2, L_3 \subseteq \Sigma^*$ *Sprachen. Falls* $L_1 \preceq_{\mathcal{P}} L_2$ *und* $L_2 \preceq_{\mathcal{P}} L_3$ *gilt, so gilt auch* $L_1 \preceq_{\mathcal{P}} L_3$.

Beweis:

$\tau_{12} : \Sigma^* \mapsto \Sigma^*, x \in L_1 \iff \tau_{12}(x) \in L_2,$
in polynomieller Zeit berechenbar: $p_{12}(n)$
$\tau_{23} : \Sigma^* \mapsto \Sigma^*, x \in L_2 \iff \tau_{23}(x) \in L_3,$
in polynomieller Zeit berechenbar: $p_{23}(n)$

$$\tau = \tau_{23} \circ \tau_{12}$$

Dann gilt $x \in L_1 \iff \tau(x) = \tau_{23}(\tau_{12}(x)) \in L_3$. Ferner gilt $|\tau_{12}(x)| \leq p_{12}(|x|) + |x|$. Also ist τ in polynomieller Zeit berechenbar mit $p(n) = p_{23}(p_{12}(n) + n)$. $\square$

$\mathcal{NP}$-vollständig

Eine Sprache $L \subseteq \Sigma^*$ heißt $\mathcal{NP}$-hart oder auch $\mathcal{NP}$-schwer ($\mathcal{NP}$-hard), falls für alle $L' \in \mathcal{NP}$ gilt $L' \preceq_{\mathcal{P}} L$.

Eine Sprache $L \subseteq \Sigma^*$ heißt $\mathcal{NP}$-vollständig, falls L $\mathcal{NP}$-hart ist und des Weiteren $L \in \mathcal{NP}$ gilt.

Satz: Sei L eine $\mathcal{NP}$-vollständige Sprache. Dann gilt $\mathcal{P} = \mathcal{NP}$ genau dann wenn $L \in \mathcal{P}$.

DOMINO

$\mathcal{D} = (D, H, V)$ heißt Dominosystem, falls D eine endliche Menge von Kacheln ist, und $H, V \subseteq D \times D$ Relationen auf D sind. Seien ferner $s \in \mathbb{N}$, $s > 0$ und $f_0 : \{0, 1, \ldots, s-1\} \mapsto D$ gegeben.

$f : \{0, 1, \ldots, s-1\} \times \{0, 1, \ldots, s-1\} \mapsto D$ ist eine legale $s \times s$-Kachelung ausgehend von f_0, falls

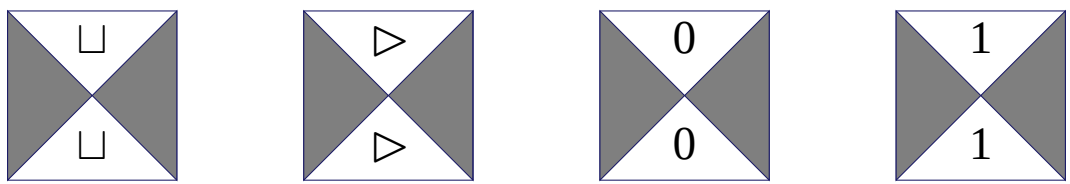
$f(m, 0) = f_0(m)$ für $0 \leq m < s$
$(f(m, n), f(m+1, n)) \in H$ für $0 \leq m < s-1,\ 0 \leq n < s$
$(f(m, n), f(m, n+1)) \in V$ für $0 \leq m < s,\ 0 \leq n < s-1$

DOMINO: Gegeben ein Dominosystem $\mathcal{D}$, $s > 0$ und f_0 wie oben beschrieben. Gibt es eine legale $s \times s$-Kachelung ausgehend von f_0?

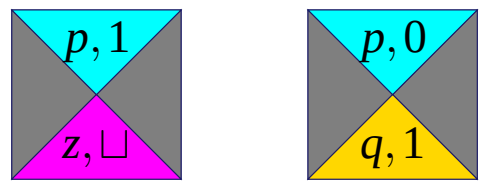
Satz: DOMINO *ist $\mathcal{NP}$-vollständig.*

Beweisskizze:
Wir müssen zeigen, dass wir jede Sprache in $\mathcal{NP}$ in polynomieller Zeit auf DOMINO reduzieren können. Sei $L \in \mathcal{NP}$. Dann gibt es eine polynomiell zeitbeschränkte nichtdeterministische Turingmaschine $M = (K, \Sigma, s, \Delta, H)$, die L entscheidet. Sei $p(n)$ ein M beschränkendes Polynom.

Zu einer Eingabe $w \in \Sigma^*$ konstruieren wir ein Dominosystem so, dass jede legale $s \times s$-Kachelung mit $s = p(|w|) + 2$ ausgehend von f_0 eine Berechnung von M bei Eingabe w repräsentiert: die obere Beschriftung der i-ten Kachelreihe repräsentiert die ersten s Zellen des Bandinhaltes von M nach $i - 1$ Schritten der Berechnung. f_0 stellt sicher, dass die unterste Kachelreihe dem Bandinhalt (gekürzt auf die s am weitesten links gelegenen Zellen – alle übrigen sind nicht relevant, weil in $s - 2$ Schritten nicht erreichbar) zu Beginn der Berechnung entspricht.
Für jedes Symbol aus Σ gibt es eine Kachel der folgenden Form

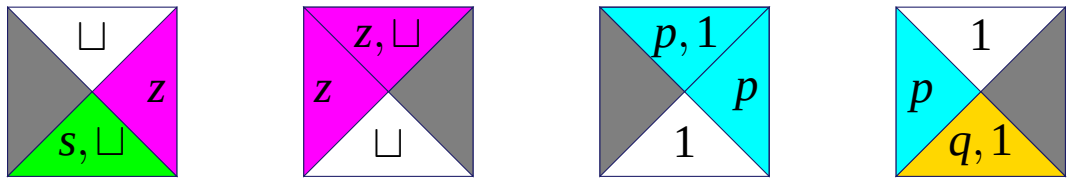


Die Position des Schreib-/Lesekopfes wird dadurch markiert, dass der Zustand zusammen mit dem Symbol unter dem Kopf auf die Kachel geschrieben wird. Der untere Rand entspricht dem Bandinhalt vor einem Berechnungsschritt, der obere dem Inhalt danach. Schreibaktionen, der Kopf bewegt sich bei solchen Aktionen nicht, werden durch Kacheln der folgenden Art repräsentiert:



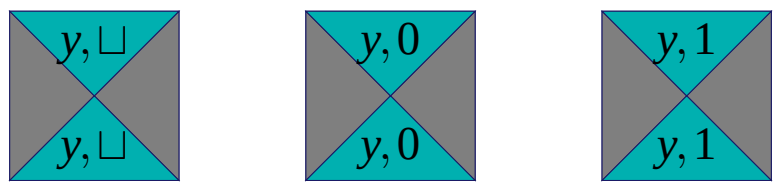
Obige Kacheln gehören zu D, falls $(z, \sqcup, p, 1) \in \Delta$
bzw. $(q, 1, p, 0) \in \Delta$.

Änderungen der Kopfposition werden durch Kachelpaare der folgenden Art propagiert:



Obige Kachelpaare gehören zu D, falls $(s, \sqcup, z, \rightarrow) \in \Delta$
bzw. $(q, 1, p, \leftarrow) \in \Delta$.

Nur eine Kachelreihe, die den (gekürzten) Bandinhalt bei Erreichen des akzeptierenden Haltezustands y repräsentiert, kann nach oben fortgesetzt werden. Dazu gibt es Kacheln der folgenden Art:



Bei Erreichen des verwerfenden Haltezustands ist das nicht möglich. Durch die Konstruktion von D ist sichergestellt, dass eine Kachelung mit s Kachelreihen genau dann existiert, wenn es eine akzeptierende Berechnung gibt, d.h., es gilt $w \in L$ genau dann wenn das konstruierte Dominosystem eine legale $s \times s$-Kachelung ausgehend von f_0 besitzt.

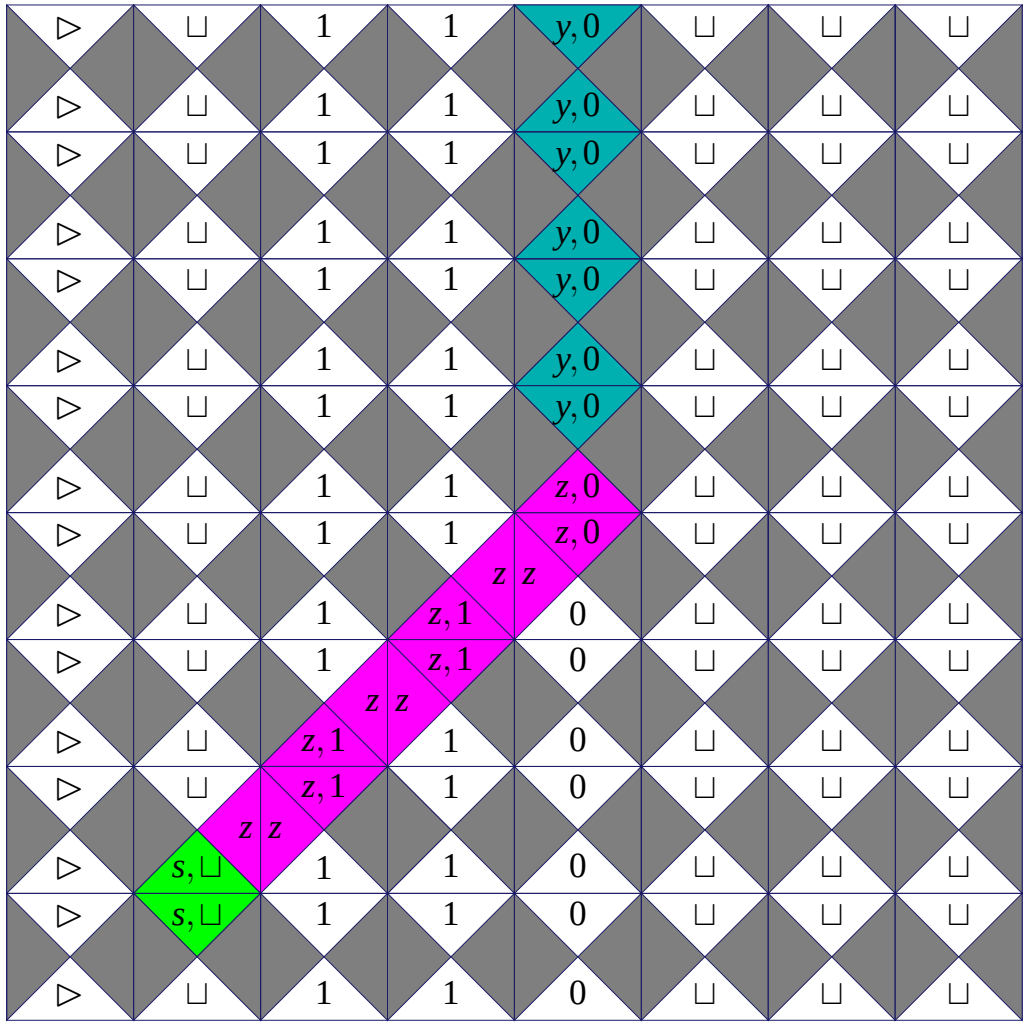
Wenn M und $p(n)$ bekannt sind, kann ein wie oben beschrieben geartetes Dominosystem D in polynomieller Zeit berechnet werden, d.h.

$$L \preceq_{\mathcal{P}} \text{DOMINO}$$

Ferner gilt $\text{DOMINO} \in \mathcal{NP}$, denn eine $s \times s$-Kachelung kann in polynomieller Zeit nichtdeterministisch erzeugt (geraten) und auf Legalität überprüft werden.

□

Beispiel:


▷ ⊔ 1 1 y,0 ⊔ ⊔ ⊔
▷ ⊔ 1 1 y,0 ⊔ ⊔ ⊔
▷ ⊔ 1 1 y,0 ⊔ ⊔ ⊔
▷ ⊔ 1 1 y,0 ⊔ ⊔ ⊔
▷ ⊔ 1 1 y,0 ⊔ ⊔ ⊔
▷ ⊔ 1 1 y,0 ⊔ ⊔ ⊔
▷ ⊔ 1 1 y,0 ⊔ ⊔ ⊔
▷ ⊔ 1 1 z,0 ⊔ ⊔ ⊔
▷ ⊔ 1 1 z,0 ⊔ ⊔ ⊔
z z
▷ ⊔ 1 z,1 0 ⊔ ⊔ ⊔
▷ ⊔ 1 z,1 0 ⊔ ⊔ ⊔
z z
▷ ⊔ z,1 1 0 ⊔ ⊔ ⊔
▷ ⊔ z,1 1 0 ⊔ ⊔ ⊔
z z
▷ s,⊔ 1 1 0 ⊔ ⊔ ⊔
▷ s,⊔ 1 1 0 ⊔ ⊔ ⊔
▷ ⊔ 1 1 0 ⊔ ⊔ ⊔

DOMINO

$\mathcal{D} = (D, H, V)$ heißt Dominosystem, falls D eine endliche Menge von Kacheln ist, und $H, V \subseteq D \times D$ Relationen auf D sind. Seien ferner $s \in \mathbb{N}$, $s > 0$ und $f_0 : \{0, 1, \ldots, s-1\} \mapsto D$ gegeben.

$f : \{0, 1, \ldots, s-1\} \times \{0, 1, \ldots, s-1\} \mapsto D$ ist eine legale $s \times s$-Kachelung ausgehend von f_0, falls

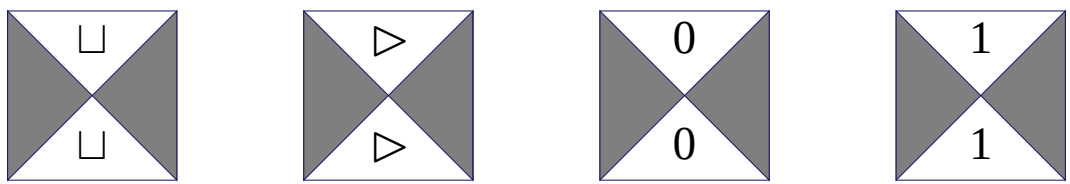
$f(m, 0) = f_0(m)$ für $0 \leq m < s$
$(f(m, n), f(m+1, n)) \in H$ für $0 \leq m < s-1,\ 0 \leq n < s$
$(f(m, n), f(m, n+1)) \in V$ für $0 \leq m < s,\ 0 \leq n < s-1$

DOMINO: Gegeben ein Dominosystem $\mathcal{D}$, $s > 0$ und f_0 wie oben beschrieben. Gibt es eine legale $s \times s$-Kachelung ausgehend von f_0?

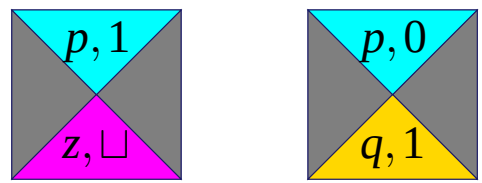
Satz: DOMINO *ist $\mathcal{NP}$-vollständig.*

Beweisskizze:
Wir müssen zeigen, dass wir jede Sprache in $\mathcal{NP}$ in polynomieller Zeit auf DOMINO reduzieren können. Sei $L \in \mathcal{NP}$. Dann gibt es eine polynomiell zeitbeschränkte nichtdeterministische Turingmaschine $M = (K, \Sigma, s, \Delta, H)$, die L entscheidet. Sei $p(n)$ ein M beschränkendes Polynom.

Zu einer Eingabe $w \in \Sigma^*$ konstruieren wir ein Dominosystem so, dass jede legale $s \times s$-Kachelung mit $s = p(|w|) + 2$ ausgehend von f_0 eine Berechnung von M bei Eingabe w repräsentiert: die obere Beschriftung der i-ten Kachelreihe repräsentiert die ersten s Zellen des Bandinhaltes von M nach $i - 1$ Schritten der Berechnung. f_0 stellt sicher, dass die unterste Kachelreihe dem Bandinhalt (gekürzt auf die s am weitesten links gelegenen Zellen – alle übrigen sind nicht relevant, weil in $s - 2$ Schritten nicht erreichbar) zu Beginn der Berechnung entspricht.
Für jedes Symbol aus Σ gibt es eine Kachel der folgenden Form

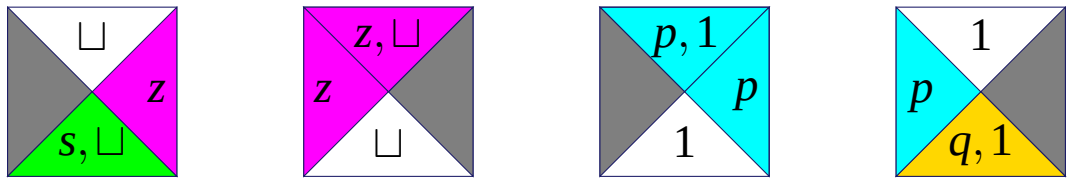


Die Position des Schreib-/Lesekopfes wird dadurch markiert, dass der Zustand zusammen mit dem Symbol unter dem Kopf auf die Kachel geschrieben wird. Der untere Rand entspricht dem Bandinhalt vor einem Berechnungsschritt, der obere dem Inhalt danach. Schreibaktionen, der Kopf bewegt sich bei solchen Aktionen nicht, werden durch Kacheln der folgenden Art repräsentiert:



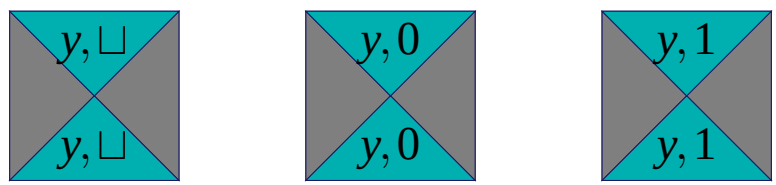
Obige Kacheln gehören zu D, falls $(z, \sqcup, p, 1) \in \Delta$
bzw. $(q, 1, p, 0) \in \Delta$.

Änderungen der Kopfposition werden durch Kachelpaare der folgenden Art propagiert:



Obige Kachelpaare gehören zu D, falls $(s, \sqcup, z, \rightarrow) \in \Delta$
bzw. $(q, 1, p, \leftarrow) \in \Delta$.

Nur eine Kachelreihe, die den (gekürzten) Bandinhalt bei Erreichen des akzeptierenden Haltezustands y repräsentiert, kann nach oben fortgesetzt werden. Dazu gibt es Kacheln der folgenden Art:



Bei Erreichen des verwerfenden Haltezustands ist das nicht möglich. Durch die Konstruktion von D ist sichergestellt, dass eine Kachelung mit s Kachelreihen genau dann existiert, wenn es eine akzeptierende Berechnung gibt, d.h., es gilt $w \in L$ genau dann wenn das konstruierte Dominosystem eine legale $s \times s$-Kachelung ausgehend von f_0 besitzt.

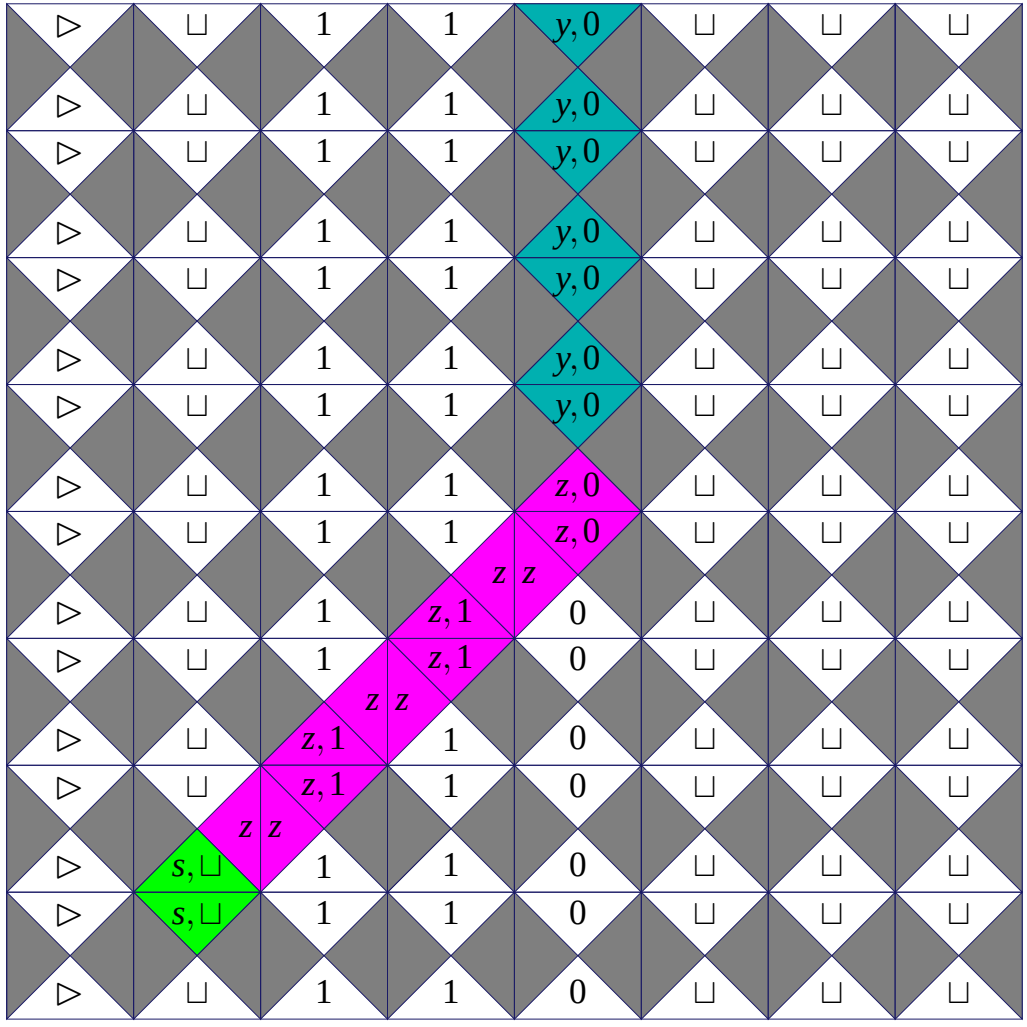
Wenn M und $p(n)$ bekannt sind, kann ein wie oben beschrieben geartetes Dominosystem D in polynomieller Zeit berechnet werden, d.h.

$$L \preceq_{\mathcal{P}} \text{DOMINO}$$

Ferner gilt $\text{DOMINO} \in \mathcal{NP}$, denn eine $s \times s$-Kachelung kann in polynomieller Zeit nichtdeterministisch erzeugt (geraten) und auf Legalität überprüft werden.

□

Beispiel:


▷ ⊔ 1 1 y,0 ⊔ ⊔ ⊔
▷ ⊔ 1 1 y,0 ⊔ ⊔ ⊔
▷ ⊔ 1 1 y,0 ⊔ ⊔ ⊔
▷ ⊔ 1 1 y,0 ⊔ ⊔ ⊔
▷ ⊔ 1 1 y,0 ⊔ ⊔ ⊔
▷ ⊔ 1 1 y,0 ⊔ ⊔ ⊔
▷ ⊔ 1 1 y,0 ⊔ ⊔ ⊔
▷ ⊔ 1 1 z,0 ⊔ ⊔ ⊔
▷ ⊔ 1 1 z,0 ⊔ ⊔ ⊔
z z
▷ ⊔ 1 z,1 0 ⊔ ⊔ ⊔
▷ ⊔ 1 z,1 0 ⊔ ⊔ ⊔
z z
▷ ⊔ z,1 1 0 ⊔ ⊔ ⊔
▷ ⊔ z,1 1 0 ⊔ ⊔ ⊔
z z
▷ s,⊔ 1 1 0 ⊔ ⊔ ⊔
▷ s,⊔ 1 1 0 ⊔ ⊔ ⊔
▷ ⊔ 1 1 0 ⊔ ⊔ ⊔

Satz von Cook

Satz (Cook): *Das Erfüllbarkeitsproblem für Boolesche Formeln in konjunktiver Normalform (*SATISFIABILITY*) ist $\mathcal{NP}$-vollständig.*

Beweisskizze: Wir zeigen

$$\text{DOMINO} \preceq_{\mathcal{P}} \text{SATISFIABILITY}$$

Zu gegebenen $\mathcal{D} = (D = \{d_1, \ldots, d_k\}, H, V)$, $s > 0$, f_0 konstruieren wir eine Boolesche Formel. Für $0 \leq m, n < s$ und $d \in D$ gibt es die Variable

$$x_{m,n,d}$$

Geplante Bedeutung: $x_{m,n,d} = 1$ g.d.w. $f(m,n) = d$.

Für $0 \leq m,n < s$ fügen wir die Klausel

$$\left(x_{m,n,d_1} \vee x_{m,n,d_2} \vee \ldots \vee x_{m,n,d_k}\right)$$

zur Formel hinzu und für $0 \leq m,n < s$ und alle $d \neq d'$ die Klausel

$$\left(\overline{x_{m,n,d}} \vee \overline{x_{m,n,d'}}\right)$$

sowie für $0 \leq i < s$ die Klausel

$$\left(x_{i,0,f_0(i)}\right)$$

Für $0 \leq n < s$ und $0 \leq m < s-1$ und alle $(d, d') \in D \times D - H$ gibt es die Klausel

$$\left(\overline{x_{m,n,d}} \vee \overline{x_{m+1,n,d'}}\right)$$

und für $0 \leq m < s$ und $0 \leq n < s-1$ und alle $(d, d') \in D \times D - V$ die Klausel

$$\left(\overline{x_{m,n,d}} \vee \overline{x_{m,n+1,d'}}\right)$$

Es ist nicht schwer, einzusehen, dass jede Belegung der $x_{m,n,d}$, die die Formel erfüllt, einer legalen $s \times s$-Kachelung ausgehend von f_0 entspricht und umgekehrt. Ferner kann die Formel in polynomieller Zeit aus D, s, f_0 konstruiert werden. $\square$

Satz: 3-SAT *ist $\mathcal{NP}$-vollständig.*

Beweisidee: (1) 3-SAT $\in \mathcal{NP}$

(2) SATISFIABILITY $\preceq_{\mathcal{P}}$ 3-SAT

$$C = (\lambda_1 \vee \lambda_2 \vee \cdots \vee \lambda_k)$$

wird ersetzt durch die Klauseln

$$(\lambda_1 \vee \lambda_2 \vee y_1),\ (\overline{y_1} \vee \lambda_3 \vee y_2),\ (\overline{y_2} \vee \lambda_4 \vee y_3),\ \ldots$$
$$\ldots,\ (\overline{y_{k-4}} \vee \lambda_{k-2} \vee y_{k-3}), (\overline{y_{k-3}} \vee \lambda_{k-1} \vee \lambda_k)$$

□

Überdeckungsprobleme

EXACT COVER:
Gegeben eine endliche Menge $U = \{u_1, \ldots, u_n\}$ und eine Familie $\mathcal{F}$ von Teilmengen von U, $\mathcal{F} = \{S_1, \ldots, S_m\}$, gibt es eine Teilmenge $\mathcal{C} \subseteq \mathcal{F}$, so dass die Mengen in $\mathcal{C}$ paarweise disjunkt sind und ihre Vereinigung U ergibt?

Satz: EXACT COVER *ist $\mathcal{NP}$-vollständig.*

Beweisidee:

(1) EXACT COVER $\in \mathcal{NP}$

(2) SATISFIABILITY $\preceq_{\mathcal{P}}$ EXACT COVER

Sei α eine Boolsche Formel mit Klauseln $C_1, \ldots, C_\ell$ in den Variablen $x_1, \ldots, x_n$. Mit λ_{jk}, $k = 1, 2, \ldots, m_j$, bezeichnen wir das k-te Literal in der Klausel C_j, die aus m_j Literalen besteht.

$\tau(\alpha) = (U, \mathcal{F})$ mit

$$
\begin{aligned}
U &= \{x_i : 1 \le i \le n\} \\
&\cup \{C_j : 1 \le j \le \ell\} \\
&\cup \{p_{jk} : 1 \le j \le \ell, 1 \le k \le m_j\}
\end{aligned}
$$

$$
\begin{aligned}
\mathcal{F} &= \{\, \{p_{jk}\} : 1 \le j \le \ell, 1 \le k \le m_j\} \\
&\cup \{\, \{x_i\} \cup \{p_{jk} : \lambda_{jk} = \overline{x_i}\} : 1 \le i \le n\} \\
&\cup \{\, \{x_i\} \cup \{p_{jk} : \lambda_{jk} = x_i\} : 1 \le i \le n\} \\
&\cup \{\, \{C_j, p_{jk}\} : 1 \le j \le \ell, 1 \le k \le m_j\}
\end{aligned}
$$

$$T_{i,\top} = \{x_i\} \cup \{p_{jk} : \lambda_{jk} = \overline{x_i}\} \in \mathcal{C} \quad \sim \quad \Psi(x_i) = 1$$
$$T_{i,\bot} = \{x_i\} \cup \{p_{jk} : \lambda_{jk} = x_i\} \in \mathcal{C} \quad \sim \quad \Psi(x_i) = 0$$

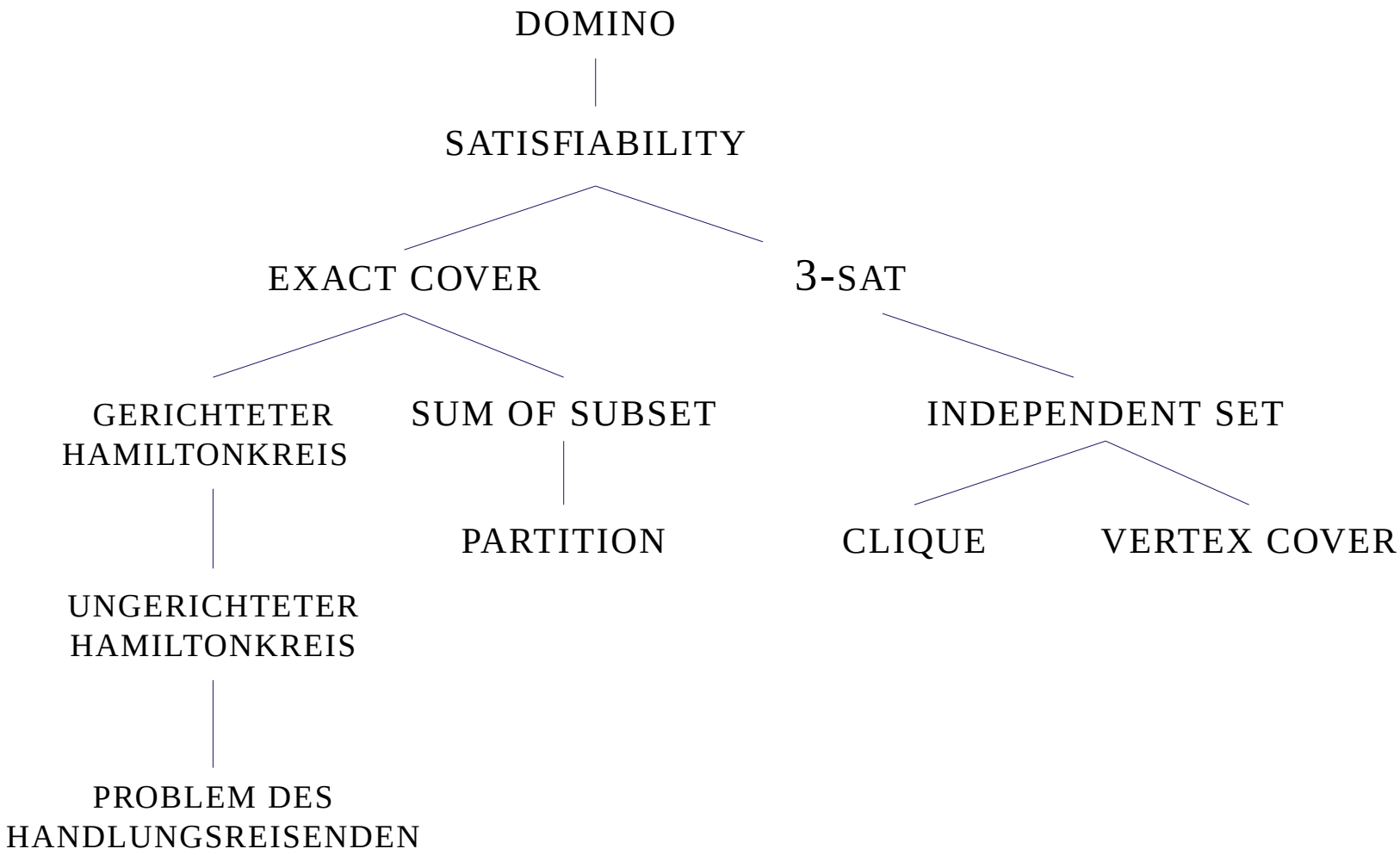
$\{C_j, p_{jk}\} \in \mathcal{C}$:

$$\begin{aligned} \lambda_{jk} &= x_i \\ &\Rightarrow T_{i,\bot} \notin \mathcal{C} \\ &\Rightarrow T_{i,\top} \in \mathcal{C} \\ &\Rightarrow \Psi(x_i) = 1 \end{aligned}$$

$$\begin{aligned} \lambda_{jk} &= \overline{x_i} \\ &\Rightarrow T_{i,\top} \notin \mathcal{C} \\ &\Rightarrow T_{i,\bot} \in \mathcal{C} \\ &\Rightarrow \Psi(x_i) = 0 \\ &\Rightarrow \Psi(\overline{x_i}) = 1 \end{aligned}$$

□

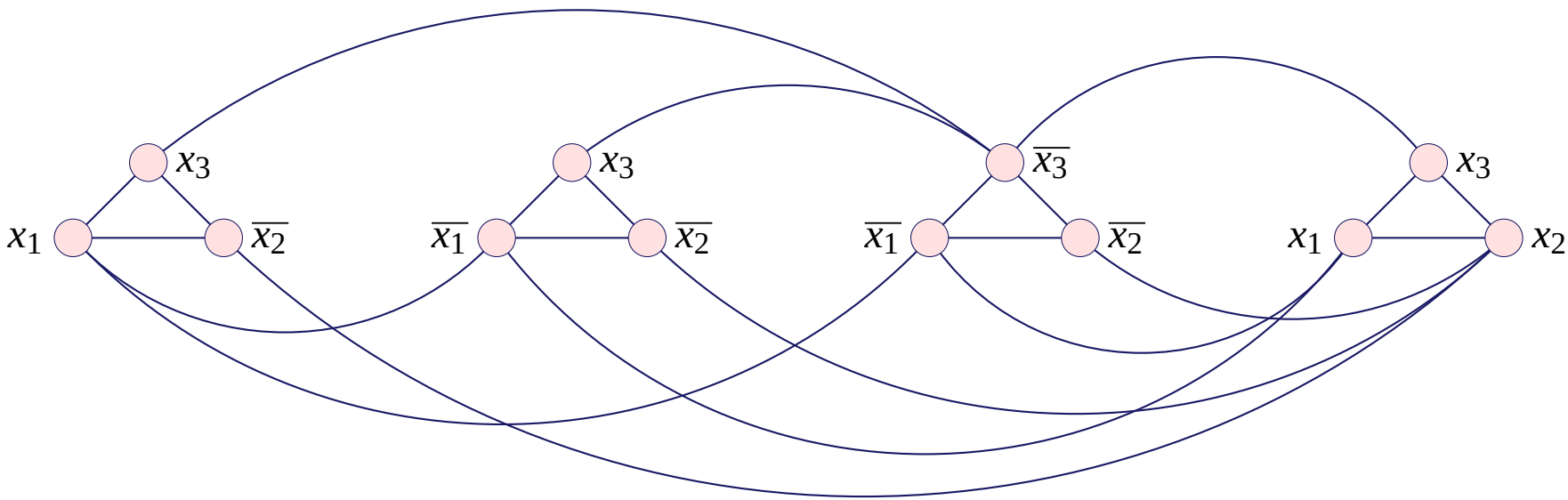
Weitere $\mathcal{NP}$-vollständige Probleme



Satz: 3-SAT $\preceq_{\mathcal{P}}$ INDEPENDENT SET

Beweisidee:

$$(x_1 \vee \overline{x_2} \vee x_3) \wedge (\overline{x_1} \vee \overline{x_2} \vee x_3) \wedge (\overline{x_1} \vee \overline{x_2} \vee \overline{x_3}) \wedge (x_1 \vee x_2 \vee x_3)$$



□

SUM OF SUBSET:

Gegeben $a_1, a_2, \ldots, a_n \in \mathbb{N}$ und $K \in \mathbb{N}$, gibt es $I \subseteq \{1, 2, \ldots, n\}$, so dass $\sum_{i \in I} a_i = K$?

Satz: EXACT COVER $\preceq_{\mathcal{P}}$ SUM OF SUBSET

Beweisidee:

$(\{u_1, \ldots, u_n\}, \{S_1, \ldots, S_m\})$

$$a_i = \sum_{u_j \in S_i} m^{j-1} \qquad K = \sum_{j=1}^{n} m^{j-1}$$

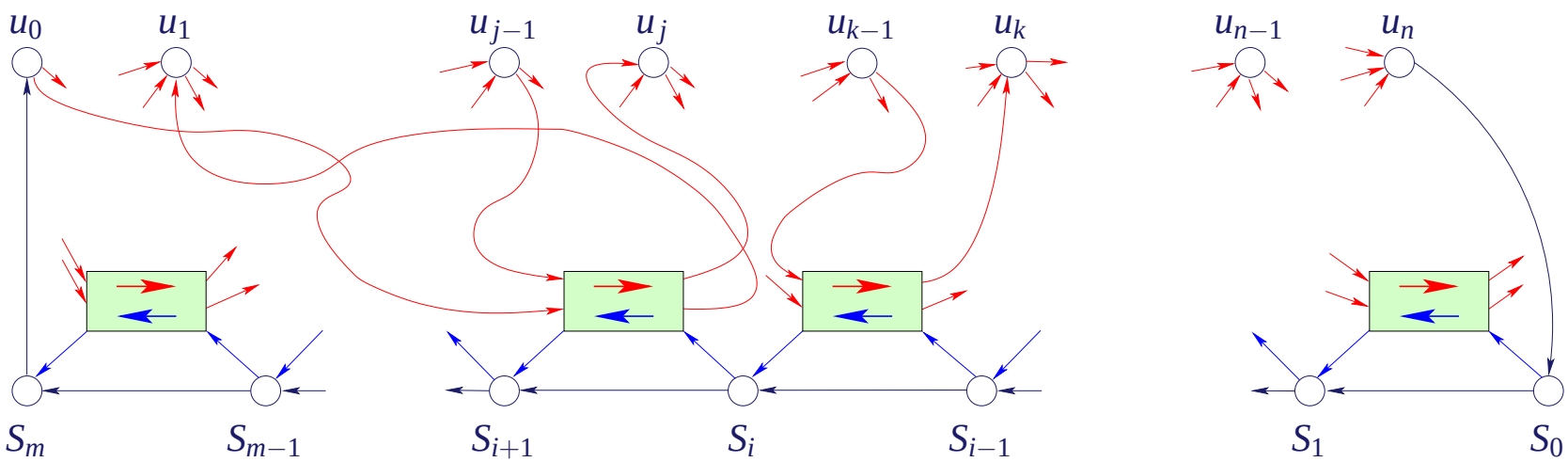
□

Satz:

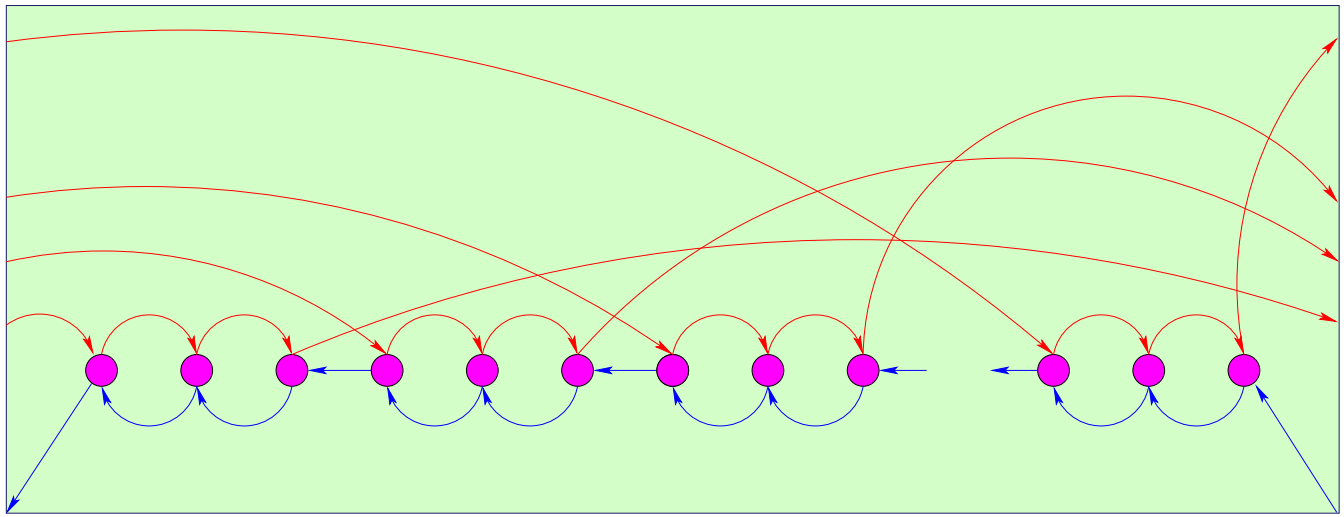
EXACT COVER $\preceq_{\mathcal{P}}$ GERICHTETER HAMILTONKREIS

Beweisidee:

$\tau(\{u_1,\ldots,u_n\},\{S_1,\ldots,S_m\}) = G$:



Für jede Teilmenge S_i existiert eine spezielle Graphkomponente. Es gibt einen roten Pfad von u_{k-1} durch die Komponente nach u_k genau dann wenn u_k zu S_i gehört.



Die Anzahl der roten Pfade ist also $|S_i|$. Um alle Knoten einer solchen Graphkomponente zu besuchen, müssen entweder *alle* roten Pfade benutzt werden oder nur der blaue.

Es gibt einen Hamiltonkreis im Graphen G genau dann wenn es für $(\{u_1,\ldots,u_n\},\{S_1,\ldots,S_m\})$ eine Menge paarweise disjunkter Teilmengen gibt, deren Vereinigung $\{u_1,\ldots,u_n\}$ ist:

S_j gehört nicht zur Menge $\mathcal{C}$ der exakt überdeckenden Teilmengen genau dann wenn der blaue Pfad von S_{j-1} nach S_j benutzt wird.

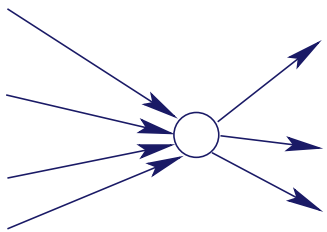
Wird der blaue Pfad nach S_j nicht benutzt, dann muss für alle $u \in S_j$ der rote Pfad benutzt werden.

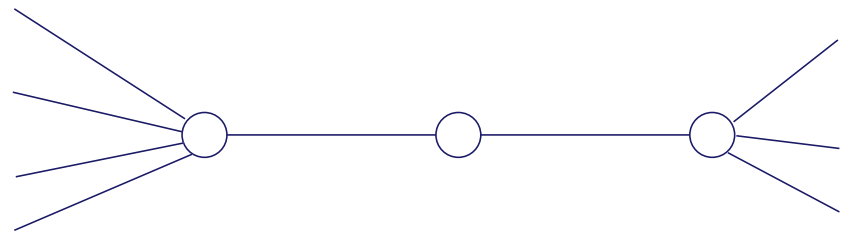
u_i gehört zu genau einer der Mengen in $\mathcal{C}$, wird also über genau einen roten Pfad erreicht. □

Satz:

$$\text{GERICHTETER HAMILTONKREIS} \preceq_{\mathcal{P}} \text{UNGERICHTETER HAMILTONKREIS}$$

Beweisidee:





□

Satz:

$$\text{UNGERICHTETER HAMILTONKREIS} \preceq_{\mathcal{P}} \text{PROBLEM DES HANDLUNGSREISENDEN}$$

Beweisidee:

$G = (\{v_1, \ldots, v_n\}, E)$

$$d_{ij} = \begin{cases} 0 & i = j \\ 1 & \{i, j\} \in E \\ 2 & \text{sonst} \end{cases}$$

□

Satz: Die Probleme GERICHTETER HAMILTONKREIS, UNGERICHTETER HAMILTONKREIS, *das* PROBLEM DES HANDLUNGSREISENDEN, SUM OF SUBSET, PARTITION, INDEPENDENT SET, CLIQUE *und* VERTEX COVER *sind $\mathcal{NP}$-vollständig.*

Beweis:

Wir haben gesehen, dass SATISFIABILITY auf jedes der Probleme in Polynomialzeit reduziert werden kann. Ferner ist jedes der Probleme in $\mathcal{NP}$. □

0/1 INTEGER LINEAR PROGRAMMING:
Gegeben m Ungleichungen

$$\sum_{j=1}^{n} a_{ij} x_j \geq b_i, \quad i = 1, \ldots, m$$

in n Variablen mit ganzzahligen Koeffizienten a_{ij} und b_i, gibt es eine Lösung, bei der alle Variablen den Wert 0 oder 1 haben?

Satz: 0/1 INTEGER LINEAR PROGRAMMING *ist $\mathcal{NP}$-vollständig.*

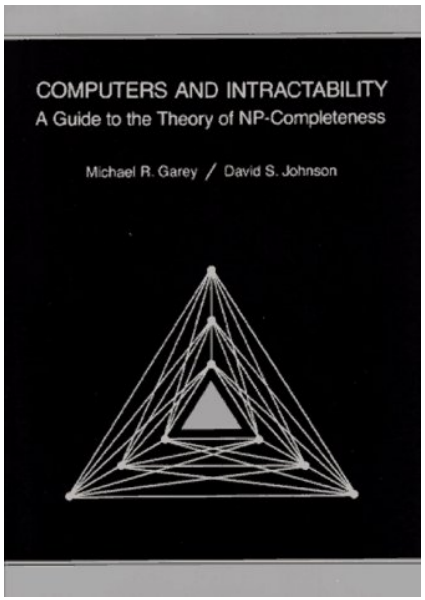
SET COVER:

Gegeben eine endliche Menge $U = \{u_1, \ldots, u_n\}$, eine Familie $\mathcal{F}$ von Teilmengen von U, $\mathcal{F} = \{S_1, \ldots, S_m\}$, und ein $k \in \mathbb{N}$, gibt es eine Teilmenge $\mathcal{C} \subseteq \mathcal{F}$ mit $|\mathcal{C}| = k$, so dass die Vereinigung der Mengen aus $\mathcal{C}$ die Menge U ergibt?

Satz: SET COVER *ist $\mathcal{NP}$-vollständig.*

Literaturhinweis zu $\mathcal{NP}$-Vollständigkeit

Michael R. Garey, David S. Johnson
Computers and Intractability:
A Guide to the Theory of NP-Completeness
Freeman (1979)



Weitere unentscheidbare Probleme bei kontextfreien Grammatiken

Satz: Die folgenden Probleme sind unentscheidbar:

(a) Gegeben eine kontextfreie Grammatik G, ist G mehrdeutig?

(b) Gegeben eine kontextfreie Grammatik G, ist $\overline{L(G)}$ kontextfrei?

(c) Gegeben eine kontextfreie Grammatik G, ist $L(G)$ regulär?

(d) *Gegeben eine kontextfreie Grammatik G, ist $L(G)$ deterministisch kontextfrei?*

(e) *Gegeben kontextfreie Grammatiken G_1 und G_2, ist $L(G_1) \cap L(G_2) = \emptyset$?*

(f) *Gegeben kontextfreie Grammatiken G_1 und G_2, ist $L(G_1) \cap L(G_2)$ endlich?*

(g) *Gegeben kontextfreie Grammatiken G_1 und G_2, ist $L(G_1) \cap L(G_2)$ kontextfrei?*

(h) *Gegeben kontextfreie Grammatiken G_1 und G_2, ist $L(G_1) \subseteq L(G_2)$?*