

# **Berechenbarkeit und Komplexität**

gehalten von Univ.-Prof. Dr. rer. nat. Wolfgang Thomas  
im Wintersemester 2004/05 an der RWTH Aachen

eine studentische Mitschrift von

Florian Heller

[florian@heller-web.net](mailto:florian@heller-web.net)

Diese Mitschrift erhebt keinen Anspruch auf Richtigkeit oder Vollständigkeit.

- Think Different -

17. März 2005



# Inhaltsverzeichnis

0.1	Einführung . . . . .	4
0.2	Gliederung der Vorlesung: . . . . .	4
<b>1</b>	<b>Berechenbarkeit und Turingmaschinen</b>	<b>5</b>
1.1	Beispiele algorithmischer Probleme . . . . .	5
1.2	Entscheidungsprobleme . . . . .	7
1.3	Algorithmen und Turingmaschinen . . . . .	9
1.4	Berechenbarkeit und nicht-Berechenbarkeit . . . . .	12
1.4.1	Church-Turing-These (1936) . . . . .	13
1.5	Berechenbarkeit, Entscheidbarkeit, Aufzählbarkeit . . . . .	15
<b>2</b>	<b>Registermaschinen und while-Programme</b>	<b>19</b>
2.1	goto-Programme . . . . .	19
2.1.1	Definition: . . . . .	19
2.2	while-Programme . . . . .	21
2.2.1	Idee: . . . . .	21
2.2.2	Definition der Programmiersprache while . . . . .	22
2.2.3	while <sub>m</sub> -Programme: Vorbereitung . . . . .	22
2.2.4	while <sub>m</sub> -Programme: Syntax . . . . .	22
2.2.5	while <sub>m</sub> -Programme: Semantik . . . . .	23
2.2.6	Vergleich loop-while . . . . .	24
2.2.7	Satz: (Satz von Ackermann) . . . . .	26
2.2.8	Äquivalenzsatz . . . . .	26
<b>3</b>	<b>Unentscheidbarkeit</b>	<b>31</b>
3.1	Wortproblem für TM . . . . .	31
3.1.1	Definition . . . . .	31
3.2	Reduktionen und weitere unentscheidbare Probleme . . . . .	34
3.2.1	Ziel: . . . . .	34
3.3	Drei unentscheidbare Probleme . . . . .	36
3.3.1	Domino-Problem . . . . .	37
3.3.2	Postisches Korrespondenzproblem (PCP) . . . . .	39
3.3.3	Halteproblem für goto <sub>2</sub> -Programme (2-Zähler-Maschinen) . . . . .	40
3.4	Universalität und Vollständigkeit . . . . .	41
<b>4</b>	<b>Komplexitätstheorie: Grundlagen</b>	<b>42</b>
4.1	Zeitkomplexität . . . . .	42
4.2	Klassen P und NP . . . . .	44
<b>5</b>	<b>NP-Vollständigkeit</b>	<b>50</b>
5.1	Platzkomplexität . . . . .	55
5.2	Approximationsalgorithmen: . . . . .	59

## 0.1 Einführung

Theoretische Informatik ist die Untersuchung der in der Informatik auftretenden Begriffe, Strukturen, Probleme mit mathematischen Methoden.

1. Naturgesetzliche Bedingungen in der Informationsverarbeitung → BuK
2. Standardmodelle und -verfahren für die Systemkonstruktion → ATFS

Zu 1) Erste Aufgabe: Präzisierung von „algorithmisches Problem“, „Algorithmus“

## 0.2 Gliederung der Vorlesung:

- I. Berechenbarkeit und Turingmaschinen
- II. Registermaschinen und WHILE-Programme
- III. Unentscheidbarkeit
- IV. Grundlagen der Komplexitätstheorie
- V. Ausblick in die Algorithmik für schwere Probleme

# 1 Berechenbarkeit und Turingmaschinen

## 1.1 Beispiele algorithmischer Probleme

**Bsp. 1:** Größter gemeinsamer Teiler.

Gegeben zwei natürliche Zahlen  $\geq 1$ , bestimme ihren größten gemeinsamen Teiler.

$(10,14) \mapsto 2$   
 $(10,15) \mapsto 5$   
 $(10,13) \mapsto 1$

Berechnung durch den Euklidischen Algorithmus (EA)

```
x,y := x1,x2
while x != y
  if x < y then y := y-x;
  else x := x-y
z:=x; //{z=ggT(x1,x2)}
```

**Bsp.2:** Primzahltest

Gegeben eine nat. Zahl  $\geq 2$ , entscheide, ob sie Primzahl ist.

**Bsp.3:** Primfaktorzerlegung

Gegeben eine natürliche Zahl  $\geq 2$ , berechne ihre Primfaktorzerlegung.

$120 \mapsto 2 \cdot 2 \cdot 2 \cdot 3 \cdot 5 = 2^3 \cdot 3 \cdot 5$

**Bsp.4:** 10. Hilbertsches Problem

Gegeben ein Polynom  $p(x_1, x_2, \dots, x_n)$  mit ganzzahligen Koeffizienten.

Gibt es eine Nullstelle  $(z_1, z_2, \dots, z_n) \in \mathbb{N}$

$3x_1^3 + x_2 - x_1^2 x_2^2 \mapsto 1$  ('ja') Nehme  $(z_1, z_2) = 0$

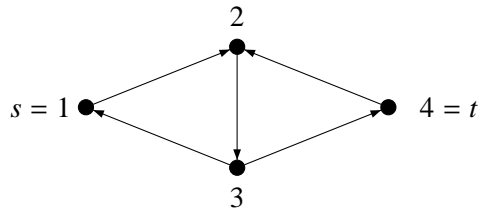
$x_1^2 + 1 \mapsto 0$  ('nein')

**Bsp.5:** Terminierung

Gegeben ein Java-Programm mit integer-Variablen, entscheide ob es bei Initialisierung mit 0 terminiert.

**Bsp.6** : Erreichbarkeitsproblem für gerichtete Graphen

Gegeben endlicher gerichteter Graph  $G$  und zwei Knoten  $s, t$ , entscheide ob Pfad in  $G$  von  $s$  nach  $t$  existiert.



**Bsp.7:** Traveling Salesman Problem (TSP)

Gegeben ein in  $\mathbb{N}_+$  gewichteter ungerichteter Graph  $G$ , bestimme eine kürzeste Rundtour durch  $G$ .

**Diskussion:** Zwei Typen von Problemen:

Entscheidungsprobleme und allgemein Berechnungsprobleme

Allgemeine Fassung von 'Problem': Beschreibung eines Zusammenhangs zwischen Eingabedaten und Ergebnisdaten.

Abstrakt: Vorgabe einer Funktion  $f$ : Eingabebereich  $\mapsto$  Ergebnisbereich

- Bsp.1:  $f : (\mathbb{N} \setminus \{0\})^2 \mapsto \mathbb{N}$
- Bsp.2:  $f : (\mathbb{N} \setminus \{0, 1\}) \mapsto \{0, 1\}$

Universelle Kodierung von Daten: Wörter über geeigneten Alphabeten.

Zeichenvorrat: Alphabet (endliche nichtleere Menge von Symbolen)

**Bsp.:**

$$\Sigma_{bool} = \{0, 1\}$$

$$\Sigma_{dezimal} = \{0, \dots, 9\}$$

$$\Sigma_{tastatur} = \{0, \dots, 9, a, b, \dots, !, @\}$$

$$\Sigma_m = \{\underline{1}, \dots, \underline{m}\}$$

$$\Sigma^* = \text{Menge der Wörter (endl. Zeichenreihen) über } \Sigma$$

Kodierung von Eingabe und Ausgabe durch Wörter

**Bsp.1:** ggT Eingabebereich:  $\{0, \dots, 9\}^* \times \{0, \dots, 9\}^*$  (auch Eingaben zugelassen mit führender Null, 0, leeres Wort). Reduktion auf ein Eingabewort durch Übergang zu  $\{0, \dots, 9, \#\}$   
Eingabe 10#14 statt (10,14)

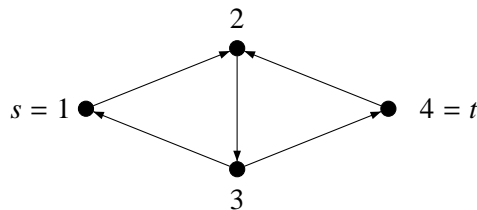
**Bsp.2,Bsp.3:** Analog

**Bsp.4:**  $3x_1^3 - 2x_1^2 + 3x_1x_2$   
 $\Sigma = \{x, 0, \dots, 9, +, -, \wedge, \_ \}$   
 $3x\_1\wedge 3-2x\_1\wedge 2+3x\_1x\_2$

15.10.04

Gesucht ist eine Darstellung endlicher gerichteter Graphen durch Wörter

**Bsp.:**



Beispielkodierung: 4(1-2)(2-3)(3-1)(3-4)(4-2)(1)(4)

Wort allg. : Knotenzahl dezimal, Folge der Kanten( $n_1 - n_2$ ) mit  $n_1, n_2$  dezimal,(S dezimal),(T dezimal)

Alphabet:  $\Sigma = \{0, \dots, 9, -, (, )\}$

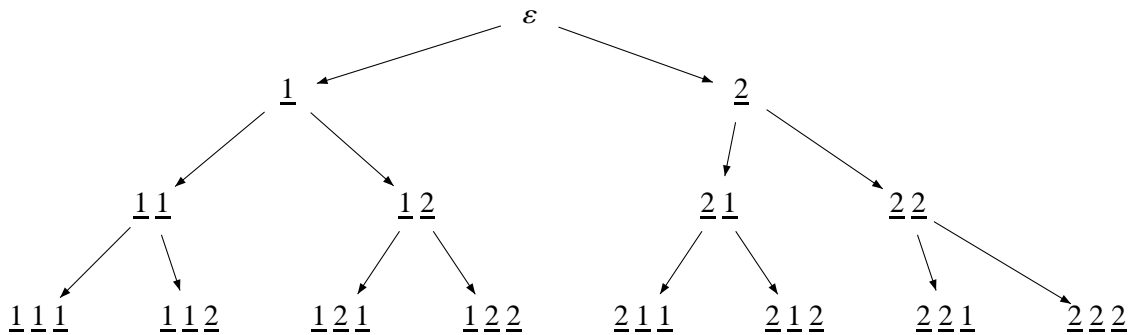
**Ergebnis:** Ein algorithmisches Problem ist gegeben durch die Definition einer Wortfunktion  $f : \Sigma^* \rightarrow \Gamma^*$  mit  $\Sigma, \Gamma$  Alphabete.

Lösung: Ist ein „Algorithmus“ der  $f$  berechnet.

## 1.2 Entscheidungsprobleme

**Ein Entscheidungsproblem** ist geg. durch die Def. einer Funktion  $f : \Sigma^* \rightarrow \{0, 1\}$  ( $\Sigma$  Alphabet).  $f$  kann als charakteristische Funktion der Wortmenge  $L : \{w \in \Sigma^* \mid f(w) = 1\}$  aufgefasst werden. Wir nennen Mengen  $L \subseteq \Sigma^*$  Sprache.

**Korrespondenz Wörter-Zahlen**  $\Sigma_m = \{\underline{1}, \dots, \underline{m}\}$  Wir ordnen die Wörter aus  $\Sigma_m^*$  in der kanonischen Reihenfolge, erst nach Länge, dann wie im Lexikon. Das 0-te Wort ist das leere Wort  $\varepsilon$   
 $\varepsilon, \underline{1}, \underline{2}, \dots, \underline{m}, \underline{1}\underline{1}, \underline{1}\underline{2}, \dots, \underline{1}\underline{m}, \underline{2}\underline{1}, \dots, \underline{m}\underline{m}, \underline{1}\underline{1}\underline{1}, \dots$



Baum aller Wörter über  $\Sigma_2$  : ordne stufenweise von links nach rechts

**Definition 1** Für  $m > 0$  sei  $\delta_m : \mathbb{N} \rightarrow \Sigma_m^*$ ,  $\delta_m(i)$  = das  $i$ -te Wort in der kanonischen Reihenfolge.

**Beispiel:**

$$\delta_2(4) = \underline{1}\underline{2} \quad \delta_m(0) = \varepsilon \quad \forall m$$

**Definition 2** Für  $m > 0$  sei  $\gamma_m : \Sigma_m^* \rightarrow \mathbb{N}$  die Umkehrfunktion von  $\delta_m$ , d.h. für  $w \in \Sigma_m^*$  ist  $\gamma_m(w)$  das  $i$  mit  $\delta_m(i) = w$

**Berechnung von  $\gamma_m$**   $\gamma_m(\varepsilon) = 0$

$$\gamma_m(\underline{k}_r \underline{k}_{r-1} \dots \underline{k}_0) := k_0 + k_1 \cdot m + \dots + k_r \cdot m^r$$

**Beispiel:**

$$\gamma_2(\underline{2}\underline{1}) = 1 + 2 \cdot 2 = 5$$

$$\gamma_2(\underline{1}\underline{1}\underline{1}) = 1 + 1 \cdot 2 + 1 \cdot 2^2 = 7$$

**Berechnung von  $\delta_m$ :** durch iterierte Division mit positivem ( $> 0$ ) Rest

**Beispiel:**

$\delta_4(43)$  :

$$43 = 10 \cdot 4 + 3$$

↙

$$10 = 2 \cdot 4 + 2$$

↙

$$2 = 0 \cdot 4 + 2$$

$$\delta_4(43) = \underline{2}\underline{2}\underline{3}$$

$\delta_4(99)$

$$99 = 24 \cdot 4 + 3$$

↙

$$24 = 5 \cdot 4 + 4$$

↙

$$5 = 1 \cdot 4 + 1$$

↙

$$1 = 0 \cdot 4 + 1$$

$$\delta_4(99) = \underline{1} \underline{1} \underline{4} \underline{3}$$

**Ergebnis:** Wörter über  $\Sigma_m$  kann man in der kanonischen Reihenfolge numerieren. Damit lassen sich Eingaben zu algorithmischen Problemen auch als natürliche Zahlen darstellen.

19.10.04

**Definition 3** ein Algorithmisches Problem wird erfasst durch die Definition einer Wortfunktion

$$f : \underbrace{\Sigma^*}_{\text{Bereich der Eingaben}} \rightarrow \underbrace{\Sigma^*}_{\text{Bereich der Ausgaben}}$$

**Lösung** eines solchen Problems durch einen Algorithmus, berechnet  $f$ , falls er, gestartet mit  $w \in \Sigma^*$  als Eingabe, nach endlich vielen Schritten terminiert und  $f(w)$  als Ausgabe liefert.

### 1.3 Algorithmen und Turingmaschinen

**Ein Algorithmus** ist ein Verfahren, dass

- Eingabewörter schrittweise bearbeitet
- durch einen endlichen Text bis ins letzte Detail eindeutig festgelegt ist.
- bei Termination eine Ausgabe liefert, oder nicht terminiert.

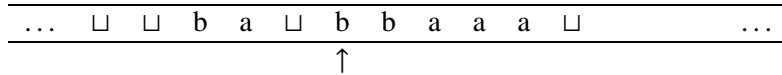
**Präzisierung** durch Alen Turing(1912-1954) mit der Turingmaschine (1936).

Computing is normally done by writing certain symbols on paper. We may suppose this paper is divided into squares like a child's arithmetic book. In elementary arithmetic the two-dimensional character of the paper is sometimes used. But such a use is always avoidable, and I think that it will be agreed that the two-dimensional character of paper is no essential of computation. I assume that the computation is carried out on one-dimensional paper, i.e., on a tape divided into squares.

The behavior of the computer at any moment is determined by the symbols which he is observing, and his state of mind at that moment. We may suppose that there is a bound  $B$  to the number of symbols or squares which the computer can observe at one moment. If he wishes to observe more, he must use successive observations. We will also suppose that the number of states of mind which need be taken into account is finite.

The reasons for this are of the same character as those which restrict the number of symbols. If we admitted an infinity of states of mind, some of them will be arbitrarily close and will be confused. Again this restriction is not one which seriously affects computation, since the use of more complicated states of mind can be avoided by writing more symbols on the tape.

Alen Turing über Turingmaschinen.



**Definition 4** Eine Turingmaschine (TM) hat die Form  $M = (Q, \Sigma, \Gamma, \delta, q_0, q_s)$  mit

- endlicher Zustandsmenge  $Q$
- Anfangszustand  $q_0$
- Stopzustand  $q_s$
- Eingabealphabet  $\Sigma$
- Arbeitsalphabet  $\Gamma \supseteq \Sigma \cup \{\sqcup\}$  „blank“
- Übergangsfunktion  $\delta : Q \setminus \{q_s\} \times \Gamma \rightarrow \Gamma \times \{R, L, N\} \times Q$

$\delta(p, a) = (a', R/L/N, q)$ : Im Zustand  $p$  mit  $a$  auf Arbeitsfeld drucke  $a'$ , gehe ein Feld nach  $R/L$  oder nicht und nehme Zustand  $q$  an.

**Ein Konfigurationswort** hat die Form  $u q v$  mit  $u \in \Sigma^*, q \in Q, v \in \Gamma^*$

Es steht für die Konfiguration  $\dots, \sqcup \sqcup u v \sqcup \sqcup \sqcup \dots$

$u'q'v'$  ist Folgekonfigurationswort von  $uqv$ .  $u q v \vdash_M u'q'v'$  falls einer der folgenden Fälle auftritt:

- $\delta(q, a) = (a', N, q')$ ,  $v = av_0$ ,  $u' = u$ ,  $v' = a'v_0$  (Identifiziere  $u = \varepsilon$  mit  $u = \sqcup$ )
- $\delta(q, a) = (a', L, q')$ ,  $u = u_0b$ ,  $v = av_0$ ,  $u' = u_0$ ,  $v' = ba'v_0$  (Identifiziere  $u = \varepsilon$  mit  $u = \sqcup$ ,  $v$  analog)

Was passiert wenn  $u = \varepsilon$

$$\delta(q, a) = (a' L q') \quad \underbrace{\sqcup}_{u=\varepsilon} q \underbrace{ab} \vdash q' \sqcup a'b$$

- $\delta(q, a) = (a', R, q')$  analog

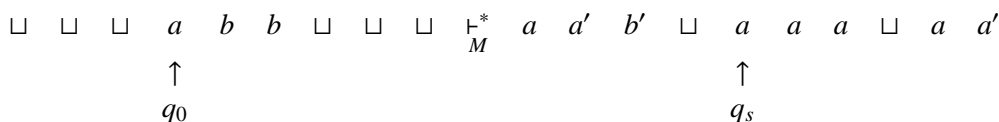
$K \vdash K'$  wie oben definiert für Konfigurationswörter  $KK'$

$K \vdash_M^* K'$  es ex. eine Folge  $K_0, \dots, K_r$  mit  $K_0 = K$ ,  $K_r = K'$ ,  $K_i \vdash K_{i+1}$  ( $i < r$ )

$K = uuqv$  heißt Stopkonfiguration falls  $q = q_s$

Ergebnis der Stopkonfiguration  $uq_s v$  ist das längste Anfangsstück von  $v$  ohne  $\sqcup$

$M$  berechnet  $f : \Sigma^* \rightarrow \Sigma^*$ , falls (für jedes  $w \in \Sigma^*$ )  $M$  von der Startkonfiguration  $q_0 w$  aus nach endlichen Schritten eine Stopkonfiguration erreicht und zwar mit Ergebnis  $f(w)$



$$q_0 \underbrace{abb}_w \rightarrow f(abb) = (aaa)$$

Ergebnis wäre:  $\underbrace{aaa}_{f(w)}$

Erweiterung des Funktionsbegriffs zu partiellen Funktionen  $f : \Sigma^* \Sigma^* \rightarrow \Sigma^* \cup \{\perp\}$  deutet Funktion an mit Definitionsbereich  $Def(f) \subseteq \Sigma^* \Sigma^*$ . Für  $w \in \Sigma^* \Sigma^* \setminus Def(f)$  schreibe  $f(w) = \perp$  (undefiniert)

TM  $M$  berechnet  $f : \Sigma^* \Sigma^* \rightarrow \Sigma^* \cup \{\perp\}$  falls von Startkonfiguration  $q_0 w$  eine Stopkonfiguration erreicht wird g.d.w.  $w \in Def(f)$ , und Ergebnis ist dann  $f(w)$ , ansonsten ( $w \notin Def(f)$ ) soll  $M$  nicht terminieren.

22.10.04

**Turingmaschinen** Übergangsfunktion  $\delta$

$\delta(p, a) = (a', L/N/R, q)$   
 Turingzeile:  $p \ a \ a' \ L/N/R \ q$  } In  $p$  mit  $a$  auf AF, drucke dort  $a'$  bewege Kopf ein Feld nach links/nicht/ein Feld nach rechts und gehe in Zustand  $q$ .

$M^*$  berechnet:  $\Sigma^* \Sigma^* \rightarrow \Sigma^* \cup \{\perp\}$  angesetzt auf  $w \in \Sigma^* \Sigma^*$  terminiert g.d.w.  $w \in Def(f)$  und in diesem Fall ist das Ergebnis der Stopkonfiguration das Wort  $f(w)$ .

**Beispiel:**

$\Sigma_{bool} = \{0, 1\}$   $F : \Sigma_{bool}^* \Sigma_{bool}^* \rightarrow \Sigma_{bool}^* \cup \{\perp\}$  sei definiert durch  $f(w) = w0$ .

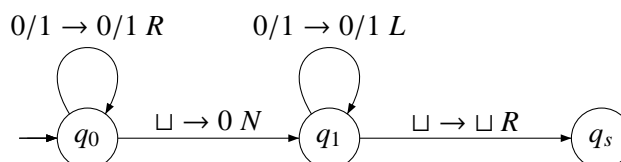
Gewünscht	Formel	Ausgabe von Turingzeilen
0 0 1 1	$q_0 w \vdash^* q_s w 0$	$q_0 \ 0/1 \ 0/1 \ R \ q_0$
$q_0$		$q_0 \sqcup \ 0N \ q_1$
$\rightsquigarrow$		$q_1 \ 0/10/1L \ q_1$
0 0 1 1 0 $\sqcup$		$q_1 \sqcup \sqcup R \ q_s$
$q_s$		
		Turingtafel für M

M-Berechnung:  $q_0 \ 0 \ 0 \ 1 \ 1 \vdash^* \ 0 \ 0 \ 1 \ 1 \ q_0 \sqcup \vdash \ 0 \ 0 \ 1 \ 1 \ q_0 \ 0 \vdash^* \ q_1 \sqcup \ 0 \ 0 \ 1 \ 1 \ 0 \vdash \ q_s \ 0 \ 0 \ 1 \ 1 \ 0$

Konvention: Fehlt für ein Paar  $(q,a)$  die entsprechende Zeile  $q \ a \dots$

denken wir uns  $q \ a \ a \ N \ q_s$  ergänzt.

Graphische Darstellung:



**Konvention** zur Berechnung mehrstelliger Funktionen  $f : \underbrace{\Sigma^* \times \dots \times \Sigma^*}_{n\text{-mal}} \rightarrow \Sigma^*$

Zur Bestimmung von  $f(w_1, \dots, w_r)$  wird TM gestartet in Konfiguration  $q_0 w_1 \sqcup w_2 \sqcup \dots \sqcup w_r$

**Beispiel:**

„Unäre Multiplikation“  $f : \{\}^* \times \{\}^* \rightarrow \{\}^*$

$q_0 |^n \sqcup |^m \rightsquigarrow \dots q_s |^{n \cdot m} \sqcup \dots$

**Illustration** für  $q_0 ||| \sqcup |||$

**Idee:** Kopiere 2. Block so oft hinter den Input wie es Striche im 1. Block gibt. Hierzu führe ein neues Symbol ein:  $\#$ .

Typische Zwischenkonfiguration:

$\# \# q | \sqcup ||| \sqcup |||||$

**Details:**

- (1) ( $q_1$ ) Falls  $|$  auf AF, drucke  $x$ , weiter bei (2)  
Sonst (AF= $\sqcup$ ) gehe (einen weiteren Schritt nach rechts, stoppe) zum nächsten  $\sqcup$  nach rechts
- (2) ( $q_2$ ) Gehe auf Feld nach nächstem  $\sqcup$  und (4) Kopiere den dort beginnenden Strichblock an das Ende des darauf folgenden Strichblocks.
- (3) gehe zurück auf letztes Feld mit  $\#$ , ein Feld nach rechts und zurück zu (1)
- (4)  $\# \# | \sqcup p ||| \sqcup |||||$ 
  - (a) Falls AF =  $\#$  überschreibe durch  $\#$  weiter bei (b)  
Sonst AF =  $\sqcup$  verwandle nach links gehend die  $\#$  in  $|$  zurück bis  $\sqcup$ , zu (3)
  - (b) Gehe nach rechts auf zweites  $\sqcup$  und überschreibe durch  $\#$
  - (c) Gehe zurück auf erstes  $\#$  einen Schritt nach rechts, weiter bei (a)

$q_1$		$\#$	$N$	$q_2$
$q_1$	$\sqcup$	$\sqcup$	$R$	$q'_1$
$q'_1$			$R$	$q'_1$
$q'_1$	$\sqcup$	$\sqcup$	$R$	$q_s$

26.10.04

## 1.4 Berechenbarkeit und nicht-Berechenbarkeit

Turingmaschine } berechnet eine Funktion  $f : \Sigma^* \rightarrow \Sigma^*$ , falls mit Eingabe  $w \in \Sigma^*$   
 Algorithmus }  
 TM }  
 Algorithmus } terminiert gdw.  $w \in Def(f)$  und in diesem Fall das Ergebnis das Wort  $f(w)$  ist.

$f$  heißt Turing-berechenbar  
berechenbar im intuitiven Sinne } gdw. es gibt Turingmaschine  
Algorithmus } die/der die Funktion  $f$  be-  
rechnet.

$f$  berechenbar  $\Leftrightarrow f$  Turing-Berechenbar?

$\Leftarrow$  klar, denn eine TM ist eine (sogar spezielle) Form von Algorithmus

$\Rightarrow$  nicht mathematisch nachweisbar.

### 1.4.1 Church-Turing-These (1936)

Jede berechenbare Funktion ist auch Turing-berechenbar.  
Argumente dafür:

- i) Turings Analyse des Rechenvorgangs
- ii) Erfahrung seit über 60 Jahren
- iii) Viele verschiedene Zugänge zum Berechenbarkeitsbegriff sind formal äquivalent (z.B.  $\lambda$ -Kalkül und TM)
- iv) die Programmiersprachlichen Fassungen des Algorithmus lassen sich auf Turingmaschinen reduzieren

### Verwendung der Church-Turing-These

1. Unwesentlich: beim Nachweis der Turing-Berechenbarkeit nach Vorlage eines Algorithmus. (durch Fleißarbeit vermeidbar)
2. wesentlich: Nachweis, dass eine Funktion nicht berechenbar ist (durch einen Algorithmus im Intuitiven Sinn). Es reicht: Funktion ist nicht Turing-berechenbar.

Erstes Beispiel einer nicht berechenbaren Funktion, die Busy-Beaver-Funktion  $BB : \mathbb{N} \rightarrow \mathbb{N}$  (Rado 1962)  
Betrachte TM mit Arbeitsalphabet  $\{ \sqcup, \sqcup \}$  nur mit L-/R-Bewegung wobei in den Zustandszahlen  $q_s$  nicht gezählt wird.

**Biber** ist TM die, auf das leere Band angesetzt, terminiert.

**Bemerkung:** Zu fester Zustandstzahl existieren nur endl. viele TM (Biber).



Für  $n > k$  gilt dann:  $F(2n + 1) \leq F(2n + 2) = G(n)$   
 = Ergebnis von  $M_G$  bei Eingabe von  $n$  Strichen  
 = Ergebnis von  $M_n$  bei Start auf leerem Band  
 $\leq BB(n + 1 + k)$  (da  $M_n$   $n+1+k$  Zustände hat)  
 $BB(2n) < BB(2n + 1) < BB(2n + 2)$

## 1.5 Berechenbarkeit, Entscheidbarkeit, Aufzählbarkeit

Ausgangspunkt: Def. von „Algorithmus A berechnet F“

Def.: Algorithmus A  $\begin{cases} \text{entscheidet} \\ \text{semi-entscheidet} \end{cases}$  die Sprache  $L \subseteq \Sigma^*$  gdw. bei Eingabe  $w \in \Sigma^*$

$\begin{cases} \text{terminiert A immer, mit Ausgabe „ja“ falls } w \in L; \text{ „nein“ falls } w \notin L \\ \text{terminiert A mit Ausgabe „ja“ für } w \in L; \text{ terminiert nicht für } w \notin L \end{cases}$

**Definition 5**  $L$  *entscheidbar (semi-entscheidbar)* gdw. es existiert ein Algorithmus, der  $L$  *entscheidet (semi-entscheidet)*

### Beispiel (10. Hilbertsches Problem):

$L$  = Menge der Kodierungen von Polynomen über  $\mathbb{Z}$ , welche eine Nullstelle über  $\mathbb{Z}$  haben.

**Semi-Entscheidungsverfahren** Zu Polynom  $p(x_1, \dots, x_n)$  teste sukzessiv für  $k = 1, 2, 3, \dots$  ob jeweils eine Nullstelle  $(z_1, \dots, z_n)$  mit  $|z_i| \leq k$  existiert.

Beachte: für festes  $k$  existieren nur endlich viele Kandidaten  $(z_1, \dots, z_n)$

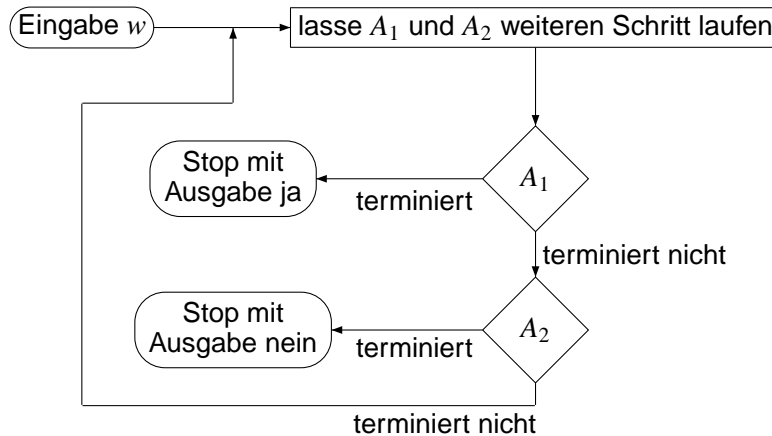
Ziel: Zusammenhänge zwischen „Berechenbar“ und „(semi-)entscheidbar“

**Satz 1.2:**  $L \subseteq \Sigma^*$  *entscheidbar*  $\Leftrightarrow L$  *semi-entscheidbar* und  $\Sigma^* \setminus L$  *ist auch semi-entscheidbar*

29.10.04

„ $\Rightarrow$ “ Sei  $A$  Entscheidungsalgorithmus für  $L$ . Bilde den Algorithmus  $A'$  durch Einbau einer nicht terminierenden Schleife für Termination von  $A$  mit „nein“  
 $A'$  *semi-entscheidet*  $L$   
 $\bar{A}$  entstehe aus  $A$  durch Vertauschen von „ja“/ „nein“  
 $\bar{A}'$  *semi-entscheidet*  $\Sigma^* \setminus L$

„ $\Leftarrow$ “ Seien  $A_1, A_2$  *semi-entscheidungsverfahren* für  $L$  bzw.  $\Sigma^* \setminus L$   
 Finde Entscheidungsverfahren  $A$  für  $L$   
 Beachte: Jedes  $w \in \Sigma^*$  gehört zu  $L$  oder  $\Sigma^* \setminus L$  d.h. mit Eingabe  $w \in \Sigma^*$  wird entweder  $A_1$  oder  $A_2$  terminieren.  
 Entscheidungsalgorithmus  $A$ :

**Definition**

Zu  $f : \Sigma^* \rightarrow \Sigma^*$  definiere  $G_f = \{w\#f(w) \mid w \in \text{Def}(f)\}$  „Graph von  $f$ “

**Satz 1.3:**  $f : \Sigma^* \rightarrow \Sigma^*$  berechenbar  $\Leftrightarrow G_f$  ist semi-entscheidbar.

**Beweis:**

„ $\Rightarrow$ “  $A_f$  berechnet  $f$  Gesucht: Semi-Entscheidungsalgorithmus  $B$  für  $G_f$

Zu Eingabe  $v$ : Test ob  $v$  die Form  $v = w_0\#w_1$  hat mit  $w_0, w_1 \in \Sigma^*$

- wenn nein: Stop mit Ausgabe „nein“.
- sonst: Eingabe  $w_0\#w_1$

Lasse  $A_f$  auf  $w_0$  laufen, speichere  $w_1$

Bei Termination von  $A_f$  (mit Ausgabe  $f(w_0)$ ) vergleiche  $f(w_0)$  mit  $w_1$

Bei Gleichheit terminiere mit „ja“

Ansonsten gehe in Endlosschleife

„ $\Leftarrow$ “ Sei  $A_{G_f}$  ein Semi-Entscheidungsalgorithmus für  $G_f$

Finde Berechnungsverfahren für  $f$

Eingabe  $w \quad w\#\varepsilon?$

$w\#0?$

$w\#1$

Lasse  $A_{G_f}$  laufen

- für alle  $v$  auf der Eingabe  $w\#v$
- für alle Schrittzahlen  $s$  nach folgendem Muster:  
Bei Termination von  $A_G$  für  $w\#v$  gebe  $f_v$  aus.

**Satz 1.4:**  $L \subseteq \Sigma^*$  semi-entscheidbar  $\Leftrightarrow L$  ist Definitionsbereich einer berechenbaren Funktion  $f : \Sigma^* \rightarrow \Sigma^*$

„ $\Rightarrow$ “ A semi-entscheidet  $L \subseteq \Sigma^*$

Sei  $a \in \Sigma$ . Definiere:  $f : \Sigma^* \rightarrow \Sigma^*$  durch  $f(w) = \begin{cases} a & w \in L \\ \perp & w \notin L \end{cases}$

Gilt  $L = Def(f)$

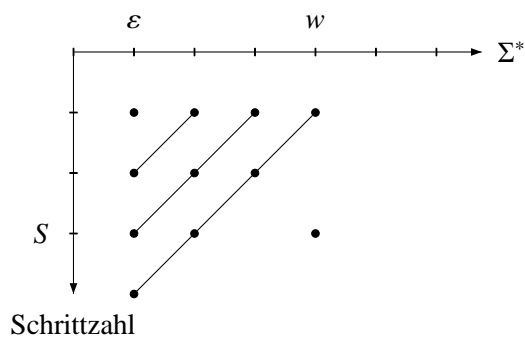
Folgender Algorithmus B berechnet f:

Für Eingabe w arbeite wie A, allerdings, bei Termination gebe a aus.

„ $\Leftarrow$ “ A berechnet f L sei  $Def(f)$

Gesucht: Semi-Entscheidungsalgorithmus B für L.

Für B übernehme A, wobei die Ausgabe jeweils geändert ist zu „ja“



Ein Aufzählungsalgorithmus ist ein Verfahren, das ohne Eingabe gestartet wird und sukzessiv Ausgabe-wörter liefert (in irgend einer Reihenfolge, eventuell mit Wiederholungen).

$L \subseteq \Sigma^*$  heißt aufzählbar, wenn es einen Aufzählungsalgorithmus gibt, für den gilt:

Für  $w \in \Sigma : w \in L \Leftrightarrow w$  wird irgendwann als Ausgabe geliefert.

2.11.04

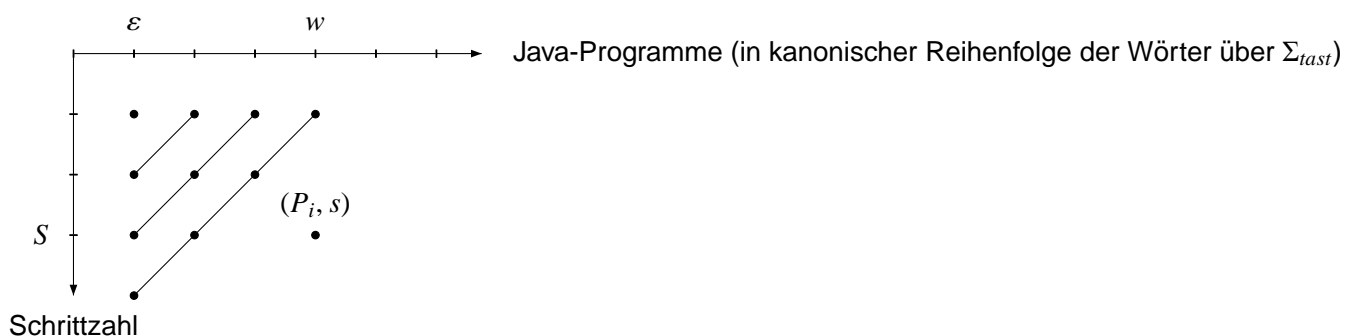
**Beispiel (zu semi-entscheidbar - aufzählbar):**

1. Die Menge der Java-Programme die mit 0 initialisiert terminieren ist semi-entscheidbar (Eingabe: ein Wort über  $\Sigma_{tastatur}$ )

Semi-Entscheidungsverfahren:

- Zu Eingabe w überprüfe, ob w Java-Programm ist. Wenn nein: Endlosschleife
- Sonst: lasse das Programm „w“laufen mit 0 initialisiert und terminiere, wenn es terminiert

2. Die Menge dieser Java-Programme ist aufzählbar. Aufzählungsverfahren geht gemäß folgendem Diagramm vor:



An stelle  $(P_i, s)$  lasse  $P_i$  für s Schritte laufen und gebe  $P_i$  aus, wenn  $P_i$  dabei stoppt.

**Dann:**  $P$  wird ausgegeben  $\Leftrightarrow$  ex. Schrittzahl  $s$  so dass  $P$  in  $\leq s$  Schritten terminiert  
 $P$ terminiert

**Bemerkung:**

„berechenbar“ verlangt nur Existenz eines Algorithmus.

**GV (Goldbach-Vermutung):** Jede gerade Zahl  $\geq 4$  ist Summe zweier Primzahlen.

**Definition**

$$f : \mathbb{N} \rightarrow \mathbb{N} \quad f(n) = \begin{cases} 1 & \text{GVwahr} \\ 0 & \text{GVfalsch} \end{cases}$$

$f \equiv 0$  oder  $f \equiv 1$

Alg  $A_0$  oder Alg.  $A_1$

Es existiert ein Algorithmus des  $f$  berechnet!

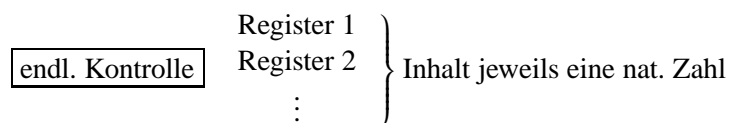
## 2 Registermaschinen und while-Programme

**Idee** „berechenbar“  $\sim$  „programmierbar“

Grundbereich hier:  $\mathbb{N}$  (Vergleich zu Turingmaschine über  $\Sigma = \{\}\ n \sim \underbrace{|\dots|}_n$ )

### 2.1 goto-Programme

**Rechnermodell:** Registermaschine



$X_i$  Variable (Name für Speicher von Register  $i$ )

#### 2.1.1 Definition:

Ein  $\text{goto}_m$ -Programm hat die Form:

1	$\alpha_1;$
2	$\alpha_2;$
3	$\alpha_3;$
	⋮
$k - 1$	$\alpha_{k-1};$
$k$	STOP

wobei  $\alpha_j =$

$j$	$X_i := X_i + 1$	$(i \in \{1, \dots, m\})$
oder	$j$	$X_i := X_i - 1$
oder	$j$	if $X_i = 0$ goto $l$ else goto $l'$ $(l, l' \in \{1, \dots, k\})$
		$(j \in \{1, \dots, k-1\})$

Konfiguration:  $(m+1)$ -Tupel  $(j, r_1, \dots, r_m)$  für Zeilennummer  $j$  und  $r_i$  als Wert von  $X_i$   
 Folgekonfiguration von  $(j, r_1, \dots, r_m)$  lautet:

- im Fall  $\alpha_j = X_i := X_i + 1 :$   
 $(j + 1, r_1, \dots, r_i + 1, \dots, r_m)$
- im Fall  $\alpha_j = X_i := X_i - 1 :$   
 $(j + 1, r_1, \dots, r_i - 1, \dots, r_m)$ 

$$\left[ r_i - 1 = \begin{cases} r_i - 1 & r_i > 0 \\ 0 & r_i = 0 \end{cases} \right]$$
- $\alpha_j = \text{if } X_i = 0 \text{ goto } l \text{ else goto } l' \begin{cases} (l, r_1, \dots, r_m) & \text{falls } r_i = 0 \\ (l', r_1, \dots, r_m) & r_i \neq 0 \end{cases}$

**Definition** zu  $\text{goto}_m$ -Programm  $P$  und  $n \leq m$  definiere die  $n$ -stellige durch  $P$  berechnete Funktion  $f_P^{(n)} : \mathbb{N}^n \rightarrow \mathbb{N}^m$  durch  $f_P^{(n)}(x_1, \dots, x_n)$  def. gdw.  $P$  ausgehend von Konfiguration  $(1, x_1, \dots, x_n, 0, \dots, 0)$  erreicht Stopkonf.  $(k, y_1, \dots, y_m)$  und in diesem Fall  $f_P^{(n)}(x_1, \dots, x_n) = y_1$   
 $f : \mathbb{N}^n \rightarrow \mathbb{N}^m$  heißt  $\text{goto}_m$ -berechenbar, wenn es ein  $\text{goto}_m$ -Programm  $P$  gibt mit  $f_P^n = f$  goto-berechenbar, falls  $\text{goto}_m$ -berechenbar für geeignetes  $m$ .

**Bsp.:**

**P1:**

1. STOP

$$f_{P1}^{(1)} = id_{\mathbb{N}} \quad f_{P1}^{(1)}(x) = x$$

$$\|P_i^{(n)}(x_1, \dots, x_n) = x_i\| \quad f_{P1}^{(2)}(x_1, x_2) = x_1 \quad f_{P1}^{(n)} = \text{Projektion auf 1. Komponente}$$

**P2:**

1. if  $X_1 = 0$  goto 1 else goto 2
2. STOP

$$f_{P2}^{(n)} = \begin{cases} \perp & \text{falls } X_1 = 0 \\ X_1 & \text{falls } X_1 \neq 0 \end{cases}$$

Konvention:  $j \text{ goto } l$  für  $j \text{ if } X_i = 0 \text{ goto } l \text{ else goto } l$

**P3:**

1. if  $X_2 = 0$  goto 5 else goto 2
2.  $X_1 := X_1 + 1$

- 3.  $X2 := X2 - 1$
- 4. goto 1
- 5. STOP

Tabelle 2.1: Rechnung

Zeile:	X1	X2
1	1	2
2	1	2
3	2	2
4	2	1
1	2	1
	⋮	
4	3	0
1	3	9
5	3	0

$(1, X1, X2) \rightsquigarrow (5, X1+X2, 0)$

Also  $f_{P3}^{(2)}$  = Additionsfunktion +  
 $f_{P3}^{(1)} = X1$  also  $f_{P3}^{(1)} = id_{\mathbb{N}}$

Multiplikationsfunktion: analog;

Idee:

$X3 := X1; X2 = 0$ ; solange  $X2 \neq 0$

Kopiere  $X3$  nach  $X1$  und  $X4$ ;

dann  $X3=0$ ;

$X2 := X2 - 1$ ;

Kopiere  $X4$  nach  $X3$  (dann  $X4 = 0$ )

X1	X2	X3	X4
----	----	----	----

3	4		
0	4	3	
3	3	0	3
6	2	0	3

## 2.2 while-Programme

### 2.2.1 Idee:

Rechnermodell wie bei goto.

Komfortablere Kontrollstrukturen (Verzweigung, Schleifen).

**Illustration:** ggT-Algorithmus:

```
while X1 ≠ X2 do
  if X1 < X2 then X2 := X2 ÷ X1
  else X1 := X1 ÷ X2
```

**Bsp.:** while<sub>2</sub>- Programm.

Tests  $X_1 > 0$ ,  $X_i = 0$  genügen

Benutze Anweisungen	$X_3 := X_2 \dot{-} X_1$ $X_4 := X_2 \dot{-} X_1$ $X_5 := X_1 \dot{-} X_2$ $X_4 := X_4 + X_3$	(Test $X_3 > 0$ )   (Test $X_4 > 0$ statt $X_1 \neq X_2$ )
---------------------	--	---

## 2.2.2 Definition der Programmiersprache while

Syntax: Gestalt der Programme als Texte (Wörter).

Semantik: Definition der Wirkung der Programme auf vorgelegte Daten.

while<sub>m</sub>-Programme benutzen nur die Variablen  $X_1, X_2, \dots, X_m$

## 2.2.3 while<sub>m</sub>-Programme: Vorbereitung

$\langle var_m \rangle$	Variablen $X_1, \dots, X_m$
$\langle WZW_m \rangle$	$X_i := 0$ $X_i := X_j$ $X_i := X_i + 1$ $X_i := X_i \dot{-} 1$ $X_i := X_j \text{ op } X_k$ op $\in \{+, \dot{-}, \cdot, \div, \text{ mod } \}$ mit $1 \leq i, j, k \leq m$
$\langle bed_m \rangle$	$X_i > 0$ $X_i = 0$ mit $1 \leq i \leq m$

## 2.2.4 while<sub>m</sub>-Programme: Syntax

```
< progrm > ::= < WZWm > | < progrm >; < progrm > |
if < bedm > then < progrm > else < progrm > end
while < bedm > do < progrm > end |
loop < varm > begin < progrm > end
```

## Zur Semantik von while<sub>m</sub>-Programmen:

Definiere zu while<sub>m</sub>-Programmen  $P$  eine Transformation  $\llbracket P \rrbracket : \mathbb{N}^m \rightarrow \mathbb{N}^m$

$\llbracket P \rrbracket(r_1, \dots, r_m) = (s_1, \dots, s_m)$  falls  $P$ , gestartet mit  $(r_1, \dots, r_m)$  als Werten für  $X_1, \dots, X_m$  terminiert und dann als Werte von  $X_1, \dots, X_m$  die Zahlen  $s_1, \dots, s_m$  liefert

**Bsp.:**  $P = X1 := X2 \quad \llbracket P \rrbracket(3, 5) = (5, 5)$

**Notation:** für  $f : M \rightarrow M$  definiere  $f^n : M \rightarrow M$  durch  $f^0 = x \quad f^{n+1} = f(f^n(x))$

### 2.2.5 while<sub>m</sub>-Programme: Semantik

Es sei  $\llbracket P \rrbracket(r_1, \dots, r_m)$  wie folgt definiert:

- für  $P = X_i := 0 \quad (r_1, \dots, \underbrace{0}_i, \dots, r_m)$
- $P = X_i := X_j \quad (r_1, \dots, \underbrace{r_j}_i, \dots, \underbrace{r_j}_j, \dots, r_m)$
- $P = X_i := X_i + 1 \quad (r_1, \dots, r_i + 1, \dots, r_m)$
- $P = X_i := X_i - 1 \quad \text{analog}$
- $P = X_i := X_j \text{ op } X_k \quad (r_1, \dots, r_j \text{ op } r_k, \dots, r_m)$
- Konvention  $r \{ \text{div} / \text{mod} \} 0 = 0$
- für  $P = P_1; P_2 \quad \llbracket P_2 \rrbracket(\llbracket P_1 \rrbracket(r_1, \dots, r_m))$
- für  $P = \text{if } X_i \{> / =\} 0 \quad \text{then } P_1 \text{ else } P_2 \text{ end}$   
 $\llbracket P_1 \rrbracket(r_1, \dots, r_m) \quad \text{falls } r_i \{> / =\} 0$   
 $\llbracket P_2 \rrbracket(r_1, \dots, r_m) \quad \text{sonst}$
- für  $P = \text{while } X_i \{> / =\} 0 \quad \text{do } P_1 \text{ end}$   
 $\llbracket P_1 \rrbracket^k(r_1, \dots, r_m) \quad \text{für das kleinste } k \geq 0 \text{ mit } p_i^{(m)}(\llbracket P_1 \rrbracket^k(r_1, \dots, r_m)) \{ \leq / \neq \} 0$   
 $\perp \quad \text{sonst}$
- für  $P = \text{loop } X_i \text{ begin } P_1 \text{ end} \quad \llbracket P_1 \rrbracket_i^r(r_1, \dots, r_m)$

Zu P def. wie bei goto-Progr.

$$f_P^{(n)}(x_1, \dots, x_n) = p_1^m \llbracket P \rrbracket(x_1, \dots, x_n, 0, \dots, 0)^m$$

**Bsp.:** zu  $\llbracket P \rrbracket = \text{loop } \underbrace{X1 := X1 - 1}_{P_0} \text{ end}$

$$\llbracket P_0 \rrbracket^i(r_1, \dots, r_m) = (r_1 - i, r_2, \dots, r_m)$$

$$\llbracket P \rrbracket(r_1, \dots, r_m) = \llbracket P_0 \rrbracket^{r_1}(r_1, \dots, r_m) = (0, r_2, \dots, r_m)$$

9.11.04

#### while-programme

P:

```
if X2 = 0 then X1 := X1 + 1;
while X1 > 0 do X1 := X1 + 1; end [P1]
else loop X1 begin X1 := X1 - 1 end end [P2]
```

ist ein while<sub>2</sub> Programm.  $\llbracket P_1 \rrbracket(x1, x2) = \perp$

$$\llbracket P_2 \rrbracket(x1, x2) = (0, x2)$$

$$\llbracket P \rrbracket(x1, x2) = \begin{cases} \perp & x2 = 0 \\ (0, x2) & x2 \neq 0 \end{cases}$$

$$f_P^{(2)}(x_1, x_2) = \begin{cases} \perp & x_2 = 0 \\ 0 & x_2 \neq 0 \end{cases} \quad f_P^{(1)} = \perp$$

**Ziel:**

1. Reduktion auf die Wertzuweisungen  $X_i := X_i + 1$ ,  $X_i := X_i - 1$   
Test  $X_i = 0$
2. Vergleich loop-while
3. Äquivalenz TM - goto - while

**Zu 1:**  $X_i = 0$  ersetzbar durch loop  $X_i$  begin  $X_i := X_i - 1$  end

$X_i := X_j$  ersetzbar durch  $X_i := 0$ ; loop  $X_j$  begin  $X_i := X_i + 1$  end

$X_i := X_j + X_k$  ersetzbar durch:  $X_i := 0$ ; loop  $X_j$  begin  $X_i := X_i + 1$

loop  $X_k$  begin  $X_i := X_i + 1$

Analog für  $-$ ,  $*$ ,  $\div$ , mod (Übungen)

Test if  $X_i > 0$  then ... ist ersetzbar durch (mit Hilfsvariablen  $Y$ , initialisiert mit 0)

$Y := Y + 1$ ;  $Y := Y - X_i$ ; if  $Y = 0$  then ...;

while  $X_i > 0$  do ... ersetzbar durch  $Y := Y + 1$ ;  $Y := Y - X_i$

while  $Y = 0$ ;  $Y := Y + 1$ ;  $Y := Y - X_j$  end.

while<sup>0</sup>-Programme seien while Programme nur mit Wertzuweisungen  $X_i := X_i + / - 1$  und Bedingungen  $X_i = 0$

Ergebnis Zu jedem while<sub>m</sub>-Programm  $P$  existiert ein while<sub>m'</sub>-Programm  $P'$  für geeignetes  $m' \geq m$  mit  $\llbracket P \rrbracket(x_1, \dots, x_m) = (y_1, \dots, y_m) \Leftrightarrow \llbracket P' \rrbracket(\underbrace{x_1, \dots, x_m, 0, \dots, 0}_{m'}) = (y_1, \dots, y_m, 0, \dots, 0)$

## 2.2.6 Vergleich loop-while

**Bem. 1)** „loop-Schleifen terminieren“

Sei  $P$ : loop  $X_i$  begin  $P_0$  end (loop<sub>m</sub>-Programm)

Wenn  $\llbracket P_0 \rrbracket : \mathbb{N}^m \rightarrow \mathbb{N}^m$  total, dann auch  $\llbracket P \rrbracket : \mathbb{N}^m \rightarrow \mathbb{N}^m$  total.

**Beweis:**  $\llbracket P \rrbracket(x_1, \dots, x_m) = \llbracket P_0 \rrbracket^{x_i}(x, 1, \dots, x_m)$  definiert.

**Bem. 2:** „while-Schleifen terminieren nicht immer“:

$P_\perp : X_1 := X_1 + 1$ ; while  $X_1 > 0$  do  $X_1 := X_1 + 1$  end

**Beispiel** zur Terminationsfrage: „ $3x+1$ -Funktion“

$$g(x) = \begin{cases} \frac{x}{2} & x \text{ gerade} \\ 3x + 1 & x \text{ ungerade} \end{cases}$$

Iterierte Anwendung:  $x = 11$   
 34,17,52,26,13,40,20,10,5,16,8,4,2,1

$$f(x) := \begin{cases} \text{kleinste } k \text{ mit } g^k(x) \leq 1 & \text{wenn solches } k \text{ ex.} \\ \perp & \text{sonst} \end{cases}$$

$f$  while-berechenbar. Offenes Problem: Ist  $f$  total?

**Satz:**

- (a) „while ist nicht immer durch loop ersetzbar“  
 (b) „loop ist durch while ersetzbar“

Zu (a)  $P_{\perp}$  terminiert nicht für jedes Tupel von Anfangswerten. (darf nie terminieren)

Zu (b) Definition:  $\text{loop}_m$ -Programme sind  $\text{while}_m$ -Programme ohne while. Zu jedem  $\text{loop}_m$ -Programm  $P$  existiert für geeignetes  $m' \geq m$  ein  $\text{while}_{m'}$ -Programm  $P'$  ohne loop-Konstrukt mit  $\llbracket P \rrbracket(x_1, \dots, x_m) = (y_1, \dots, y_m) \Leftrightarrow \llbracket P' \rrbracket(x_1, \dots, x_m, \underbrace{0, \dots, 0}_{m'}) = (y_1, \dots, y_m, 0, \dots, 0)$

Beweis: induktiv über den Aufbau des  $\text{loop}_m$ -Programms.

Interessanter Fall:  $P : \text{loop } Xi \text{ begin } P_0 \text{ end}$

Induktions-Voraussetzung liefert  $m_0$  und  $\text{while}_{m_0}$ -Programm  $P'_0$ , das Beh. erfüllt.

Schreibe  $Y$  für erste freie Variable  $X(m_0 + 1)$

$P' : Y := Xi; \text{ while } Y > 0 \text{ do } P'_0; Y := Y - 1 \text{ end}$

Frage: Reicht für die Programmierung totaler Funktionen die loop-Schleife?

Nein: Beispiel: Ackermann-Funktion  $A : \mathbb{N} \times \mathbb{N} \rightarrow \mathbb{N}$

**Def:**  $A(0, y) = y + 1$

$A(x + 1, 0) = A(x, 1)$

$A(x + 1, y + 1) = A(x, A(x + 1, y))$

$$\begin{aligned} A(1, 3) &= A(0, A(1, 2)) \\ &= A(0, A(0, A(1, 1))) \\ &= A(0, A(0, A(0, A(1, 0)))) \\ &= A(0, A(0, A(0, A(0, 1)))) \\ &= 5 \end{aligned}$$

Bem:

$A(0, y) > y$

$A(1, y) > y + 1$

$$\left. \begin{array}{l} A(2, 4) > 2y \\ A(3, 4) > 2^y \\ A(4, 4) > 2^{2^{2^{\cdot^{\cdot^2}}}} \\ A(5, 4) > 10^{10000} \end{array} \right\} y \text{ mal}$$

### 2.2.7 Satz: (Satz von Ackermann)

$A$  ist total und (while-)berechenbar, jedoch nicht loop-berechenbar. Im detail: Lemma: Zu jedem loop- $m$ -Programm  $P$  existiert Zahl  $k_p$  mit  $\max_{y_1, \dots, y_m} (\llbracket P \rrbracket(x_1, \dots, x_m)) < A(k_p, \max(x_1, \dots, x_m))$

Beweis durch Induktion über Aufbau der loop-Programme.

**Beweis des Satzes:** Annahme:  $P$  loop-Programm berechnet  $A$ . Wähle  $k_p$  gemäß Lemma.

$$A(x, y) = f_p^{(2)}(x, y) \leq \max(\llbracket P \rrbracket(x, y, 0 \dots, 0)) < A(k_p, \max(x, y)) \quad \text{wenn } x = y = k_p$$

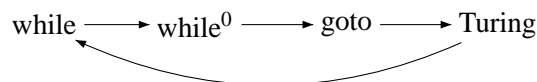
### 2.2.8 Äquivalenzsatz

**Ziel:** Für eine Funktion  $F : \mathbb{N}^n \rightarrow \mathbb{N}$  sind folgende Aussagen äquivalent:

1.  $f$  Turing-berechenbar
2.  $f$  goto-berechenbar
3.  $f$  while-berechenbar

Bei TM unterstellen wir die unäre Darstellung natürlicher Zahlen ( $n$  repräsentiert durch  $\underbrace{\llbracket \dots \rrbracket}_{n\text{-mal}}$ )

Beweisstrategie:



**Satz 2.1 (while<sup>0</sup> → goto):** Zu einem while<sup>0</sup>-Programm  $P$  kann man ein goto-Programm  $P'$  angeben mit  $\llbracket P' \rrbracket = \llbracket P \rrbracket$

(Dann klar:  $f : \mathbb{N}^n \rightarrow \mathbb{N}$  while<sup>0</sup>-berechenbar  $\Rightarrow$   $f$  goto-berechenbar).

**Beweis (induktiv über den Aufbau der while<sup>0</sup>-Programme):**

**Ind. Anf.:**  $P$  Wertzuweisung ( $P = X_i := X_i + 1$  oder  $X_i := X_i - 1$  ( $i \in \{1, \dots, m\}$ ))

**Ind. Schritt:**

- (a)  $P = P_1; P_2$   
 Ind.Vor.(\*): Zu  $P_1, P_2$  jeweils goto<sub>m</sub>-Programme  $P'_1, P'_2$  vorhanden mit  $\llbracket P'_1 \rrbracket = \llbracket P_1 \rrbracket$  und  $\llbracket P'_2 \rrbracket = \llbracket P_2 \rrbracket$
- (b)  $P = \text{if } X_i = 0 \text{ then } P_1 \text{ else } P_2 \text{ end}$   
 Ind.Vor. (\*)
- (c) loop bereits durch while ersetzt  
 $P = \text{while } X_i = 0 \text{ do } P_1 \text{ end}$   
 Ind.Vor.(\*) nur für  $P_1$

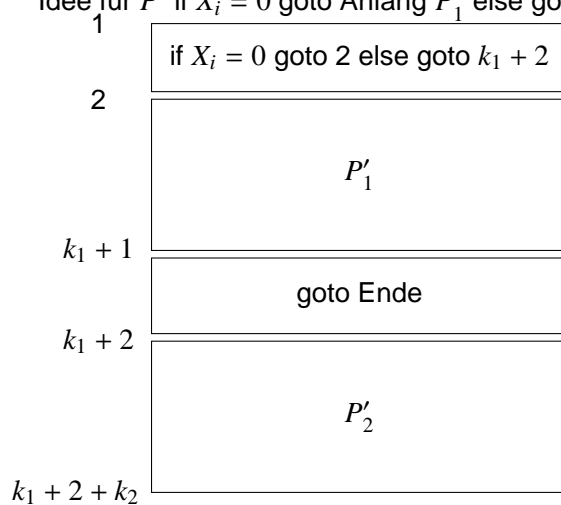
Beweis:

**Ind.Anf.:** Nehme  $P'$ :

1.  $X_i := X_i \pm 1$
2. STOP

**Ind. Schritt**

- (a)  $P = P_1; P_2$   
 Gemäß Induktions-Vorraussetzung betrachte goto<sub>m</sub>-Programme  $P'_1$  und  $P'_2$  (mit  $k_1$  bzw  $k_2$  Zeilen)  
 $P'$  entsteht aus Hintereinanderausführung von  $P'_1, P'_2$  unter Streichung der von STOP in  $P_1$  Erhöhung der Zeilennummern in  $P'_2$  um  $k_1 - 1$
- (b)  $P = \text{if } X_i = 0 \text{ then } P_1 \text{ else } P_2.$   
 Nach I.V. wähle  $P'_1, P'_2$   
 Idee für  $P'$  if  $X_i = 0$  goto Anfang  $P'_1$  else goto Anfang  $P'_2$



Vorraussetzung:  $P'_1$  mit  $k_1$  Zeilen  
 $P'_2$  mit  $k_2$  Zeilen

$P'$  entsteht aus: Zeile if  $X_i = 0$  goto 2 else goto  $k_1 + 2$

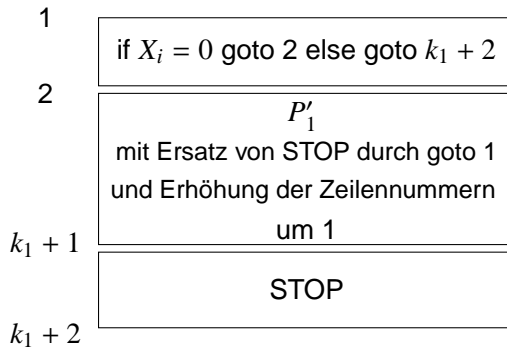
$P'_1$  mit um 1 erhöhten Zeilennummern und Ersatz von STOP durch goto  $k_1 + 1 + k_2$

$P'_2$  mit um  $k_1 + 1$  erhöhten Zeilennummern.

(c)  $P = \text{while } X_i = 0 \text{ do } P_1 \text{ end}$

I.V. liefert  $P'_1$

Idee für  $P'$ :



**Satz 2.2:** Zu jedem goto<sub>m</sub>-Programm  $P$  kann man TM  $M_P$  konstruieren mit  
 $\llbracket P \rrbracket(x_1, \dots, x_m) = (y_1, \dots, y_m) \Leftrightarrow M$  erreicht von  $q_0 |^{x_1} \sqcup |^{x_2} \sqcup \dots |^{x_m} \sqcup \dots$  aus die Stopkonfiguration  
 $q_2 |^{y_1} \sqcup |^{y_2} \sqcup \dots |^{y_m} \sqcup$

**Beweis:**

Simuliere Zeilenweise die Anweisungen

j  $X_i := X_i + 1$

j  $X_i := X_i - 1$

j if  $X_i = 0$  goto  $k$  else goto  $l$

16.11.04

**Satz 2.3:** Jede goto-berechenbare Funktion ist Turing-berechenbar

**Beweis:**

Zu gegebenem goto<sub>m</sub>-Programm  $P$  (etwa mit Zeilen  $1, \dots, k$ ) konstruiere TM  $M_P$ , die  $P$  schrittweise simuliert.

Verwende Blöcke von Turingzeilen für Simulation der Zeilen von  $P$

$q_1, \dots, q_k$  Zustände für den Einstieg in diese Blöcke ( $q_i$  für  $i$ -te  $P$ -Zeile).

$q_1$  Anfangszustand,  $q_k$  Stopzustand.

Von  $q_i$  aus simuliere  $i$ -te Zeile für goto-Konfiguration  $(i, n_1, \dots, n_m)$  von Turing-Konfiguration

$q_i |^{n_1} \sqcup |^{n_2} \sqcup \dots \sqcup |^{n_m} \sqcup$

Simulationsvorgang:

Fall 1  $X_j := X_j + 1$

- Gehe auf  $j$ -tes „nach rechts, markiere durch „Unterstreichen“  
gehe weiter auf das  $m$ -te „
- Verschiebe, von rechts zurücklaufend bis zur stelle „ $\sqcup$ “, die Bandinschrift um 1 Feld nach rechts (wobei  $\sqcup \rightsquigarrow \sqcup$ ), drucke auf Platz von „ $\sqcup$ “ ein |

- Gehe nach links auf j-te  $\sqcup$ (Position 1 Feld vor Anfang)
- Gehe 1 Feld nach rechts und in Zustand  $q_{i+1}$

Fall 2  $i \quad X_j := X_j - 1$

Fall 3  $i \quad \text{if } X_j = 0 \text{ goto } l \text{ else goto } l'$

- gehe auf j-tes  $\sqcup$ nach rechts, prüfe, ob davor  $\sqcup$ steht.  
In diesem Fall gehe zum Anfang zurück in  $q_l$ , sonst zum Anfang und in  $q_r$

**Satz 2.4:** Jede Turing-berechenbare Funktion ist while-berechenbar

**Beweis:**

Geg. TM  $M$  mit Zuständen  $q_1, \dots, q_r, q_0$  (Anfangszustand:  $q_1$ ; Stopzustand  $q_0$ ) und einem Arbeitsalphabet  $\Gamma = \{a_0, \dots, a_n\}$

Kodiere  $M$ -Konfiguration  $a_{i_s} \dots a_{i_1} a_{i_0} q_l a_{j_0} a_{j_1} \dots a_{j_k}$  durch ein Zahlentripel  $(L,R,Z)$  mit  
 $L = ((i_s \dots i_0)_{n+1}) \quad K = (j_k \dots j_0)_{n+1} \quad Z = l$

**Bsp.:**  $\Gamma = \{\sqcup\} = \{a_0, a_1\} \quad n=2$   
 Konf:  $||| \sqcup |q_{17} \sqcup |||$   
 $L = (11101)_2 = 29 \quad R = (1110)_2 = 14 \quad Z = 17$

Phasen der Simulation für den Fall, dass durch  $M$  eine zweistellige Funktion berechnet wird.  
 while-Programm muss Werte  $X_1, X_2$  in Ergebniswert für  $X_1$  überführen

$P$  verwendet Hilfsvariablen  $X_3, X_4, X_5, X_6$  (schreibe  $L,R,Z,B$ ) ( $B =$  Basis für Zahlendarstellung)

**1. Phase** Stelle den Kode der Anfangskonfiguration her

Für  $X_1 = n_1, X_2 = n_2$  Kode von  $q_1 |^{n_1} \sqcup |^{n_2}$

$L=0 \quad R = (\underbrace{11 \dots 1}_{n_2} 0 \underbrace{1 \dots 1}_{n_1})_{n+1} \quad Z = 1$

**2. Phase** Schrittweise Simulation von  $M$ , durch Update der Werte für  $L,R,Z$ , bis  $M$  Stopzustand  $q_0$  erreicht. Simulation stoppt für  $Z=0$

**3. Phase**  $\underbrace{\quad\quad\quad}_{L} q_0 \underbrace{||| \dots |}_{e} \underbrace{\quad\quad\quad}_{R}$

Extraktion der Länge  $e$  der Strichfolge aus  $R$  und entsprechende Zuweisung von  $e$  in  $X_1$

**Details**

Phase 1: wie oben

Berechne  $R = \underbrace{1 + 1 \cdot B + 1 \cdot B^2 + \dots + 1 \cdot B^{X_1-1}}_{\text{loop-programmierbar}} + \underbrace{1 \cdot B^{X_1+1} + \dots + 1 \cdot B^{X_1+X_2}}$

```

P := 1;
S := 0;
loop X1 begin S := S + P
              P := P * B
            end

```

Phase 2: while  $Z > 0$  do (Einzelschrittsimulation)  
 if Zustand  $i$ , AF mit  $a_j$   
 then führe Turingzeile  $q_i a_j \dots$  durch

Konvention: if  $\sim$  then  $\_$  für if  $\sim$  then  $\_$  else  $X_1 := X_1$

Detail: if  $Z=i$  and  $(R \bmod B)=j$  then (update)  
 (update) nach Fallunterscheidung:  $q_i a_j a_{j'} R/L/N q_{i'}$

Fall R:  $\dots q_i a_j | \dots$   
 $210 \quad \beta 59$   
 $4$   
 $R = R \text{ div } B \quad L = L * B + j \quad Z_i = i'$

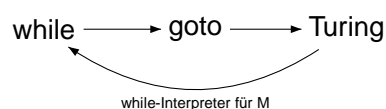
Fall L:  $210 q_i 359$   
 $4$   
 $R = R - j + j'$   
 $R = R * B + (L \bmod B)$   
 $L = L \div B \quad Z_i = i'$

Fall N  $R := R - j + j'$   
 $Z_i := i'$

Phase 3: Benutze Zähler für iterierte Division von  $R$  durch  $B$  bis erstmals  $R \bmod B = 0$   
 $X_1 := 0$   
 loop R begin if  $R \bmod B > 0$  then  $X_1 := X_1 + 1$  else  $R := 0$ ; end  
 $R := R \text{ div } B$   
 end

**Satz 2.5:** Äquivalent sind für  $f : \mathbb{N}^n \rightarrow \mathbb{N}$

- $f$  ist Turing-berechenbar
- $f$  ist goto-berechenbar
- $f$  ist while-berechenbar



## 3 Unentscheidbarkeit

**Erstes unentscheidbares Problem:** Eingabe: Tupel  $(n, m) \in \mathbb{N}^2$

**Frage:** Gilt  $(n, m) \in (n, BB(n))$

Schon gezeigt:  $BB(n)$  ist nicht berechenbar (Satz von Rado)

Wir wissen:  $f : \mathbb{N} \rightarrow \mathbb{N} \Leftrightarrow G_f = \{(n, f(n))\}$  semi-entscheidbar

$G : BB$  ist nicht semi-entscheidbar  $\Rightarrow$  Vorherige Frage ist unentscheidbar

### 3.1 Wortproblem für TM

Sei  $M$  TM über  $\Sigma_{bool}$ ,  $w \in \Sigma_{bool}^*$

- $M : w \rightarrow STOP$  falls  $M$  angesetzt auf  $w$  stoppt
- $M : w \rightarrow v$  falls  $M : w \rightarrow$  stoppt und  $v$  ausgibt
- $M : w \rightarrow \infty$  falls nicht  $M : w \rightarrow STOP$

#### 3.1.1 Definition

Gegeben eine TM  $M$ ,  $w \in \Sigma_{bool}$

Frage: Gilt  $M : w \rightarrow STOP$

**Satz 3.1:** Das Wortproblem für TM ist unentscheidbar.

**Mit Church-Turing-These (CTT):** Es gibt keinen Algorithmus, der zu jeder TM  $M$  über  $\Sigma = \{0, 1\}$  und zu jedem Wort  $w \in \Sigma^*$  entscheidet, ob  $M : w \rightarrow STOP$

(Es gibt keine TM  $M_0$  die angesetzt auf das Paar  $(M, w)$  entscheidet ob  $M : w \rightarrow STOP$  gilt.)

#### Vorbemerkung:

Kodierung von TM  $M$  als Wörter über  $\Sigma_{bool}$ .  $TM = (Q, \Sigma, \Gamma, q_0, q_s, \delta)$

$Q = \{1, \dots, r\}$       $\Sigma = \Sigma_{bool}$       $\Gamma = \{1, \dots, s\}$  wobei 1 „=“

$q_s$  der einzige Zustand, von dem keine Trans. ausgeht.

Kodiere Turingzeile  $z = (i, j, j', R/L/N, i')$  mit  $code(z) = 0^i 1 0^j 1 0^{j'} 1 0 0/00/000 1 0^{i'} 11$

Definiere Kodierung von  $M$ :

$\langle M \rangle := code(z_1) \dots code(z_l) 1$  wobei  $z_1 \dots z_l$  alle Turingzeilen von  $M$  sind und  $q_0$  ist der erste Zustand der in der Kodierung auftritt

**Eigenschaften von  $\langle M \rangle$**

(a)  $\langle M \rangle$  bestimmt eindeutig eine TM  $M$  über  $\Sigma_{bool}$

(b) Es gibt einen Algorithmus, der zu  $w \in \Sigma_{bool}$  überprüft ob  $w = \langle M \rangle$  für eine TM  $M$

(c) 111 kommt nur am Ende der Kodierung vor.

**Umformulierung:** Es gibt keine TM  $M_0$  über  $\Sigma_{bool}$  die zu einer Eingabe  $\langle M \rangle w \in \Sigma_{bool}^*$  entscheidet, ob  $M : w \rightarrow STOP$  gilt

**Beweis:**

**Ann.** Es gibt eine TM  $M_0$  mit  $M_0 : \langle M \rangle w \rightarrow \begin{cases} 1 & \text{falls } M : w \rightarrow STOP \\ 0 & \text{sonst} \end{cases}$

für alle TM  $M$  und alle  $w \in \Sigma_{bool}^*$

Modifiziere  $M_0$  zu  $M_1$  mit Eingabe  $\langle M \rangle$ .  $M_1$  arbeite wie folgt:

1. Stelle aus  $\langle M \rangle$  das Wort  $\langle M \rangle \langle M \rangle$
2. Arbeite wie  $M_0$  auf  $\langle M \rangle \langle M \rangle$
3. Falls  $M_0 : \langle M \rangle \langle M \rangle \rightarrow 1$  dann gehe  $M_1$  in eine  $\infty$ -Schleife

Zu 3) Falls  $M_0 : \langle M \rangle \langle M \rangle \rightarrow 0$  dann stoppe  $M_1$

Dann gilt für alle TM  $M$ :

$$\begin{aligned} M_1 : \langle M \rangle \rightarrow STOP &\Leftrightarrow M_0 : \langle M \rangle \langle M \rangle \rightarrow 0 \\ &\Leftrightarrow M : \langle M \rangle \rightarrow \infty \end{aligned}$$

Für  $M = M_1$

$$M_1 : \langle M_1 \rangle \rightarrow STOP \Rightarrow M_1 : \langle M_1 \rangle \rightarrow \infty$$

**Satz:** Es gibt keinen Algorithmus, der zu jeder TM  $M$  über  $\Sigma = \{0, 1\}$  und zu jedem Wort  $w \in \Sigma^*$  entscheidet, ob  $M : w \rightarrow STOP$

„Das Wortproblem für TM ist unentscheidbar“

**Bemerkung 1** Der Satz überträgt sich unmittelbar auf jeden Programmierformalismus mit dem man Turingmaschinen simulieren kann.

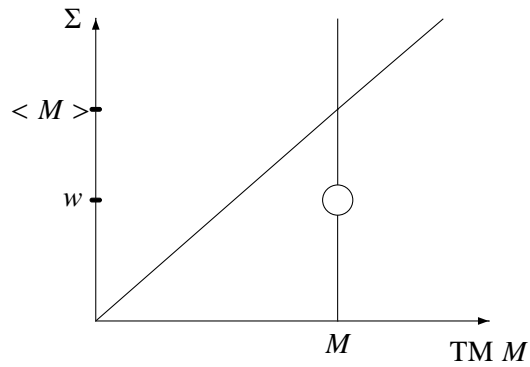
Also z.B.:

Gegeben ein Java-Programm  $P$  und Eingabedaten  $E$ , terminiert  $P$  gestartet mit  $E$ ?

ist unentscheidbar.

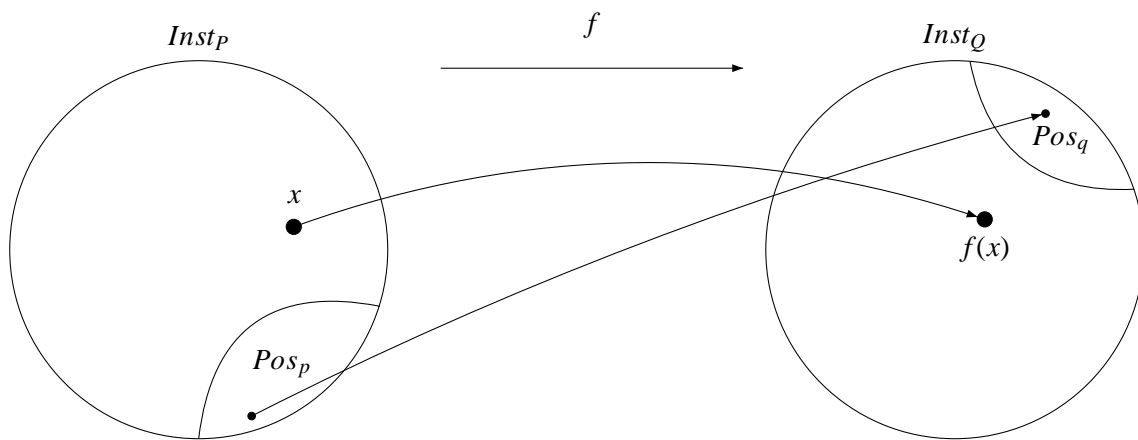
**Bemerkung 2** Algorithmus über beliebige TM wäre selbst durch TM darstellbar und würde somit über sich selbst Auskunft geben.

**Beweismethode** (zum Satz) „Diagonalschluss“



$M \quad w = \langle M \rangle$   
 $M_0 \rightarrow M_1$  „Diagonale umgedreht“  
 Herkunft: Cantor

**Urform des Cantorschen Beweises**



$B \ A \rightarrow B$

$A \rightarrow$

Beh. Menge der unendlichen 0-1-Folgen ist nicht abzählbar  
 Ann. Es gibt derartige Abzählung  
 $\beta_0 = b_{00}b_{01}b_{02} \dots$   
 $\beta_1 = b_{10}b_{11}b_{12} \dots$

$$\beta_2 = b_{20}b_{21}b_{22} \dots$$

⋮

$$\text{Betrachte } \bar{\beta} := \bar{b}_{00}\bar{b}_{11}\bar{b}_{22} \dots$$

$\bar{\beta}$  von jedem  $\beta_i$  verschieden, fehlt somit in der Liste  $\zeta$

### Satz 3.2 (über Programmierformalismen für totale Funktionen):

#### Vorbemerkung:

Sei  $P$  ein Programmierformalismus mit Programmen  $P_0, P_1, P_2, \dots$  mit

1. Jedes  $P_i$  berechnet eine totale Funktion  $f : \mathbb{N} \rightarrow \mathbb{N}$
2. Es gibt einen Algorithmus, der zu  $i$  jeweils das  $P_i$  herstellt und dann für Eingabe  $j$  den Wert „ $P_i(j)$ “ liefert.

Dann existiert eine berechenbare totale Funktion  $f$ , die durch kein  $P_i$  berechnet wird.

#### Beweis:

Betrachte  $F : \mathbb{N} \rightarrow \mathbb{N}$  definiert durch  $F(i) = P_i(i) + 1$

$F$  ist total, berechenbar (Vor. 2)

Zeige:  $F$  wird durch kein  $P_i$  berechnet (dann fertig)

Annahme:  $P_{i_0}$  berechnet  $F$

$$P_{i_0} = F_{i_0} \otimes P_{i_0} + 1 \zeta$$

## 3.2 Reduktionen und weitere unentscheidbare Probleme

<b>WP:</b>	Gegeben TM $M$ über $\Sigma = \{0, 1\}$ , $w \in \{0, 1\}^*$
Wortproblem	Frage: $M : w \rightarrow STOP?$
<b>HP</b>	Gegeben: TM $M$ über $\Sigma = \{0, 1\}$
Halteproblem	Frage: $M : \varepsilon \rightarrow STOP?$
<b>ÄP</b>	Gegeben: TM $M_1, M_2$ über $\Sigma = \{0, 1\}$
Äquivalenzproblem	Frage: Berechnen $M_1, M_2$ dieselbe Funktion $f : \Sigma^* \rightarrow \Sigma^*$ ?

### 3.2.1 Ziel:

Neben WP sind auch HP und ÄP unentscheidbar

**Vorbereitung:** 1. Normierung des Begriffs „Entscheidungsproblem“

2. Reduktion zwischen Entscheidungsproblemen

**Zu 1.:** Ein Entscheidungsproblem hat die Form  $P = (Inst_P, Pos_P)$ , wobei  $Inst_P$  die Instanzenmenge von  $P$  ist

$Pos_P$  die Menge der Instanzen, die die positive Antwort (ja) verlangen.

$$Pos_P \subset Inst_P$$

#### Beispiel:

(a) Sprache  $L \subset \Sigma^*$  bestimmt Problem  $P = (\Sigma^*, L)$

(b) WP

 $Inst_{WP}$ : Menge der Paare (TM, Eingabewort) $Pos_{WP}$ : Menge der Paare  $(M, w)$  mit  $M : w \rightarrow STOP$ 

(c) HP

 $Inst_{HP}$  = Menge der TM  $M$  $Pos_{HP}$  = Menge der TM  $M$  mit  $M_\varepsilon \rightarrow STOP$ 

**Definition 6** Für Probleme  $P = (Inst_P, Pos_P)$ ,  $Q = (Inst_Q, Pos_Q)$  def.  $P \leq Q$  („ $P$  auf  $Q$  reduzierbar“), falls ex. eine berechenbare Funktion  $f : Inst_P \rightarrow Inst_Q$  mit für jedes  $x \in Inst_P : x \in Pos_P \Leftrightarrow f(x) \in Pos_Q$

**Lemma**Wenn  $P \leq Q$  und  $P$  unentscheidbar, dann auch  $Q$  unentscheidbar.Anschauung: „ $P \leq Q$ “ ~ „ $Q$  mindestens so schwer wie  $P$ “**Beweis** Formuliere Behauptung um zu folgender Äquivalenzaussage:Gelte  $P \leq Q$ . Wenn  $Q$  entscheidbar, dann  $P$  entscheidbar.Sei  $A_Q$  eine Entscheidung für  $Q$  (Entscheidung, ob für  $y \in Inst_Q$  gilt  $y \in Pos_Q$ )Sei  $A_f$  ein Algorithmus, der  $f : Inst_P \rightarrow Inst_Q$  berechnet mit  $x \in Pos_P \Leftrightarrow f(x) \in Pos_Q$  (\*)Gesucht: Entscheidungsalg.  $A_P$  für  $P$ AP: Zu  $x \in Inst_P$  wende  $A_f$  an, erhalte  $f(x)$ Auf  $f(x)$  wende  $A_Q$  an, übernehme dessen Antwort.AP testet zu  $x \in Inst_P$ , ob  $\underbrace{f(x) \in Pos_Q}_{\Leftrightarrow x \in Pos_P}$ Also entscheidet  $A_P$  das Problem  $P$ 

26.11.04

WP: TM über  $\Sigma = \{0, 1\}$ ,  $w \in \{0, 1\}^*$   $Inst_{WP}$  = Menge der Paare  $Pos_{WP}$  = Menge der  $(M, w)$  mit  $M : w \rightarrow STOP$ HP:  $Inst_{HP}$  = Menge der TM über  $\Sigma = \{0, 1\}$  $Pos_{HP}$  = Menge der TM  $M$  mit  $M : \varepsilon \rightarrow STOP$ **Satz 3.3:** HP unentscheidbar.**Beweis:**Nach Lemma genügt :  $WP \leq HP$ Finde berechenbare Transformation  $(M, w) \mapsto M'$  (von  $Inst_{WP}$  nach  $Inst_{HP}$  mit  $M : w \rightarrow STOP \Leftrightarrow M' : \varepsilon \rightarrow STOP$  Definiere  $M'$  so, dass  $M'$  folgendes tut (angesetzt auf das leere Band): $M'$  schreibt  $w$  auf das Band, gehe auf den ersten Buchstaben von  $w$  zurück, und arbeite dann wie  $M$ .Klar:  $f$  berechenbar, ebenso (\*) nach Def. von  $M'$ **Folgerung:** Das Entscheidungsproblem: gegeben Java-Progr.  $P$  mit Integer-Variablen:Frage: Terminiert  $P$ , initialisiert mit 0?

ist unentscheidbar

**ÄP:**  $Inst_{\ddot{A}P}$ : Menge der Paare von Turingmaschinen über  $\Sigma = \{0, 1\}$  $Pos_{\ddot{A}P}$ : Menge der Paare  $(M_1, M_2)$  von Turingmaschinen, wobei  $M_1, M_2$  dieselbe Funktion  $f : \Sigma^* \rightarrow \Sigma^*$  berechnen  $(M_1, M_2)$  „äquivalent“.

**Satz 3.4:** Äp ist unentscheidbar

**Beweis:**

Genügt nach Lemma:  $HP \leq \text{ÄP}$

Finde Transformation  $f$  (berechenbar), so dass  $f : M \mapsto (M_1, M_2)$  (Tm  $\rightarrow$  (Paar von TM'en)) mit :

$M : \varepsilon \rightarrow STOP \Leftrightarrow M_1, M_2$  berechnen dieselbe Funktion

**Idee:**  $M_1$  löscht Eingabewort (hat also leeres Band hergestellt) und arbeitet dann wie  $M$ , wobei bei STOP noch  $\sqcup$  gedruckt wird (dann Ausgabe das leere Wort  $\varepsilon$ ).

**Bemerkung:**  $M_1$  berechnet die konstante Funktion mit Wert  $\varepsilon$ , falls  $M : \varepsilon \rightarrow STOP$

$M_1$  berechnet die überall undefinierte Funktion, falls  $M : \varepsilon \rightarrow \infty$

Wähle als  $M_2$  die TM  $q_0 \sqcup /0/1 \sqcup N q_S$  welche die konst. Funktion mit Wert  $\varepsilon$  berechnet.

Also  $M : \varepsilon \rightarrow STOP \Leftrightarrow M_1 M_2$  berechnen dieselbe Funktion (nämlich Konstante  $\varepsilon$ ).

Insgesamt: Trans.  $f$  berechenbar und erfüllt (\*) wie gewünscht.

30. 11. 04

Ein **TM-Problem** ist ein Entscheidungsproblem  $P = (Inst_P, Pos_P)$  mit  $Inst_P =$  Menge der Turingmaschinen (mit Eingabealphabet  $\{0, 1\}$ )

Ein TM-Problem ist **nichttrivial**, wenn weder  $Pos_P = Inst_P$  noch  $Pos_P = \emptyset$

Ein TM-Problem heißt **semantisch**, wenn die Mitgliedschaft einer TM  $M$  in  $Pos_P$  nur von der durch  $M$  berechneten Funktion abhängt. (d.h.  $M_1, M_2$  äquivalent  $\Rightarrow M_1 \in Pos_P \Leftrightarrow M_2 \in Pos_P$ )

**Beispiel:**

$P_1 = (Inst_{TM}, Pos_{P_1})$  mit  $Pos_{P_1} =$  Menge der TM mit  $\leq 17$  Zeilen (nicht semantisch)

$P_2 = (Inst_{TM}, Pos_{P_2})$  mit  $Pos_{P_2} =$  Menge der TM die totale Funktionen berechnen (semantisch)

**Satz 3.5 (Satz von Rice):** Jedes nichttriviale semantische TM-Problem ist unentscheidbar.

**Beweis:**

Sei  $f_{\perp}$  die überall undefinierte Funktion.

Gegeben nichttriviales semantisches TM-Problem  $P = (Inst_{TM}, Pos_P)$

Da  $P$  semantisch ist, gilt  $\begin{cases} \text{Fall 1} & \text{Die TM, die } f_{\perp} \text{ berechnet, alle nicht in } Pos_P \\ \text{Fall 2} & \text{Die TM, die } f_{\perp} \text{ berechnet, alle in } Pos_P \end{cases}$

Hier Fall 1 (analog Fall 2): Zeige  $HP \leq P$ , dann fertig.

Finde Transformation  $f : M \mapsto M'$  mit  $M : \varepsilon \rightarrow STOP \Leftrightarrow M' \in Pos_P$

Wähle TM  $M_h$  aus  $Pos_P$ ; diese berechnet Funktion  $H \neq f_{\perp}$  (Fall 1).

Angabe von  $M'$  aus gegebenem  $M$

$M'$   $\left\{ \begin{array}{l} \text{bei Eingabe } w \text{ arbeite wie } M \text{ auf leerem band (und bewahre das Eingabewort } w, \text{ eventuell verschoben, a} \\ \text{Wenn } M : \varepsilon \rightarrow STOP, \text{ arbeite anschließend wie } M_h \text{ auf } w \end{array} \right.$

Also  $f : M \mapsto M'$  berechenbar.

Prüfe ob  $M : \varepsilon \rightarrow STOP \Leftrightarrow M' \in Pos_P$

- Falls  $M : \varepsilon \rightarrow STOP$ , berechnen  $M', M_h$  dieselbe Funktion, nämlich  $h$ . Da  $M_h \in Pos_P$ , auch  $M' \in Pos_P$  (beachte  $P$  semantisch).
- Falls  $M : \varepsilon \rightarrow \infty$ , terminiert  $M'$  nicht für beliebige Eingabe, also berechnet  $M'$  die Funktion  $f_{\perp}$ . Wegen Fall 1  $M' \notin Pos_P$ .

### 3.3 Drei unentscheidbare Probleme

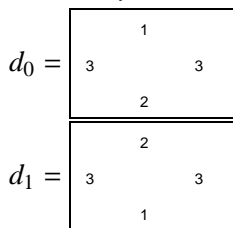
#### 1. Domino-Problem

- 2. Postsches Korrespondenzproblem
- 3. Halteproblem für goto<sub>2</sub>-Programme (2-Zähler-Maschinen)

### 3.3.1 Domino-Problem

**Beispiel:**

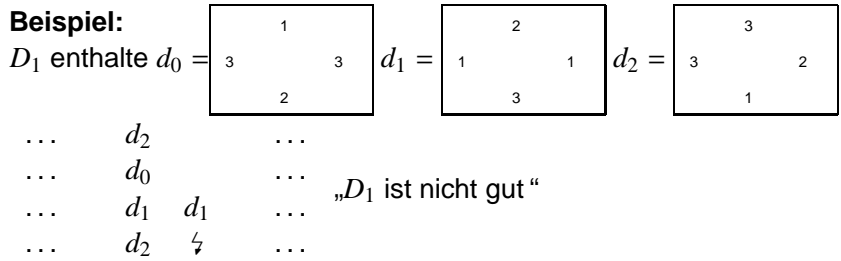
Dominospiel  $D_0$  sei gegeben durch die Dominotypen



Parkettierung der  $\mathbb{Z} \times \mathbb{Z}$ -Ebene:  $\dots d_1 d_1 d_1 d_1 \dots$   
 $\dots d_0 d_0 d_0 d_0 \dots$   
 $\dots d_1 d_1 d_1 d_1 \dots$   
 $\dots d_0 d_0 d_0 d_0 \dots$

„ $D_0$  ist gut“

**Beispiel:**



**Allgemein:** Ein Dominospiel ist eine Folge  $D = (d_0, \dots, d_k)$  von Dominotypen  $d_i$ , jedes  $d_i$  ein Quadrupel von „Farben“ (hier Zahlen) der Form  $(d_{i1}, d_{i2}, d_{i3}, d_{i4})$   $di1(N) W(di4) di2(O) di3(S)$   $d_{i1} = N(d_i), d_{i2} = O(d_i), d_{i3} = S(d_i), d_{i4} = W(d_i)$   
 Eine Parkettierung ist eine Funktion  $\pi : \mathbb{Z} \times \mathbb{Z} \rightarrow \{d_0, \dots, d_k\}$  mit  $\pi(0, 0) = d_0$  und  $O(\pi(i, j)) = W(\pi(i, j + 1))$   
 $S(\pi(i, j)) = N(\pi(i + 1, j))$   $(i, j) \in \mathbb{Z} \times \mathbb{Z}$

**D ist gut:**  $\Leftrightarrow$  ex. Parkettierung (von  $\mathbb{Z} \times \mathbb{Z}$ ) mit  $D$

**Dominoproblem:** Gegeben Dominospiel  $D$ . Frage ist  $D$  nicht gut?

**Satz 3.6:** Das Dominoproblem ist unentscheidbar

**Beweis:**

HP  $\leq D$

Finde eine Transformation, berechenbar,  $f : M \mapsto D_M$  mit  $M : \varepsilon \rightarrow STOP \Leftrightarrow D_M$  nicht gut  
 Ansatz zur Konstruktion von  $D_M$ :

- Untere Farbfolge der zentralen Zeile der Parkettierung = Anfangskonf. von  $M$
- Untere Farbfolge der Zeile darunter: nächste Konfig von  $M$
- usw.

Keine Dominotypen mit oberer Farbe, die  $q_s$  (Stopzustand) enthält.

$$\left| \begin{array}{l} M : \varepsilon \rightarrow STOP \Rightarrow D_M \text{ nicht gut} \\ M : \varepsilon \rightarrow \infty \Rightarrow D_M \text{ gut} \end{array} \right.$$

**Beispiel:**

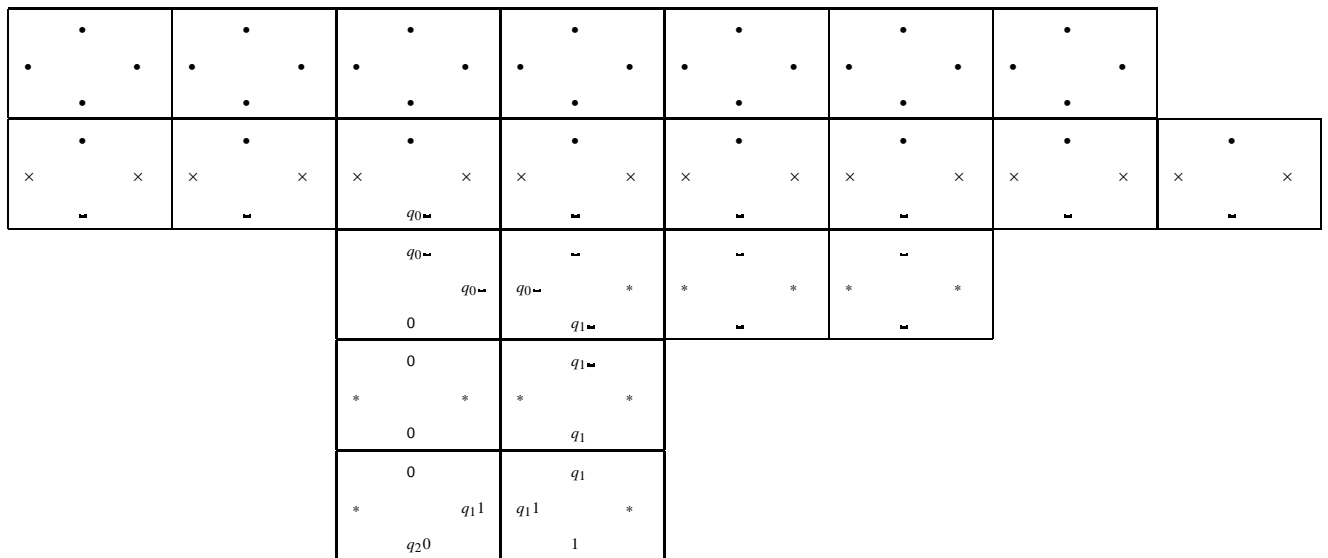
$M$ :

$q_0$	$\sqcup$	0	R	$q_1$
$q_1$	$\sqcup$	1	N	$q_1$
$q_1$	1	1	L	$q_2$
$q_2$	0	0	N	$q_s$

Konfigurationen:

$\dots \sqcup (q_0 \sqcup) \sqcup \dots$   
 $\dots \sqcup 0 (q_1 \sqcup) \sqcup \dots$   
 $\dots \sqcup 0 (q_1 1) \sqcup \dots$   
 $\dots \sqcup (q_2 0) 1 \sqcup \dots$   
 $\dots \sqcup (q_s 0) 1 \sqcup \dots$

**Parkettierung**



**Lemma**

Läuft  $M$  auf leeres Band angesetzt  $k$  Schritte, dann erlaubt  $M$  von eindeutiger Zentralzeile aus Parkettierung von  $k$  Zeilen, und untere Farbfolge der  $k$ -ten Zeile ist die  $M$ -Konfig nach  $k$  Schritten.

Also:  $M : \varepsilon \rightarrow \infty \Rightarrow D_M \text{ gut}$   
 $M : \varepsilon \rightarrow STOP \Rightarrow D_M \text{ nicht gut.}$

### 3.3.2 Postsches Korrespondenzproblem (PCP)

Gegeben: Liste  $X = (x_1, \dots, x_n)$ ,  $Y = (y_1, \dots, y_n)$  von Wörtern  $x_i, y_i$  über Alphabet  $\Sigma$

**Frage:** Hat  $(X, Y)$  eine Lösung, d.h. ex. Indexfolge  $i_1, \dots, i_k$  mit  $x_{i_1} \dots x_{i_k} = y_{i_1} \dots y_{i_k}$  (erzeugtes Wort heißt Lösungswort)

Modifiziertes PCP (MPCP) verlangt Indexfolge mit  $i_1 = 1$

**Beispiel:**

$X = (abbaaa, ab, a)$        $Y = (ab, b, aaa)$

$X: abaaa a a$

$Y: ab aaa aaa$

Lösung: 1,3,3,2

**Beispiel:**

$X = (ab, baa, aba)$        $Y = (aba, aa, baa)$

$X: ab aba aba$

$Y: aba baa baa$

Lösung:  $(X, Y)$  hat keine spezielle Lösung

**Satz 3.7:** Das Problem MPCP ist unentscheidbar

(Analog gilt: Das PCP ist unentscheidbar)

**Beweis:**

Zeige  $WP \leq MPCP$

Hierzu finde berechenbare Transformation  $(M, w) \mapsto (X_{M,w}, Y_{M,w})$  mit  $M : w \rightarrow STOP \Leftrightarrow (X_{M,w}, Y_{M,w})$  hat Lösung

Idee für Konzeption von  $X_{M,w}, Y_{M,w}$ : Aufbau des Lösungswortes als Folge der Konfigurationen von  $M$ , angesetzt auf  $w$

**Beispiel:**

$M: q_0 a b R q_1$

$q_1 b b R q_2$

$q_2 \sqcup b L q_3$

$w = ab$

TM-Berechnung (Folge der Konf.wörter):

$\#q_0ab\#bq_1b\#bbq_2\#bq_sbb\#$

$X_{M,w}$	$Y_{M,w}$	
$\#$	$\# q_0 ab \#$	1. Herstellung der Anf.Konf.
$q_0 a$	$b q_1$	2. Konfig.-Update
$b$	$b$	
$\#$	$\#$	
$q_1 b$	$b q_2$	
$c q_2 \sqcup$	$q_s c b$	Für $c \in \Gamma$ (Arbeitsalphabet)
$c q_2 \#$	$q_s c b \#$	
$q_s c$	$q_s$	
$q_s$	$q_s$	
$b q_s$	$q_s$	
$q_s \# \#$	$\#$	

**Konstruktion zeigt:**  $M : w \rightarrow \begin{cases} STOP \\ \infty \end{cases} \Rightarrow$  Aufbau eines Lösungswortes ist eindeutig, reproduziert TM-Berechnung und erlaubt Herstellung des gleichen Wortes im Fall  $M : w \rightarrow STOP$

10.12.04

**Dominospiel  $D_M$  für TM  $M = (Q, \Sigma, \Gamma, q_0, \delta, q_s)$**

### 3.3.3 Halteproblem für goto<sub>2</sub>-Programme (2-Zähler-Maschinen)

Gegeben: ein goto<sub>2</sub>-Programm  $P$  (Variablen  $X_1, X_2, +1, -1, \text{Sprung}$ )

Frage: Stoppt  $P$ , gestartet mit Wert 0 für  $X_1, X_2$ ? — HP (goto<sub>2</sub>)

**Satz 3.8:** Das Halteproblem für goto<sub>2</sub>-Programm ist unentscheidbar

**Beweis:**

$HP \stackrel{a}{\leq} HP(\text{goto}_4) \stackrel{b}{\leq} HP(\text{goto}_2)$  **Zu (a)** Finde berechenbare Transformation TM  $M \mapsto$  goto<sub>4</sub>-Programm  $P_M$  mit  $M : \varepsilon \rightarrow STOP \Leftrightarrow P_M : (0, 0, 0, 0) \rightarrow STOP$ .

Die TM  $M$  habe Zustände  $q_0, \dots, q_s$ , Arbeitsalphabet  $\Gamma = \{0, 1, \dots, 9\}$  ( $0 \equiv \sqcup$ )

Idee für Simulation von  $M$  durch  $P_M$  benutzt Darstellung der TM-Konfiguration ... 000317 $q_i$ 208800 ...

durch  $P_M$ -Konfig:  $( \underbrace{\quad}_{\text{Zeilennummer}}, \underbrace{8802}_{X_1}, \underbrace{317}_{X_2}, \underbrace{0}_{X_3}, \underbrace{0}_{X_4} )$

Allgemein für  $|\Gamma| = m$  benutzte  $m$ -adische Darst.

Update der Konfiguration unterschieden nach Linksschritt, Rechtsschritt, stationärer Schritt der TM.

Hier z.B. Linksschritt  $q_i \ 2 \ 5 \ L \ q_j$

Neue  $P_M$ -Konfiguration  $|(l_j, 88057, 31)$

Mit Zeile  $l_i$  beginnt folgender Programmabschnitt in  $P_M$ :

- prüfe, ob  $8802 \bmod 10 = 2$
- in diesem Fall:  $X_2 := X_1 \dot{-} 2$   
 $X_1 := X_1 + 5$   
 $X_1 := \times 10$   
 $X_1 := X_1 + (X_2 \bmod 10)$   
 $X_2 := X_2 \text{div} 10$  Sprung nach  $l_j$   
 $X_3, X_4$  als Hilfsvariablen.

**Zu (b)** Gesucht: Simulation eines goto<sub>4</sub>-Programms  $P$  durch goto<sub>2</sub>-Programm  $P'$

Idee: Darstellung der  $P$ -Konfiguration  $(i, k_1, k_2, k_3, k_4)$  in Form  $(i', 2^{k_1} \cdot 3^{k_2} \cdot 5^{k_3} \cdot 7^{k_4}, 0)$

Simulation von  $X_j := X_j + 1 \rightarrow X_1 := X_1 \times 3$

$$X_j := \dot{-} 1 \rightarrow \begin{cases} X_1 \text{div} 3 & X_1 \bmod 3 = 0 \\ X_1 & \end{cases}$$

if  $X_j = 0$  goto  $i_1$  else goto  $i_2 \rightarrow$  analog mit Test  $X_1 \bmod 3 = 0$

### 3.4 Universalität und Vollständigkeit

14.12.04

Turingmaschine als Präzisierung des Algorithmusbegriffs ermöglicht Idee von „Algorithmus auf Algorithmus“.

Technische Umsetzung: TM  $M$  mit Eingabe  $code(M_1)$

Realisierung in Informatik:

1. Compiler (Programmübersetzer)
2. Universalprozessor in von-Neumann-Rechner  
 Prozessor: Algorithmus der
  - auszuführendes Programm  $P$
  - Eingabedaten  $D$  für  $P$

Erste konzeptuelle Realisierung durch Turing 1936 anhand der Kodierung des Wortproblems:

$L_{WP} = \{code(M)w \mid M : w \rightarrow STOP\}$

Bewiesen:  $L_{WP}$  nicht entscheidbar.

**Satz 3.9 (Turing):**  $L_{WP}$  ist (Turing-)semi-entscheidbar.

**Beweis:**

Beweis erfordert Konstruktion einer „universellen Turingmaschine“  $U$  mit

$$U : code(M)w \rightarrow STOP \quad \forall M$$

Mühsame Konstruktion im Detail.

$$\begin{array}{l} U \sim \text{Prozessor} \\ code(M)w \sim \text{ausgeführtes Programm } P \text{ Eingabedaten } D \end{array}$$

**Analog:** Kodierung  $L_{HP} = code(M) \mid M : \varepsilon \rightarrow STOP$  Turing semi-entscheidbar und Turing-aufzählbar.

Anordnung der Sprachen mit  $\leq$

$L$  entsch.  $\Rightarrow L \leq L_{WP}$ , aber  $L_{WP} \not\leq L$

**Satz 3.10:**  $L$  aufzählbar  $\Rightarrow L \leq L_{WP}$

Also ist  $L_{WP}$  **vollständig** für Klasse der aufz. Sprachen bzgl.  $\leq$ .

Formal:

1.  $L_{WP}$  aufz.
2.  $L$  aufz.  $\Rightarrow L \leq L_{WP}$

**Beweis:**

Gegeben  $L$  aufz., etwa mit TM  $M$ , so dass  $M : w \rightarrow STOP \Leftrightarrow w \in L$

Finde Transformation  $w \mapsto code(M_w)u_w$  mit  $\underbrace{w \in L}_{M:w \rightarrow STOP} \Leftrightarrow \underbrace{code(M_w)u_w \in L_{WP}}_{M_w:u_w \rightarrow STOP}$

Setze einfach  $M_w = M$ ,  $u_w = w$

# 4 Komplexitätstheorie: Grundlagen

## 4.1 Zeitkomplexität

Berechnungsaufgabe gegeben, z.B. repräsentiert durch eine Sprache  $L \subseteq \Sigma^*$   
 (Gegeben  $w \in \Sigma^*$ , entscheide ob  $w \in L$ )

**Frage:** Kriterium für Effizienz einer algorithmischen Lösung?

Problem 1 Theoretisches Modell der Turingmaschine angemessen?

Problem 2 Genaue Def. von Zeitaufwand.

**Zu Problem 2** Messe die (Zeit-)Komplexität durch Anzahl der Schritte einer TM in Abhängigkeit von Eingabegröße (Wortlänge)

Zu fester Eingabelänge  $n$  nehme die maximale Schrittzahl für Eingaben  $w$  der Länge  $n$  (Worst Case).

Erhalte  $t : \underbrace{\mathbb{N}}_{\text{Eingabelänge}} \rightarrow \underbrace{\mathbb{N}}_{\text{max. Schrittzahl}}$

Unterscheide die Funktion  $t$  nur hinsichtlich asymptotischer Wachstumsrate.

**Definition 7** Zu  $g : \mathbb{N} \rightarrow \mathbb{N}$  sei  $O(g)$  die Klasse der Funktion  $f : \mathbb{N} \rightarrow \mathbb{N}$  mit  $\exists c > 0 \exists n_0 \forall n \geq n_0 f(n) \leq c \cdot g(n)$

Typische Funktionen  $g : n, \log(n), n \log(n), n^2, 2^n, \dots$

**Definition 8** TM  $M$  terminiere für jede Eingabe  $w$ .

$M$  heißt  $O(g(n))$ -zeitbeschränkt, falls folgende Funktion  $f$  zu  $O(g(n))$  gehört:

$f(n) = \max\{m \mid \text{ex. } w \text{ mit } |w| = n, M \text{ auf } w \text{ läuft } m \text{ Schritte}\}$

**Beispiel:**

Sprache  $L = \{a^i b^j \mid i \geq 1\}$

ā a a a b b b b̄

$M_1$ :

1. ein Durchlauf (Rückkehr zum 1. Buchstaben) für Test, ob Eingabe aus a-Block gefolgt von b-Block besteht.
2. falls bei Durchlauf sowohl a's als auch b's vorhanden, lösche jeweils erstes a und letztes b.
  - Prüfe, ob Restwort
    - a's und b's hat → weiter
    - nur a's hat → „nein“

- nur b's hat → „nein“
- kein a, kein b → „ja“

$n$  = Länge der Eingabe

Zeitaufwand für Phase 1:  $O(n)$

Zeitaufwand für Phase 2: Anzahl der Durchläufe  $\leq \lceil \frac{n}{2} \rceil$  pro Durchlauf  $\leq 2n$  Schritte.

[genauer insgesamt  $2n + 2(n - 2) + 2(n - 4) + \dots + 2$ ]

Schrittzahl insgesamt  $O(n^2)$ .

$M_1$  ist  $O(n^2)$ -zeitbeschränkt.

a a̅ a a̅ b b̅ b b̅ b  
a a̅ b b̅ b | a b b

Alternative:

$M_2$ : streiche jeweils jeden 2. Buchstaben.

Dann pro Durchlauf  $O(n)$  Schritte.

Anzahl der Durchläufe  $O(\log n)$

Gesamtaufwand  $O(n \log n)$

### Probleme

1. Finde möglichst effiziente Lösung
2. Finde untere Schranken für Effizienz.

17.12.04

$M_2$  zum Test, ob  $w \in \{a^i b^j | i \geq 1\}$

1. Test, ob Eingabe  $w$  die Form  $a^i b^j$ ; wenn nicht, stop mit „nein“

(\*). prüfe, ob  $\leq$  ein  $a$  und  $\leq$  ein  $b$  vorhanden; in diesem Fall stop mit entsprechender Antwort.

2. streiche jedes zweite  $a$  und jedes zweite  $b$

führe Test (\*) durch

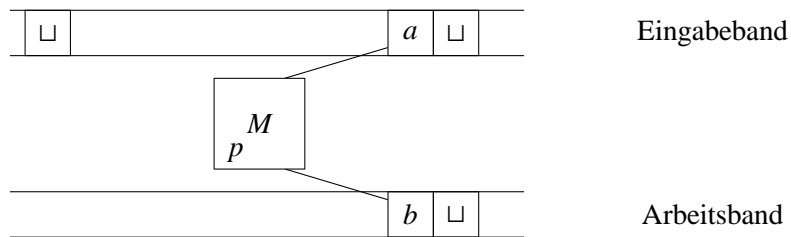
sonst: falls entweder nur letztes  $a$  oder nur letztes  $b$  gestrichen, stop mit „nein“ sonst zurück zu 2.

### Modifikation des TM-Modells: Offline-TM

**Definition 9** Eine Offline-TM hat 2 Bänder (Eingabeband zum Lesen und Arbeitsband zum Lesen und Schreiben) und Anweisungen folgender Form:

$$p a b b' L/N/R L/N/R q$$

In Zustand  $p$  mit  $a$  auf Eingabe- und  $b$  auf Arbeitsband, drucke  $b'$  statt  $b$ , bewege Eingabe- und Arbeitskopf gemäß 5. und 6. Komponente und gehe in Zustand  $q$ .



$M_3$  (**Offline-TM**) entscheidet  $\{a^i b^j | i \geq 1\}$  in Zeit  $O(n)$

1. Test, ob Eingabe die Form  $a^i b^j$  hat ( $O(n)$ )
2. Kopiere beim Lesen von  $a^i$  dieses Wort auf Arbeitsband  
 Beim Lesen von  $b^j$  lösche auf Arbeitsband nach links gehend, für jedes gelesene  $b$  ein  $a$ .  
 Akzeptiere bei gleichzeitigem Erreichen von  $\sqcup$ .

Gesamtlaufzeit:  $O(n)$

## 4.2 Klassen P und NP

Motivation für P: Präzisierung von „L ist effizient entscheidbar“

TM  $M$  ist polynomial zeitbeschränkt, falls ex. Polynom  $p(n)$  so dass  $M$   $O(p(n))$ -zeitbeschränkt ist.

**Bemerkung** Äquivalente Bedingung: ex.  $k \geq 0$  so dass  $M$   $O(n^k)$ -zeitbeschränkt ist.

**Definition 10**  $P =$  Klasse der durch polynomial zeitbeschränkte TM entscheidbaren Sprachen.

**Beispiel:**

1.  $L = \{a^i b^j | i \geq 1\} \quad L \in P$
2.  $L_{\mathbb{P}}$  = Menge der nat. Zahlen in Darstellung, die Primzahl sind.  
 Probieralgorithmus (für alle Teilerkandidaten):  
 Erzeuge sukzessiv alle Zahlen  $<$  Eingabe (bzw.  $\sqrt{\text{Eingabe}}$ ) und führe Division durch.  
 Anzahl der Teiler in der Länge  $n$  der Eingabe:  $2^n$ . für kein  $k$   $2^n \in O(n^k)$ .  
 (Es ex. eine  $O(2^n)$ -zeitbeschränkte TM)  
 inzwischen bekannt:  $L_{\mathbb{P}} \in P$

21.11.04

**P und NP**  $P =$  Klasse der durch polynomial zeitbeschränkte TM entscheidbaren Sprachen

**These von Cobham und Edmonds:**  $P$  enthält (ziemlich) genau die algorithmischen Probleme, die praktisch lösbar sind.

**Argumente dafür:**

1. Erfahrung mit polynomial zeitbeschränkten Algorithmen
2. Qualitätssprung gegenüber exponentiellen Laufzeiten
3. Bezug auf Turingmaschinen ist unwesentlich  
Präziser: Liegt ein polynomial zeitbeschränkter Algorithmus für irgendein sequentielles Standard-Rechnermodell vor, so ex. dafür auch polynomial zeitbeschränkte TM.

**Einwände:**

1. Aufgrund der Beschleunigung der Laufzeiten mit Entwicklung neuer Hardware verwischt die Unterscheidung polyn./ nicht polyn.  
**Nein:** Behandelbare Eingabegrößen bei fester Rechenzeit und Beschleunigung der Hardware um Faktor 1000:  
im Fall  $O(n^2)$ : 30 fache Eingabegröße  
Im Fall  $O(2^n)$ : bisherige Eingabegröße + 10
2. Es gibt Abstufungen, die die Unterscheidung aufweichen:
  - Polynome hohen Grades — mit großen Koeffizienten
  - Exponentialzeit-Algorithmen (worst case) mit besserem Algorithmus im Durchschnitt (Simplex-Verfahren)

### Probleme mit großem Suchraum

**Beispiel:**

Primzahlproblem (wie zuvor)

Für (Dualzahl-) Eingabelänge  $n$

Test für alle Teilerkandidaten (Anzahl  $2^n$ ), jeweils Division effizient.

**Beispiel:**

COLOR(3)

Gegeben: Endl. unger. Graph  $G = (V, E)$

Frage: Ex. 3-Färbung von  $G$ , d.h.  $c : V \rightarrow \{1, 2, 3\}$  mit  $c(u) \neq c(v)$  für  $(u, v) \in E$

Probieralgorithmus testet alle  $3^n$  viele Farbverteilungen  $c$  und überprüfe jeweils, ob korrekte Färbung vorliegt.

Zusammenfassung dieser Probleme in Klasse NP

**Vorbemerkung:**

Vorbereitung: Relation  $R \subseteq \Sigma^* \times \Gamma^*$  heißt polynomial entscheidbar, falls TM von Konf.  $q, u \sqcup v$  die Entscheidung „ $(u, v) \in R$ “ liefert in Laufzeit  $P(|u| + |v|)$  mit Polynom  $P$

**Definition 11**  $L \in NP \Leftrightarrow$  ex. polynomial entsch. Relation  $R \subseteq \Sigma^* \times \Gamma^*$  und Polynom  $q(n)$ , so dass  $w \in L \Leftrightarrow \exists v (|v| \leq q(|w|) \wedge (w, v) \in R)$   
„Teste Mitgliedschaft von  $w$  in  $L$  mit Probieren aller Lösungskandidaten  $v$ , deren Länge polynomial ind  $|w|$  beschränkt.“

**Bem.** Zumeist genügt  $q(n) = n$

**Beispiel:**

COLOR(3)

Gegeben: Ungerichteter Graph  $\underbrace{(V, E)}_G$

Frage: Ex. eine 3-Färbung von  $G$ , d.h. eine Partition von  $V$  in  $V_1, V_2, V_3$ , so dass für  $(u, v) \in E$  gilt  $u, v$  in Verschiedenen  $V_i$   $n = |V|$  Farbverteilung ist Wort der Länge  $n$  über Alphabet  $\Gamma = \{1, 2, 3\}$

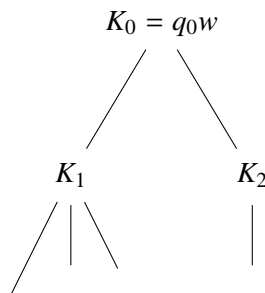
Nichtdet. Algorithmus erzeugt die  $3^{|V|}$  vielen Färbungen nichtdet. in  $n$  Schritten. Probieralgorithmus: Überprüfe alle Lösungskandidaten  $v$  (Anzahl:  $|\Gamma|^{q(|w|)}$ ) und teste jeweils, ob  $(w, v) \in R$

**Bem.:** Ist  $R$  durch  $p(n)$ -Zeitbeschr. TM entscheidbar, so ist Überprüfung von  $(w, v)$  in Zeit  $\underbrace{p(|w| + q(|w|))}$  möglich.  
also polynomial in  $|w|$

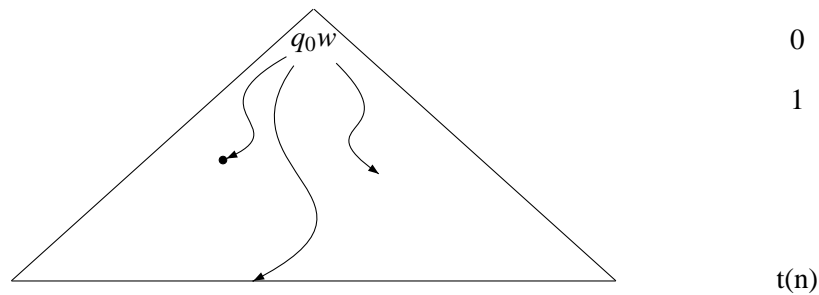
**Definition 12** Nichtdeterministische TM (NTM) ist wie zuvor definiert, kann jedoch für  $(q, a) \in \text{Zustandsmenge } Q \times \text{Arbeitsalphabet } \Gamma$  mehrere Turingzeilen enthalten.

Zu geg. Konfiguration  $K$  existieren dann gegebenenfalls mehrere Folgekonfigurationen.

Zu Eingabe  $w$  und Anfangskonf.  $q_0w$  erhalte Konfigurationsbaum



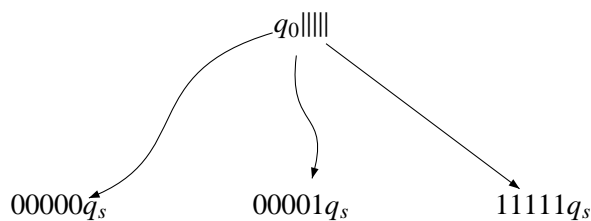
NTM heißt  $t(n)$ -zeitbeschränkt, falls jeder Zweig des Konfigurationsbaums endet nach  $\leq t(n)$  Schritten, jeweils für Eingabe  $w$  der Länge  $n$



NTM akzeptiert Eingabewort  $w$ , falls für  $\geq$  eine Stopkonf. des Konf.baums zu  $w$  das Ergebnis 1 („ja“) lautet.

**Beispiel:**

NTM wandelt ein Wort  $|w|^n$  in alle Bitwörter der Länge  $n$  um.



**Definition 13**  $L \in NP \Leftrightarrow$  ex. polynomial zeitbeschränkte NTM, die  $w$  akz. gdw.  $w \in L$

**Satz 4.1:** Beide Def. von NP sind äquivalent

**Beweis:**

Zu  $p(n)$ -zeitbeschränkte NTM  $M$  finde  $q(n)$  und polynomial entsch. Relation  $R$  gemäß 1. Def.

$M$  entscheide  $L(w, v) \in R \Leftrightarrow v$  ist eine Konfigurationsfolge  $\#q_0w\#\dots\#q_s1\sqcup\dots\#$  von  $M$  zu Eingabe  $w$

$M$  akz.  $w \Leftrightarrow \exists v((w, v) \in R$  und Schrittzahl in  $v$  ist  $\leq p(n)$ )

$|v| \leq (|w| + 2 + p(|w|)) \cdot p(n) = q(n)$

11.1.05

**Beispiel (SAT (Satisfiability, Erfüllbarkeit aussagenlog. Ausdrücken (boolesche Ausdrücke))):**

Aufbau der Formeln:

- Aussagenvariablen  $X_0, X_1, X_{10}, X_{11}, \dots$   $X_i$  mit  $i$  in Binärdarstellung
- Junktoren:  $\neg, \wedge, \vee$

Bsp:  $(X_1 \vee \neg X_1) \wedge \neg X_{11}$

Für Ausdrücke  $\varphi(X_1, \dots, X_n)$  ist eine Belegung einer Funktion  $B : \{X_1, \dots, X_n\} \rightarrow \{0, 1\}$

$B$  erfüllt  $\varphi(X_1, \dots, X_n)$ , wenn bei der Substitution von  $X_i$  durch  $B(X_i)$  mit üblicher Auswertung von  $\varphi$  der Wert 1 sich ergibt.

$\varphi$  erfüllbar, falls ex. Belegung  $B$ , die  $\varphi$  erfüllt.

**SAT: Gegeben:** aussagenlogischer Ausdruck  $\varphi$

**Frage:** ist  $\varphi$  erfüllbar?

**Beispiel:**

$(X1 \vee \neg X1) \wedge \neg X11$  ist erfüllbar, mit folgendem  $B$ :  $B(X1) = 0$  und  $B(X11) = 0$

**Bemerkung:**

$$SAT \in NP$$

Angabe einer polynomialzeitbeschränkten NTM  $M$

$M$  läuft durch Eingabe und ersetzt jedes  $X$  durch 0 oder 1 (nichtdet.)

$$\begin{array}{cccccc} q & x & \underline{0} & R & q \\ q & x & \underline{1} & R & q \\ q & \wedge & \wedge & R & q \\ q & ( & ( & R & q \\ & & \vdots & & \end{array}$$

Dann wird deterministisch überprüft, ob

- für gleiche  $X$ -Indizes gleiches Bit gewählt
- bei Auswertung Wert 1 herauskommt.

Im folgenden wird als Eingabe für SAT angenommen, dass  $\varphi$  in KNF (konjunktiver Normalform) vorliegt.

**KNF:**

$$\begin{aligned} \varphi &= c_1 \wedge \cdots \wedge c_m && c_i \text{ Klausel} \\ c_i &= l_{i1} \vee \cdots \vee l_{in_i} && l_{ij} \text{ Literal} \\ l_{ij} &= xk \text{ oder } l_{ij} = \neg Xk && Xk \end{aligned}$$

Duale Form: DNF (disjunktive Normalform)  $\varphi = d_1 \vee \cdots \vee d_m$

$$d_i = l_{i1} \wedge \cdots \wedge l_{in_i}$$

**Klar:**  $P \subseteq NP$        $P \stackrel{?}{\subseteq} NP$

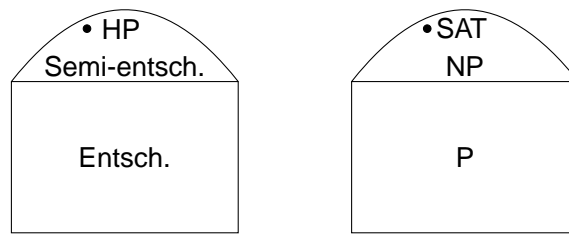
$P = NP$  würde bedeuten: Zu jedem Probieralgorithmus existiert ein effizienter Algorithmus.

**Teillösungen:** Die allgemeine Frage  $P \stackrel{?}{\subseteq} NP$  kann man auf Studium konkreter Beispielprobleme reduzieren.

$$P \stackrel{?}{\subseteq} NP \Leftrightarrow SAT \in NP \setminus P \quad (SAT \notin P)$$

**Bemerkung:**

$$SAT \in NP$$



## 5 NP-Vollständigkeit

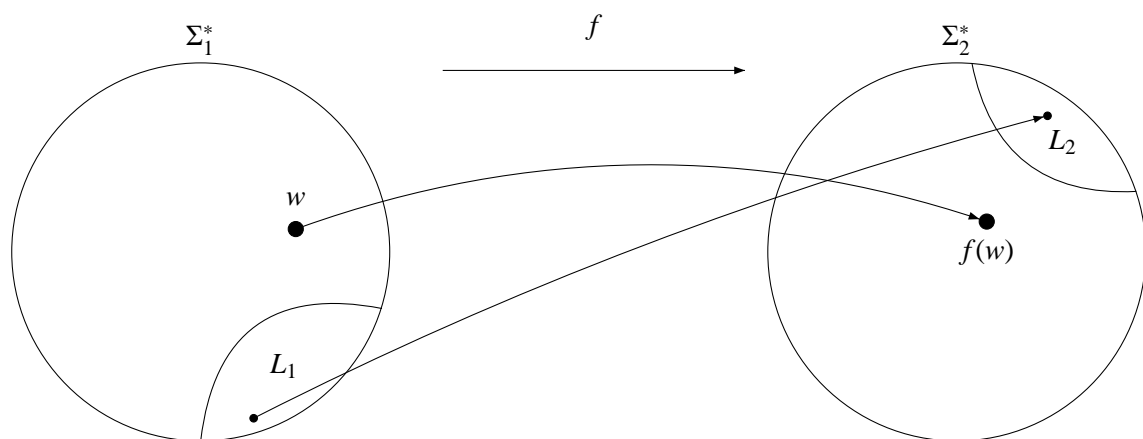
Problem kodiert durch Sprache  $L$  | Gegeben:  $w \in \Sigma^*$   
 | Frage:  $w \in L$ ?

Vergleich der Komplexität von Sprachen  $L_1 \subseteq \Sigma_1^*, L_2 \subseteq \Sigma_2^*$

$L_1 \leq L_2$  ( $L_1$  polynomzeit-reduzierbar auf  $L_2$ ):  $\Leftrightarrow$

ex. eine polynomzeitberechenbare Funktion  $f : \Sigma_1^* \rightarrow \Sigma_2^*$  mit  $w \in L_1 \Leftrightarrow f(w) \in L_2$  (\*)

$f : \Sigma_1^* \rightarrow \Sigma_2^*$  polynomzeitberechenbar gdw. es ex. polynomzeitbeschr. TM die  $f$  berechnet.



### Bemerkung:

$L_1 \leq_p L_2, L_2 \in P \Rightarrow L_1 \in P$

### Beweis:

Sei  $M_f$  TM die  $f$  berechnet,  $p(n)$ -zeitbeschr.

$M_2$  TM die  $L_2$  entscheidet,  $q(n)$ -zeitbeschr.

$p, q$  Polynome

Folgende TM entscheidet  $L_1$ : Aus  $w \in \Sigma_1^*$  stelle mit  $M_f$   $f(w)$  her.

Auf  $f(w)$  wende  $M_2$  an und übernehme Antwort, wegen (\*) korrekt.

Zeitabschätzung für  $M_1$  (angenommen  $M_f$  liefert neben  $f(w)$  nur  $\perp$ )

$M_1 : w \xrightarrow{M_f} f(w)$  in Zeit  $p(|w|)$   
 $|f(w)| \leq |w| + p(|w|)$   
 $f(w) \xrightarrow{M_2} \text{Ja/Nein}$  in Zeit  $q(|w| + p(|w|))$ .  
 Zeit:  $p(|w|) + q(|w| + p(|w|))$

**Definition 14**  $L_0$  NP-vollständig:  $\Leftrightarrow$

1.  $L_0 \in NP$
2. für alle  $L \in NP : L \leq_p L_0$

**Satz 5.1 (Cook,Levin 1971):** SAT ist NP-vollständig

**Bemerkung:**

Sei  $L_0$  NP-vollständig. Dann  $P \subsetneq NP \Leftrightarrow L_0 \notin P[L_0 \in NP \setminus P]$

**Beweis:**

$\Leftarrow$  trivial

$\Rightarrow$  Zeige äquivalente Aussage  $L_0 \in P \Rightarrow P = NP$  genügt beh.  $NP \subseteq P$  unter Vorauss.  $L_0$  NP-vollst.,  $L_0 \in P$

Sei  $L \in NP$   $L \leq_p L_0$  Nach Bem.  $L \in P$

14.01.05

**Genauer:**

1.  $SAT \in NP$
2. f. alle  $L \in NP$   $L \leq_p SAT$   
 $L_1 \subseteq \Sigma_1^*, L_2 \subseteq \Sigma_2^*$   
 $L_1 \leq_p L_2 \Leftrightarrow$  ex. polynomzeitber. Funktion  $f : \Sigma_1^* \rightarrow \Sigma_2^*$  mit  $w \in L \Leftrightarrow f(w) \in L_2$  f. alle  $w \in \Sigma_1^*$

**Beweis:**

Finde polynomzeitber. Funktion  $f : \Sigma^* \rightarrow \Sigma_{SAT}^*$  mit  $w \in L \Leftrightarrow \underbrace{f(w) \in SAT}_{f(w) \text{ ist erfüllb. aussagenlog. Ausdruck}}$  f. alle

$w \in \Sigma^*$

$\Sigma_{SAT} = \{x, 0, 1, \neg, \wedge, \vee, (, )\}$  Umformulierung der linken Seite: Benutze Voraussetzung  $L \in NP$ .

Wähle NTM  $M = (Q, \Sigma, \Gamma, q_0, q_s, \Delta)$  zeitbeschr. durch Polynom  $p(n)$

$w \in L \Leftrightarrow$  ex.  $M$ -Konfigurationsfolge  $K_0, K_1, \dots, K_r$  mit  $K_0 = q_0 w$

$K_{i+1}$  entsteht aus  $K_i$  vermöge  $\Delta$ ,  $K_r$  akzeptierend

$R \leq p(|w|)$

Repräsentation der Konfigurationsfolge durch Matrix

	$p( w ) - 1 \dots$	-2	-1	0	1	2	3	4	5	$\dots$	$p( w ) + 1$	$p(\neg w) + 2$	
0		$\sqcup$	$\sqcup$	$q_0$	$q_1$	$a_{i_2}$	$a_{i_3}$	$\sqcup$	$\sqcup$		#		$\Leftarrow K_0$
1													$\Leftarrow K_1$
										$q_s$	1	$\sqcup$	
										$q_s$	1	$\sqcup$	
$p( w )$													

Kopiere akzeptierende Konfiguration für nachfolgende Zeilen. Einträge aus der Menge  $C = \Gamma \cup Q \cup \{\#\}$

Also:  $w \in L \Leftrightarrow \text{ex. } I \times J\text{-Matrix wie aufgegeben mit:}$

1. Zeile repräsentiert Konf.  $q_0 w$  Zeilenwechsel gemäß TM  $M$  ex. Zeile mit Segment  $q_s 1 \sqcup$

$$I = \{0, \dots, p(|w|)\} \quad J = \{-p(|w|) - 1, \dots, p(|w|) + 2\}$$

Führe für  $(i, j) \in I \times J$  und  $c \in C$  Variable  $x_{i,j,c}$  ein (wahr falls  $c$  an Position  $(i, j)$  steht)

Ziel: Aufbau eines Ausdrucks  $\varphi_w$ , der die Existenz einer Matrix mit diesen Eigenschaften formuliert.

[ Setze dann  $f(w) := \varphi_w$  ]

$$\phi_w := \varphi_{\text{konsistent}} \wedge \varphi_{\text{anfang}} \wedge \varphi_{\text{übergang}} \wedge \varphi_{\text{ende}}$$

$$\varphi_{\text{konsistent}} : \bigwedge_{(i,j) \in I \times J} \bigvee_{c \in C} x_{i,j,c} \wedge \bigwedge_{(i,j) \in I \times J} \bigwedge_{c \neq c'} \underbrace{\neg(x_{i,j,c} \wedge x_{i,j,c'})}_{\neg x_{i,j,c} \vee \neg x_{i,j,c'}}$$

$$\varphi_{\text{anfang}} : x_{0,-p(|w|),\#} \wedge x_{0,-p(|w|),\sqcup} \wedge \dots \wedge x_{0,0,q_0} \wedge x_{0,1,a_n} \wedge x_{0,n+1,\sqcup} \wedge \dots \wedge x_{0,p(|w|)+1,\sqcup} \wedge x_{0,p(|w|)+2,\#}$$

$$w = a_{i_1} \dots a_{i_n}$$

$$\varphi_{\text{ende}} : \bigvee_{(i,j) \in I \times J} (x_{i,j,q_s} \wedge x_{i,j+1,1} \wedge x_{i,j+2,\sqcup})$$

**Zu Übergang** Illustration für einen R-Schritt  $p a a' R q$

$$\begin{array}{ccc} p a b & & b p a \\ a' q b & \text{analog für } p a a' L q & q b a' \end{array}$$

$2 \times 3$ -Matrix über  $C$  heißt zulässig, falls in Matrix zu  $M$  als Submatrix formal möglich.

$$\varphi_{\text{übergang}} : \bigwedge_{i,j} \bigvee_{\substack{c_1 \ c_2 \ c_3 \\ c_4 \ c_5 \ c_6}} x_{i,j,c_2} \wedge x_{i,j-1,c_1} \wedge x_{i,j+1,c_3} \wedge x_{i+1,c_4} \wedge x_{i+1,c_5} \wedge x_{i+1,c_6} \quad \text{Zulaessig}$$

18. 1. 05

$x_{i,j,c}$  "Aus Pos.  $(i, j)$  steht  $c$ "

$$(x_1 \wedge x_2) \vee (x_3 \wedge x_4) \Leftrightarrow (x_1 \vee x_3) \wedge (x_2 \vee x_3) \wedge (x_1 \vee x_4) \wedge (x_2 \vee x_4)$$

$\varphi_{\text{ende}}$  durch Ergänzung von  $M$  durch Testschritte am Ende reduzierbar auf Form

$$\bigvee_{(i,j)} x_{i,j,q_s}$$

$\varphi_w$  in KNF notierbar.

**Abschätzung der Länge von  $\varphi_w$ :** Länge von  $x_{i,j,c} \leq 2 + 2(\log(p(|w|) + 1))$

Jede der Formeln  $\varphi_{\text{konsistent}}, \varphi_{\text{anfang}}, \varphi_{\text{ende}}, \varphi_{\text{übergang}}$  hat Länge  $|I| \cdot |J| \cdot \text{const} \cdot \underbrace{q(|w|)}_{\text{eine Var.}}$  insgesamt polynomiale Länge.

Fleißarbeit zeigt:  $w \mapsto \varphi_w$  polynomzeit-berechenbar durch TM. Weitere NP-vollständige Probleme: SAT(3), COLOR(3).

Methode: Um zu zeigen, dass  $L_0$  NP-vollst., genügt es, zu zeigen:

1.  $L_0 \in NP$
2. Für geeignete NP-vollst. Sprache  $L_1$  gilt:  $L_1 \leq_p L_0$

Dann:  $\forall L \in NP \underbrace{L \leq_p L_1 \leq_p L_0}_{\leq_p}$

SAT(3): **Gegeben:** KNF-Ausdruck  $\varphi$  mit je 3 Literalen pro Klausel.

**Frage:** Ist  $\varphi$  erfüllbar?

**Satz 5.2:** SAT(3) ist NP-vollständig.

**Beweis:**

1.  $SAT(3) \in NP$  klar, da Spezialfall von SAT
2.  $SAT \leq_p SAT(3)$   
gesucht: Transformation KNF-Ausdruck  $\varphi$  mit  $\varphi$  erfüllbar  $\Leftrightarrow \varphi'$  erfüllbar und polynomzeitberechenbar.

Ansatz für Transformation: Klausel  $l_1 \vee \dots \vee l_k (k > 3)$  zu transformieren in Konjunktion von Klauseln mit je 3 Literalen.

Es ergibt sich aus  $\varphi$  ein 3-KNF Ausdruck  $\varphi'$

Benutze für Umformung von  $l_1 \vee \dots \vee l_k$  Hilfsvariablen  $y_1, \dots, y_{k-3}$

$l_1 \vee \dots \vee l_k \mapsto (l_1 \vee l_2 \vee y_1) \wedge (\neg y_1 \vee l_3 \vee y_2) \wedge \dots \wedge (\neg y_{k-4} \vee l_{k-2} \vee y_{k-3}) \wedge (\neg y_{k-3} \vee l_{k-1} \vee l_k)$

**Satz 5.3:** COLOR(3) ist NP-vollständig

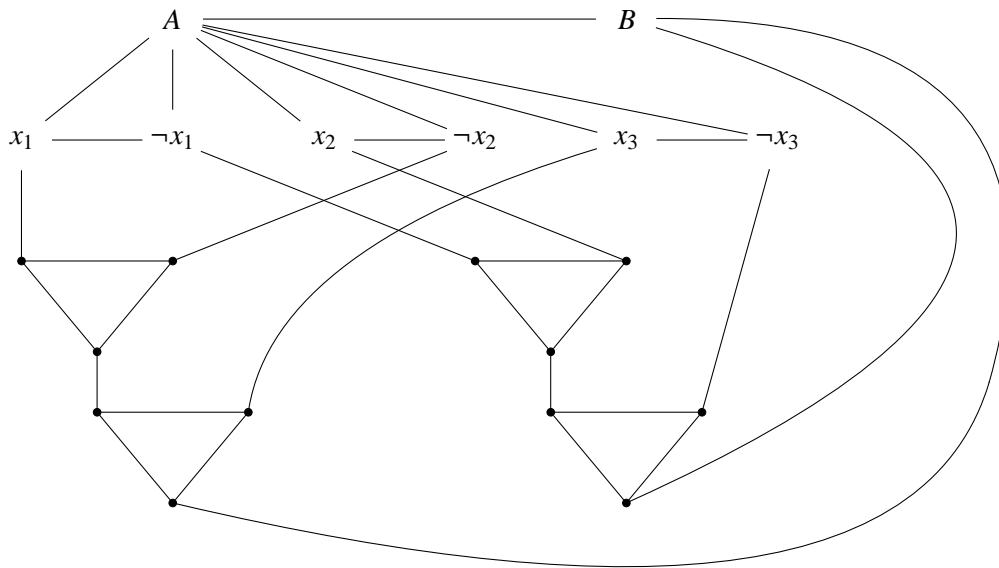
**Beweis:**

1. COLOR(3)  $\in NP$  (gezeigt)
2.  $SAT(3) \leq_p COLOR(3)$

Zu finden: polynomzeitberechenbare Transformation: 3-KNF Ausdruck  $\varphi \mapsto \text{Graph } G_\varphi = (V_\varphi, E_\varphi)$  mit  $\varphi$  erfüllbar  $\Leftrightarrow G_\varphi$  3-färbbar. Sei  $\varphi = c_1 \wedge \dots \wedge c_m$ , vorkommende Variablen:  $x_1, \dots, x_n$

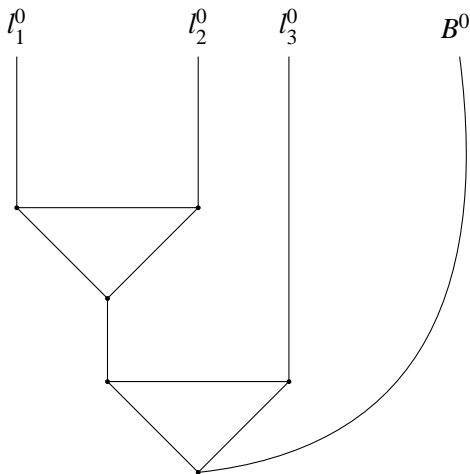
$c_i$  hat Form  $(l_{ij} \vee l_{i2} \vee l_{i3} \quad l_{ij} = x_k \text{ oder } = \neg x_k)$

$G_\varphi$  hat zwei Knoten  $A, B, x_1, \neg x_1, \dots, x_n, \neg x_n$  weitere 6 Knoten je Klausel.  $\varphi : (x_1 \vee \neg x_2 \vee x_3) \wedge (\neg x_1 \vee x_2 \vee \neg x_3)$



21.1.05

Von der Graphenfärbung zu erfüllender Belegung: oBdA. Farbe 2 auf A, 0 auf B Sicherzustellen: kein Klauselgraph ist an Literalen mit dreimal Farbe 0 angebunden. Sonst:



Zum Problem **CLIQUE**.

Gegeben: Graph  $G$ , Zahl  $k$

Frage: Existiert Clique der Größe  $k$  in  $G$  d.h. eine Menge von  $k$  Knoten, die paarweise untereinander durch Kanten verbunden sind.

**Satz 5.4:** CLIQUE ist NP-vollständig

**Beachte:** CLIQUE  $\neq$  CLIQUE( $k$ )

CLIQUE( $k$ ): Gegeben: Graph  $G$

Frage: Ex. Clique der Größe  $k$  in  $G$ ?

Üb: CLIQUE( $k$ ) ist durch polynomzeit-Algorithmus  $A_k$  lösbar, also in P.

**Beweis:**

Zeige 1. CLIQUE  $\in$  NP, 2. SAT(3)  $\leq_p$  CLIQUE

Zu 1. Rate Teilmenge der Größe  $k$   
Überprüfe, ob Clique vorliegt.

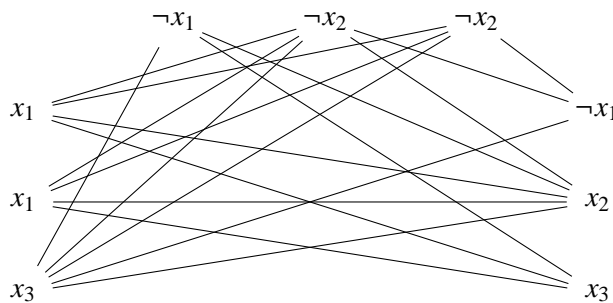
Zu 2. Definiere Transf. 3-KNF-Formel  $\varphi \mapsto G_\varphi, k_\varphi$

$\varphi$  erfüllbar  $\Leftrightarrow G_\varphi$  hat Clique mit  $k_\varphi$  vielen Knoten.

$\varphi \mapsto G_\varphi, k_\varphi$

$\varphi = (x_1 \vee \neg x_1 \vee x_3) \wedge (\neg x_1 \vee \neg x_2 \vee \neg x_2) \wedge (\neg x_1 \vee x_2 \vee x_3)$

$k_\varphi = 3$  (Anzahl der Klauseln)



Knoten: Vorkommen der Literale

Kanten: jeweils zwischen Knoten, außer

- gehören zu gleicher Klausel
- untereinander komplementär

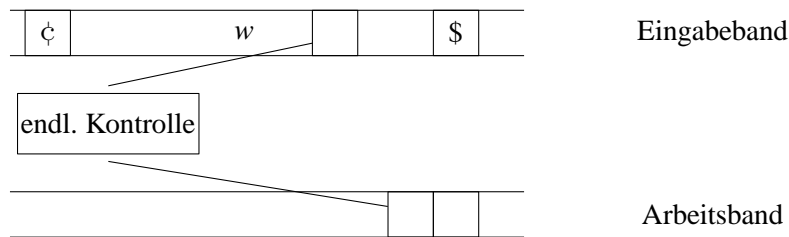
Erfüllende Belegung liefert Clique mit 3 Knoten und umgekehrt Dreierclique liefert erfüllende Belegung.

## 5.1 Platzkomplexität

**Idee** für TM: Zähle die Anzahl der besuchten Felder während einer Berechnung.

Unterscheide Feldzugriffe zwecks Lesen der Eingabe und Feldzugriffen zwecks Rechnung.

**Offline-Turingmaschine** (Lese-) Eingabeband



Befehlszeile:  $p a_1 b_1 L/R/N b_2 L/R/N q$

Im Zustand  $p$  mit  $a_1$  auf Eingabeband-Feld,  $b_2$  auf Arbeitsbandfeld. Bewege Kopf auf Eingabeband, drucke auf Arbeitsband, bewege Kopf auf Arbeitsband gemäß der drei folgenden Komponenten und gehe nach  $q$  25.1.05

**Beispiel:**

$L = \{a^i b^j | i > 0\}$  Offline TM notiert in Binärdarstellung jeweils die Anzahl der bisher gelesenen Buchstaben  $a$  (für jedes  $a$  Erhöhung der Binärzahl um 1) und substrahiert anschließend für jedes  $b$  um 1.  $\lceil \log n \rceil + 1$

**Beispiel:**

Erreichbarkeitsproblem über gerichteten Graphen

Kodiere  $G = (V, E)$  durch Kantenliste, Knotennamen sind Binärzahlen.

Gegeben:  $G$ , Knoten  $s, t$

Frage: Existiert in  $G$  ein Pfad von  $s$  nach  $t$ ?

Nichtdet. Offline-TM entscheidet dieses Problem mit Platzverbrauch  $2 \log |V| + 1$

**Ansatz:** Raten eines Pfades von  $s$  nach  $t$  durch sukzessives Notieren von Knotennamen auf dem Arbeitsband (jeweils Test, ob gültige Kante erzeugt vom letzten Knoten aus und ob  $t$  erreichbar ist).

Zunächst Platzbedarf  $|V| \cdot (\log |V| + 1)$

**Verfeinerung:** Überschreiben des jeweils vorletzten Knoten durch den neubesuchten. Platzbedarf:  $2 \cdot \log |V| + 1$

**Notationen** DLOG = Klasse der Sprachen die durch DTM an Platz  $O(\log(n))$  entschieden werden können. ( $L$  von Bsp. 1 gehört zu DLOG)

NLOG analog für NTM. Sprache der Kodierung des Erreichbarkeitsproblems gehört zu NLOG.

(D/N)SPACE analog für Polyom  $p(n)$  an Stelle  $\log(n)$

**Satz 5.5 (Vergleich von Platz- zu Zeitkomplexität):** Eine  $f(n)$ -platzbeschränkte Offline-TM kann durch eine  $2^{O(f)}$ -zeitbeschr. TM simuliert werden.

**Ansatz:** Abschätzung: Anzahl der verschiedenen möglichen Konfigurationen der gegebenen TM  $M$   
 $M$ -Konfiguration auf Eingabewort  $w$  der Länge  $n$ : { Zustand, AF des Eingabebandes, Beschriftung des Arbeitsbandes, AF des Arbeitsbandes.

Bei Zustandsmenge  $Q$ , Arbeitsalphabet  $\Gamma$  ist die Anzahl  $\leq |Q| \cdot (n + 2) \cdot |\Gamma|^{f(n)} \cdot f(n) \in 2^{O(f(n))}$

Termination muss innerhalb dieser Schrittzahl erfolgen (sonst Konfigurationswiederholung, Zyklus).

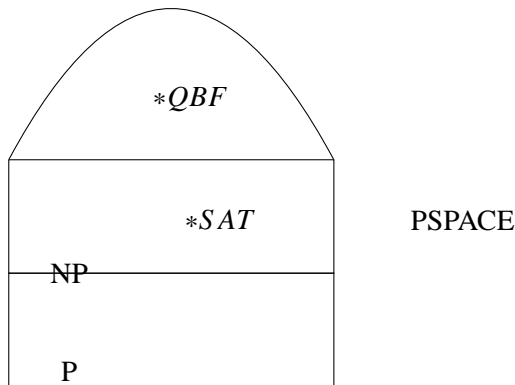
**Bemerkung:**

Eine  $f(n)$ -zeitbeschränkte TM kann durch eine  $f(n)$ -platzbeschränkte (Offline-)TM simuliert werden.

**Zur Klasse PSPACE** Vorbemerkung (Satz von Savitch) Eine polynomial  $p(n)$ -platzbeschr. Offline-NTM kann durch eine  $O(p^2(n))$ -platzbeschr. Offline-DTM simuliert werden. Folglich ist der Bezug auf Det./Nichtdet.

in Def. von PSPACE unwesentlich. Unterstelle von jetzt an DTM.

**Bem.**  $P \subseteq NP \subseteq PSPACE$



**Zum Problem QBF** Quantifizierte Boolesche Formeln (in pränex-Normalform) haben die Form  $Q_1x_1Q_2x_2 \dots Q_nx_n\varphi(x_1, \dots, x_n)$  mit  $Q_i \in \{\exists, \forall\}$  aussagenlog. Formel.

$\exists x_i\psi$  besagt "es ex. Wahrheitswert (0 oder 1) so dass  $\psi$  wahr wird, wenn dieser für  $x_i$  eingesetzt wird."

$\exists x_i\psi(\dots x_i \dots) \leftrightarrow \psi(\dots 0 \dots) \vee \psi(\dots 1 \dots)$  Analog für  $\forall$  mit  $\wedge$  an Stelle von  $\vee$ .

**Bemerkung:**

In QBF in pränex-Normalform wird keine Belegung für  $x_1, \dots, x_n$  benötigt, um Wahrheitswert zu ermitteln, Formel ist selbst wahr oder falsch.

**Illustration**  $\varphi(x_1, \dots, x_n)$  aussagenlog. Formeln benötigt Belegung  $\exists x_1 \dots \exists x_n \varphi(x_1 \dots x_n)$  ist wahr  $\Leftrightarrow \varphi$  erfüllbar.

falsch  $\Leftrightarrow \varphi$  nicht erfüllbar.

**QBF-Problem:** Gegeben QBF  $\psi$ .

Frage: Ist  $\psi$  wahr?

**Bemerkung:** Spezialfall  $\exists x_1 \dots \exists x_n \varphi(x_1 \dots x_n)$  ist das Problem SAT.

**Beispiel:**

$\forall x_1 \exists x_2 ((x_1 \vee x_2) \wedge (\neg x_1 \vee \neg x_2))$  wahr. (Zu  $x_1$  wähle  $x_2 = \neg x_1$ )

$\exists x_1 \forall x_2 ((x_1 \vee x_2) \wedge (\neg x_1 \vee \neg x_2))$  falsch.

Lösung von QBF in polynomialem Platz.

Auswertungsalgorithmus bei Eingabe  $\varphi :=$

- Falls keine Quantoren in  $\varphi$ . Werte  $\varphi$  aus.
- Falls  $\varphi = \exists x_i \psi(x_1 \dots)$ , rufe alg. wieder auf für  $\psi(0, \dots), \psi(1, \dots)$  und führe Ergebnisse mit  $\vee$  zursammen.
- Falls  $\varphi = \forall x_i \psi(x_1 \dots)$ , rufe alg. wieder auf für  $\psi(0, \dots), \psi(1, \dots)$  und führe Ergebnisse mit  $\wedge$  zursammen. Platzbedarf für Hilfsspeicher (Stack) der Länge  $c \cdot n$  ( $n$  Anzahl der  $x_i$ ) also polynomial platzbeschr. in  $|\varphi|$

**Illustration:**  $\exists x_1 \forall x_2 \exists x_3 \underbrace{\varphi(x_1, x_2, x_3)}_{(x_2 \vee x_3) \wedge \neg x_1}$

$\vee 0 \wedge 0 \vee 0$   
 $\vee 0 \wedge 0 \vee F 1$   
 $\vee 0 \wedge 0 \vee F W$

Verwaltung der Zwischenergebnisse in Stack der Länge  $\leq 3 \cdot n$

$$DLOG \subseteq NLOG \subseteq P \subseteq NP \subseteq DSPACE$$

**Satz 5.6 (Platzhierarchiesatz):** Sei  $f(n) \geq \log(n)$ .

Es gibt ein Problem, das durch eine  $f^2(n)$  platzbeschränkte det. TM entschieden wird, aber nicht durch eine  $f(n)$  platzbeschränkte det. TM.

**Korollar:**  $DLOG \subsetneq DSPACE$  (sonst ist nichts bekannt!).

**Satz 5.7 (Savitch):**

$$DSPACE = NSPACE$$

Deshalb wird in der Literatur nur  $SPACE$  verwendet.

**Beweis:**

Wir zeigen: Zu jeder durch  $p(n)$  platzbeschränkten NTM  $M$  existiert eine  $p(n)^2$  platzbeschränkte DTM  $M'$  mit  $\forall w \quad w \in L(M) \Leftrightarrow w \in L(M')$ .

Sei  $M$  eine  $p(n)$  platzbeschränkte NTM.

Wähle  $c = |Q_M| \cdot n \cdot p(n) \cdot |\Gamma_M|^{p(n)}$  Das ist die Anzahl der möglichen Konfigurationen bei Eingabe  $w$  der Länge  $n$ .

Falls  $M$   $w$  akzeptiert, dann ist das in  $\leq c$  Schritten möglich.

Sei  $k_0$  die Startkonfiguration von  $M$ ;  $M'$  überprüft für jede Stoppkonfiguration  $k$  ob  $M : k_0 \vdash^{\leq c} k$ .

Wähle  $m$  so, dass  $s^m \geq c$ . Dann gilt  $M : \vdash^{\leq 2^m} k_s \Leftrightarrow$ . Dies wird rekursiv überprüft.

**Algorithmus für  $M'$ :**

Eingabe: Wort  $w$  der Länge  $n$

1. Berechne  $c$
2. Sei  $k_0$  die Anfangskonfiguration. von  $M$
3. Für jede mögliche Endkonf.  $k_s$  überprüfe ob  $R(k_0, k_s, c)$   
Falls einmal "1" akzeptiere  $w$

**Prozedur R:**

Eingabe: Konfigurationen  $k_1, k_2$ , Zahl  $i$ ;

Ausgabe: 0/1

- a) Falls  $i = 0$  überprüfe  $k_1 = k_2$  bzw.  $M : k_1 \vdash^1 k_2$ , Falls ja  $\rightarrow$  Ausgabe 1.
- b) Falls  $i > 1$ : Für alle Konf.  $k$  von  $M$ : Falls  $R(k_1, k, i-1) = 1$  und  $R(k, k_2, i-1) = 1$  dann  $\rightarrow$  Ausgabe 1
- c) Ausgabe 0

Der Algorithmus ist korrekt.

**Platzverbrauch:** Jede Konfiguration von  $M : \leq p(n) + 2 \log(p(n)) + \log(|Q|) \in O(p(n))$  viel Platz.

Vor jedem Argument von  $R$  in Zeile (b) muss  $k_1, k_2, i$  und  $k$  gespeichert werden  $\rightarrow O(p(n))$  viel Platz.

Die Anzahl der rek. Aufrufe ist beschränkt durch  $n \leq \lceil \log(c) \rceil \in O(p(n))$

Insgesamt Platzkomplexität  $O(p(n)^2)$

## 5.2 Approximationsalgorithmen:

Problemtyp: Optimierungsproblem (statt Entscheidungsproblem)

### Beispiel (Cliquesproblem):

Gegeben: Graph  $G = (V, E)$

Gesucht: möglichst große Clique in  $G$

Zulässige Lösungen: Cliques (Teilmengen  $M \subseteq V$  die Clique bilden).

Der Wert von  $M$  :  $c(M)$  ist zu maximieren.

**Allgemeine Fassung:** (Version Maximierungsproblem): Zu Eingabe sind zulässige Lösungen definiert ( $M$ ).

$M$  hat jeweils Wert  $c(M)$  (Hier Wertebereich  $\mathbb{N}$ ).

Gesucht ist Lösung maximalen Wertes.

Zu Eingabe berechnet ein Approximationsalgorithmus eine zulässige Lösung  $M$ .

Vergleich mit Wert einer Lösung  $M_0$  maximalen Wertes.

Approximationsgüte:  $\frac{c(M_0)}{c(M)} \leq 1 + \epsilon$

Falls  $\frac{c(M_0)}{c(M)} \leq 2$ : Gefundene Lösung hat Wert mindestens der Hälfte der optimalen.

**Allgemeines Ziel:** Finde Algorithmen, die

- polynomiale Laufzeit haben.
- garantierte Approximationsgüte haben. ( $1 + \epsilon$  für kleines  $\epsilon$ ).

Illustration hier anhand des einfachen Rucksackproblems RS.

Gegeben:  $w_1, \dots, w_n, b \in \mathbb{N}_+$

Gesucht:  $I \subseteq \{1, \dots, n\}$  mit  $\sum_{i \in I} w_i$  möglichs groß, mit Bedingung  $\sum_{i \in I} w_i \leq b$

Hier zulässige Lösung  $I \subseteq \{1, \dots, n\}$  mit  $\sum_{i \in I} w_i \leq b$

Wert  $c(I) = \sum_{i \in I} w_i$ .

Spezialfall: Entscheidungsproblem RS

Gegeben:  $w_1, \dots, w_n, b$

Frage: ex.  $I \subseteq \{1, \dots, n\}$  mit  $\sum_{i \in I} w_i = b$ ?

**Satz 5.8 (RS ist NP-vollständig):** einfach: RS  $\in$  NP.

Zur NP-Vollständigkeit:  $\text{SAT}(3) \leq_p \text{RS}$ .

Finde Transformation (polynomzeit-beschr.) 3-KNF-Ausdruck  $\varphi \mapsto (w_1, \dots, w_n, b)_\varphi$  mit  $\varphi$  erfüllbar  $\Leftrightarrow$  ex.  $I \subseteq \{1, \dots, n\}$  mit  $\sum_{i \in I} w_i = b$ .

### Beispiel:

$\varphi = (x_1 \vee \neg x_2 \vee x_3) \wedge (\neg x_1 \vee x_2 \vee \neg x_4) \wedge (\neg x_1 \vee \neg x_3 \vee \neg x_3)$

Def. Vektoren  $v_1, \overline{v}_1, \dots, v_4, \overline{v}_4$  wie folgt:

$v_1 = b_1 b_2 b_3$  mit  $b_i = 1$  falls  $x_i$  kommt in Klausel Nr.  $i$  vor. (und macht Klausel somit wahr)..

$\overline{v}_1$  analog mit  $\neg x_1$ , entsprechend für  $v_2, \overline{v}_2, \dots, v_4, \overline{v}_4$

$v_1 = 1\ 0\ 0\ 1\ 0\ 0\ 0$      $\overline{v}_1 = 0\ 1\ 1\ 1\ 0\ 0\ 0$

$v_2 = 0\ 1\ 0\ 0\ 1\ 0\ 0$      $\overline{v}_2 = 1\ 0\ 0\ 0\ 1\ 0\ 0$

$v_3 = 1\ 0\ 0\ 0\ 0\ 1\ 0$      $\overline{v}_3 = 0\ 0\ 1\ 0\ 0\ 1\ 0$

$v_4 = 0\ 0\ 0\ 0\ 0\ 0\ 1$      $\overline{v}_4 = 0\ 1\ 0\ 0\ 0\ 0\ 1$

Belegung der  $x_i$  durch Wahrheitswerte liefert vier Vektoren:

$x_1, x_2, x_3 \mapsto \text{wahr}$      $x_4 \mapsto \text{falsch}$     Vektoren:  $v_1, v_2, v_3, \bar{v}_4$ .

$\varphi$  wird wahr gdw. Summe der zugehörigen Vektoren wird in jeder Komponente  $\geq 1$

$\varphi$  erfüllbar gdw. ex. Teilmenge der  $v_1, \bar{v}_i$  mit jeweils genau einem Vektor aus  $\{v_i, \bar{v}_i\}$  und Wert  $\geq 1$  in jeder Komponente der Summe.

**Verfeinerung 1:** Ergänze  $\bar{v}_i$  um 4 Bits, mit i-tem Bit = 1 für  $v_i, \bar{v}_i$ .

Verlange Summenwert 1 in jeder dieser Komponenten.

Gesichert: Bei Auswahl der  $v_i, \bar{v}_i$  wird zu jedem  $i$  nur  $v_i$  oder  $\bar{v}_i$  ausgewählt.

Zeilensummenwert:

(\*)  $\geq 1 \geq 1 \geq 1 \ 1 \ 1 \ 1 \ 1$

(\*) erreichbar *Left-right arrow*  $\varphi$  erfüllbar

**Verfeinerung 2:** Hilfsvektoren zur Ergänzung der ersten drei Komponenten auf den Wert 4 ausgehend

von Wert  $\geq 1$      $u_1 = 1 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0$      $u_3 = 0 \ 1 \ 0 \ 0 \ 0 \ 0 \ 0$      $u_5 = 0 \ 0 \ 1 \ 0 \ 0 \ 0 \ 0$

$u_2 = 2 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0$      $u_4 = 0 \ 2 \ 0 \ 0 \ 0 \ 0 \ 0$      $u_6 = 0 \ 0 \ 2 \ 0 \ 0 \ 0 \ 0$

**Dann:**  $\varphi$  erfüllbar gdw. ex. Teilmenge der Vektoren  $\underbrace{v_1, \bar{v}_1, \dots, v_4, \bar{v}_4, u_1, \dots, u_6}_{w_1, \dots, w_{14}}$  mit Summe  $\underbrace{4 \ 4 \ 4 \ 1 \ 1 \ 1 \ 1}_{b}$ .

Also  $\varphi$  erfüllbar  $\Leftrightarrow$  ex.  $I \subseteq \{1, \dots, 14\}$      $\sum_{i \in I} w_i = b$  ( $w_i, b$  Dezimalzahlen).

**1. Approximationsalgorithmus** Zu  $w_1, \dots, w_n, b$  verfare "gierig" (greedy Algorithmus).

**GA:** Ordne die  $w_i$  an in Form  $w_{i_1} \geq w_{i_2} \geq \dots$

Für  $j = 1, \dots, n$  nehme  $w_{i_j}$  hinzu, falls Teilsumme mit  $w_{i_j}$  noch  $\leq b$ .

Formal:  $I := \emptyset$      $c := 0$ . Für  $j = 1, \dots, n$ : falls  $c + w_{i_j} \leq b$  do  $I := I \cup \{i_j\}$   $c := c + w_{i_j}$

**Zeitaufwand:** mit Einheitskostenmaß (in  $n$ , Addition/Vergleich kostet  $O(1)$ ).

1. Sortieren:  $O(n \log n)$  polynomial in  $n$

2. Schleife:  $O(n)$

log. Kostenmaß: Eingabegröße = Länge der Dezimaldarstellungen von  $w_1, \dots, w_n, b$

Polynomiale Laufzeit wegen:  $n \leq l$

1. Schritt  $\rightarrow l^2$  Schritte

EKM    LKM

**GA ist Polynomzeitalgorithmus**

Zeige für optimale Lösung  $I_0 \subseteq \{1, \dots, n\}$  und die durch GA gefundene Lösung  $I$   $\frac{c(I_0)}{c(I)} \leq 2$ . Hierzu genügt

$c(I) \geq \frac{b}{2}$

OBdA  $w_1 \geq w_2 \geq \dots \geq w_n$ . Sei  $j+1$  der erste nicht gewählte Index (sonst alle gewählt,  $c(I_0) = c(I)$ ).

$1, \dots, j$  wurden gewählt.

Fall  $j = 1$   $w_1 + w_2 > b, w_1 \geq w_2$

$2w_1 > b : w_1 > \frac{b}{2}$

Fall  $j \geq 2$  Ergebniswert: bisherige Summe  $+ w_{j+1} > b$

$w_{j+1} \leq w_j = \frac{jw_j}{j} \geq \frac{w_1 + w_2 + \dots + w_j}{j} \leq \frac{b}{j}$  Ergebnis.

$c > b - \frac{b}{j} \geq \frac{b}{2}$

4.2.05

**Approximationsalgorithmus RS:** Geg.  $w_1, \dots, w_n, b \in \mathbb{N}_+$

Gesucht:  $I \subseteq \{1, \dots, n\}$  mit  $\sum_{i \in I} w_i$  max., wobei  $\sum_{i \in I} w_i \leq b$

**Definition 15** Ein Polynomzeit-Approximationsschema (PTAS (polyn. time. approx. scheme.)) liefert für jedes  $\epsilon > 0$  einen Algorithmus der zu Eingabe  $x$  eine Lösung  $M$  liefert mit  $\frac{c(M_0)}{c(M)} \leq 1 + \epsilon$  (mit  $M_0$  optimale Lösung), polynomzeitbeschr. in  $|x|$ .

Erlaubte Laufzeit:  $|x|^{\frac{1}{\epsilon}}$

**Beispiel (Ein PTAS für das Rucksackproblem):**

1. Sortiere  $b \geq w_1 \geq w_2 \geq \dots \geq w_n$
2. Setze  $k := \lceil \frac{1}{\epsilon} \rceil$
3. Für jedes  $I \subseteq \{1, \dots, n\}$  mit  $|I| \leq k$  wende GA initialisiert mit  $I$  an und ermittle so Ergänzung  $I \subseteq I^*$  mit  $\sum_{i \in I} w_i \leq b$

Ausgabe: maximales gefundenes  $I^*$

Polynomzeit für festes  $\epsilon$  (festes  $k$ )

Anzahl der  $I$ :  $\sum_{0 \leq i \leq k} \binom{n}{i} \leq \sum_{0 \leq i \leq k} n^i \in O(n^k)$  Erzeugung der  $I$  etwa in kanonischer Reihenfolge der Wörter über  $\{1, \dots, n\}$

**Behauptung** Bei Ausgabe  $I$  und optimaler Lösung  $I_0 = \{i_1, \dots, i_p\}$  gilt:  $\frac{c(I_0)}{c(I)} \leq 1 + \frac{1}{k} \leq 1 + \epsilon$

Fall 1  $p \leq k$  Dann wird  $I$  in 3, behandelt. Optimum wird erreicht.

Fall 2  $k < p$  Betrachte GA' für Initial.  $I = \{i_1, \dots, i_k\}$  Ausgabe  $I^*$

Wenn  $I^* = I_0$ , dann fertig.

Sonst ex.  $i_q \in I_0$ , aber in GA' übergangen wurde.

$i_q > i_k$   $i_q \in I_0 \setminus I^*$  Bekannt  $c(I^*) + w_{i_q} \geq b \geq c(I_0)$

$$w_{i_q} = \frac{(k+1)w_{i_q}}{k+1} \leq \frac{w_{i_1} + w_{i_2} + \dots + w_{i_k} + w_{i_q}}{k+1} \leq \frac{c(I_0)}{k+1}$$

$$\frac{c(I_0)}{c(I^*)} \leq \frac{c(I_0)}{c(I_0) - w_{i_q}} \leq \frac{c(I_0)}{c(I_0) - \frac{c(I_0)}{k+1}} = \frac{1}{1 - \frac{1}{k+1}} = \frac{k+1}{k} = 1 + \frac{1}{k}$$

**Ende** Kommentare und Korrekturvorschläge sind erwünscht.

Vielen Dank an alle die mich unterstützt haben dieses Skript zu erstellen.

Viel Erfolg bei der Klausur