GTI SS-2002 Mitschrift

nach einer Vorlesung von
Prof. Dr. Joachim Biskup

Letzte Änderung:
17. Mai 2002

Mitwirkende:
M. Meyer, C. Kolek, C. Schumann, A. Swiatek

Diese Mitschrift versucht so weit wie Möglich wiederzugeben, was Prof. Biskup in der Vorlesung schriftlich zur Verfügung stellt. Sie versucht so vollständig wie möglich zu sein, doch kann dies natürlich nicht garantiert werden. Außerdem werden zahlreiche Tippfehler und ähnliches enthalten sein.

Offizielle Website: www.ja-me.de/veritas/

Inhaltsverzeichnis

Tabellenverzeichnis

Abbildungsverzeichnis

15.04.2002

1 Einleitung

1.1 Motivation

- Welche Aufgabenklassen maschinell beschreiben ?
- Welche dann effizient ?
- Welche sogar mit endlichem Speicher ?
- Anordnung / Behandlung von :
  - Festlegung von Programmiersprachen
  - Übersetzung von Programmen

1.2 Was ist "Berechenbar" ?

Zur Erinnerung

- von Neumann Universalrechner
- Programm-gesteuert: kann beliebiges Programm ausführen
- digital: optimiert auf Werten (über {0,1})
- grundlegende Fähigkeiten:
  - Speicherzellen adressierbar
  - Worte (Datum / Befehl) holen (lesen)
  - Worte (Datum / Befehl) wegbringen (schreiben)
  - Steuerung durch Sequenzbildung, beliebige Sprünge
  - Arithmetik

2 Turingmaschine mit Turing-Programmen als Abstraktion

- Programmgesteuert
- operiert auf Worten über endl. Alphabet Γ mit B $\in \Gamma$, z.B. Γ = {0,1,B} — B = Blank
- Arbeitsspeicher (Turing Band) siehe S. 8

  Alle Felder des Turingbandes müssen mit gültigen Buchstaben aus dem Alphabet gefüllt sein. Das Band ist weder links noch rechts beschränkt und ist und ungenutzten Stellen mit B $\in \Gamma$ gefüllt. Der Schreib- / Lesekopf kann per Schritt lokal bewegt werden (R,L,N). — Rechts, Links, Stehenbleiben
- Programme: (zunächst seperat)

  Zustände (Stellen): Q endlich

  Programmzeile: (q $\in$ Q, a $\in \Gamma$) $\rightarrow$ (q' $\in$ Q, a' $\in \Gamma$, d $\in$ {L,R,N}) — a = gelesenes Zeichen
- Semantik: Situation (Konfiguration)

  Programmzustand: q $\in$ Q / Bandinhalt / Kopfposition α q β:

  $\alpha \in \Gamma^*$, q , $\beta \in \Gamma^*$

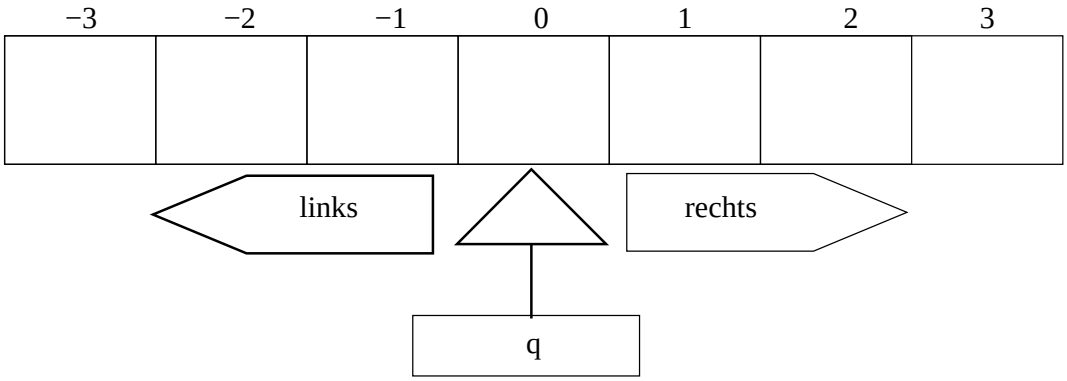

Abbildung 1: Turing Band

- Ausgangssituation: ε q_0 β
  β = Eingabewort mit $\beta \in \Sigma^* \subset \Gamma$, B $\notin \Sigma$

  ε = leeres Wort

  $\Sigma = \{a,b,c\} = \Sigma^a$
  $\{aa,ab,ac,...,ca,cb,cc\} = \Sigma^2$
  $\{x_1, x_2, ..., x_i \in \Sigma\} = \Sigma^i$
  $\{\varepsilon\} = \Sigma^0$

- Folgesituation: gemäß $\delta \curvearrowright$ Zweistellige Relationen auf Situationen $\vdash$, $\dot{\vdash}$: reflexive, transitive Abbildung
- Stopp-Situation: α q β mit β = a β' mit δ(q,a) = q,a,N), a $\in \Gamma$, $\beta' \in \Gamma^*$
- Ausgabewort: Präfix von β bis zum ersten Vorkommen von B. Alternativ F $\subset$ Q als aktueller Zielzustand.
- Turing Programm TM berechnet Funktion δ(TM) :

$$\Sigma^* \overrightarrow{partiell}(\Gamma\{B\})^*$$

  vermöge:

  Eingabewert $\mapsto$ Ausgabesituation $\mapsto \uparrow$ (undefiniert) oder
  Stopp Situation $\mapsto$ Ausgabewert

- Turing Programm semi-enscheidet Sprache

  $\mathcal{L}$(TM) $\subset \Sigma^*$ vermöge w $\in \mathcal{L}$(TM) :gdw Eingabewert w $\mapsto$ Ausgangssituation $\mapsto$ Stopp Situation $\mapsto$ Ausgangswert = 1

- Turing Programm hat Laufzeit(-verhalten)

  Time(TM) : $\Sigma^* \overrightarrow{partiell} \mathcal{N}$, Eingabewert w $\mapsto$ # (Schritte zum Stoppen)

  analog Platzbedarf: #(Zellen, die benötigt werden)

  häufig auch: $\mathcal{N} \overrightarrow{partiell} \mathcal{N}$, Eingabegröße n $\mapsto$ MaxTime(TM)(w) w $\in \Sigma^n$

3 (Eine ansatzweise) formale Definition des Berechenbaren

- f: $\Sigma^* \overrightarrow{p} \Sigma^*$ heißt (partiell) rekursiv / berechenbar :gdw ex. TM mit δ(TM) = f

- f: $\mathcal{N}^* \overrightarrow{p} \mathcal{N}$ heißt (partiell) rekursiv / berechenbar :gdw ex TM mit δ(TM)(bin(i_1) # bin(i_2) # ... # bin(i_k)) = bin(f(i_1 ... i_k)) falls $(i_1 ... i_k) \in \mathcal{N}$
  bin(x) = binärdarstellung von x, # = Trennzeichen
- L $\subset \Sigma^*$ heißt rekursiv / entscheidbar gdw: ex: TM mit
  δ(TM)(x) = $\mathcal{X}_L$(x) := 1 falls x $\in$ L, 0 falls x $\notin$ L
- L $\subset \Sigma^*$ heißt rekursiv / aufzählbar / semi-entscheidbar :gdw ex. TM mit
  $\delta(TM)(x) = 1$ gdw $x \in L$

$\delta(TM)$: totale Fkt.

$\delta(TM)$ möglicherweise partielle Fkt.

4 Programm als Graph

4.1 Zustandsübergang

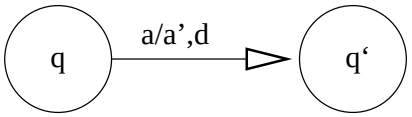

Abbildung 2: Programm als Graph Bsp.

4.2 Idee

1. Eingabe von Typ $0^i 1^j, j \leq 1$
2. Wenn rechts 1 gelöscht dann links 0 löschen
3. akzeptieren: Wort verschwindet

Programm als Graph siehe Seite 10

5 Turing Programmierung und Varianten

- "alle Tricks" der normalen Programmierung
- Makros (als Abkürzung)
- Unterprogramm-Aufruf
- komplexe Datentypen $\curvearrowright$ **Mehrspur-Maschine** für $\Gamma \in \Sigma_1 \times \Sigma_2 \times ... \times \Sigma_{\mathcal{L}}$ mit **beliebigem Alphabet**.
- Kontrollstrukturen, insbesondere Wiederholungen [beliebige Sprünge bereits nach Definition!]
- tabellenartige Speicher [wie CASE Kontrollstruktur]
  $\curvearrowright$ **Mehrband-Maschinen**

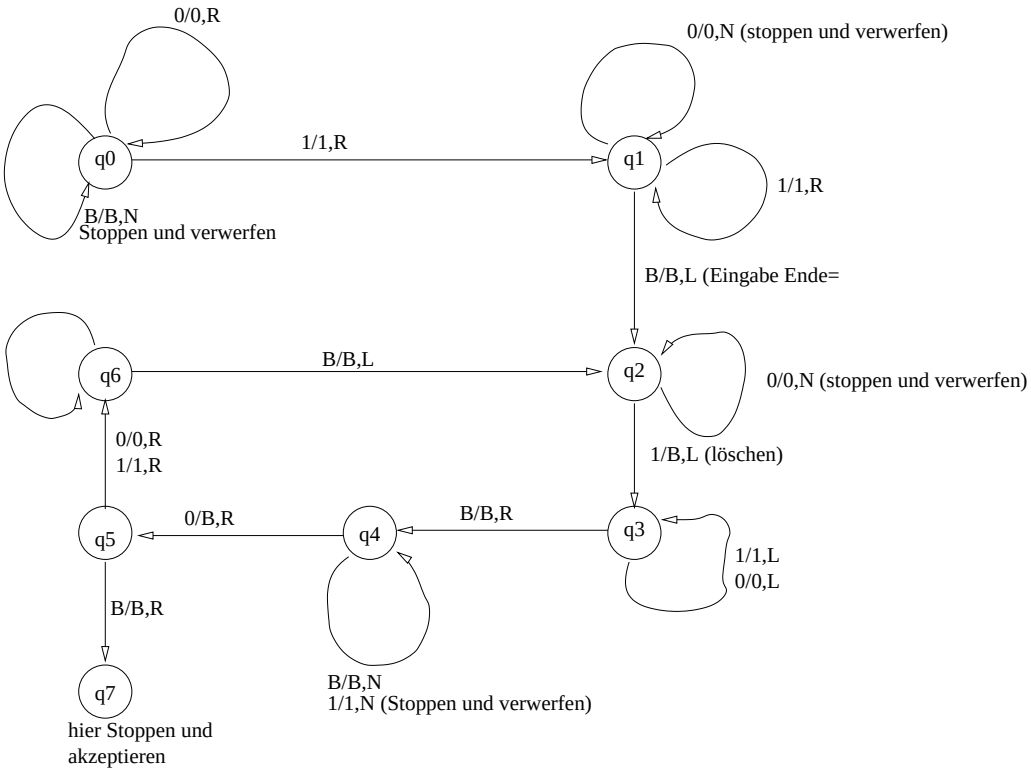

Abbildung 3: Programm zu Idee

18.04.2002

6 Einleitung 18.04.2002

"Vorleser"	"Hörer"
"ließt"	"hören"
(antwortet)	(fragen)
alles Geschriebene lesbar	hören, lesen, verstehen (?)
↷ Folien vorfertigen	Vorlesungsnotizen
↷ am Projektor schreiben	↓
↷ nur vordere Tafeln benutzen	Vorlesungsmitschrift
	↑
	Bücher, Übungen, Nachdenken
Arbeitszeit ca. 40 Sdt.	
ca. 8 Std. "lehre liefern"	ca. 20 Std. "lehre abholen"
+ ...	+ ...
Verteilt von Montag 08:00 Uhr bis Freitag	18:00, ggf. Sa. nach vorgabe Stundenplan

7 Die Turing Maschine

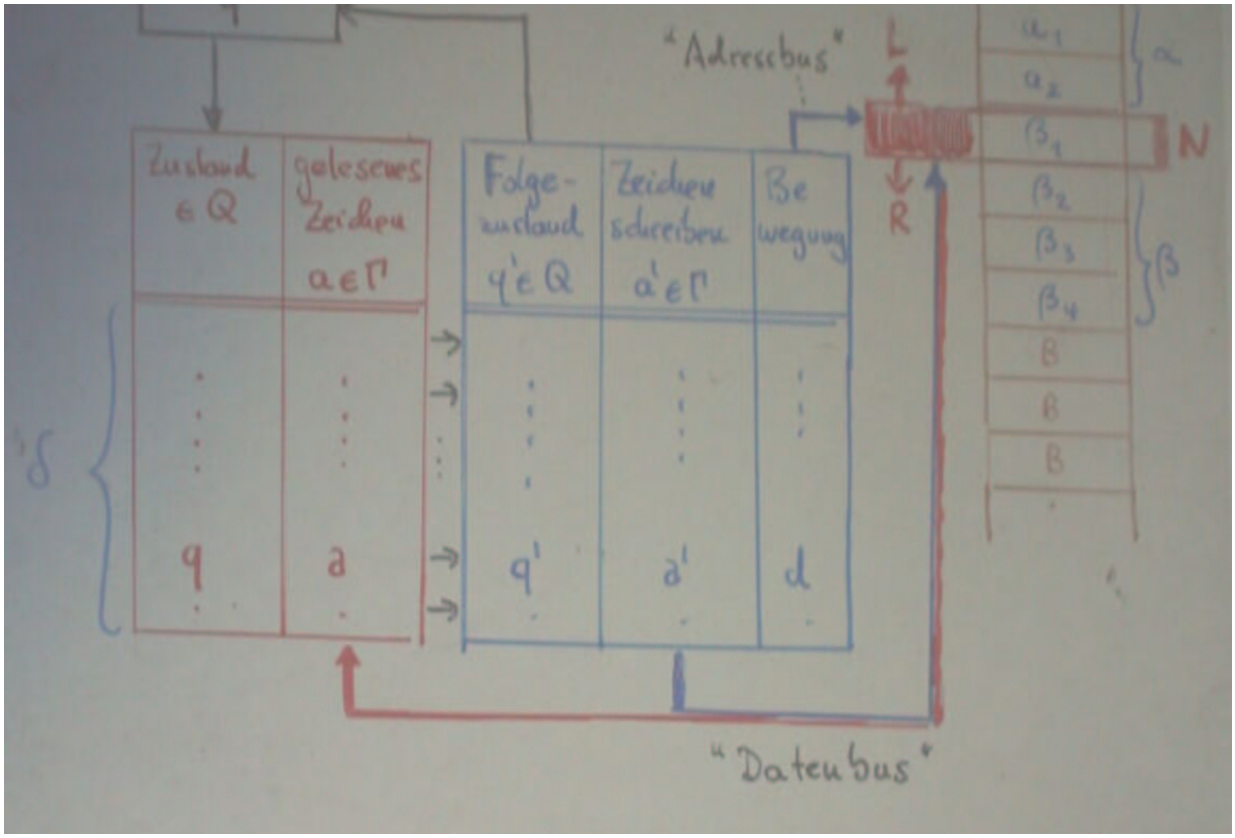

Abbildung 4: Eine Turing Maschine

7.1 Syntax von Turing Programm TM

$\delta : Q \times \Gamma \to Q \times \Gamma \times \{\text{R,L,N}\}$ Zustandsüberführung. $q_0 \in Q$ Anfangszustand

7.2 Semantik von TM

$\delta(TM) : \Sigma^* \overrightarrow{p} (\Gamma \setminus \{B\})^*$

Eingabewort $\mapsto$ Ausgangssituation $\mapsto$ $\begin{cases} \uparrow \\ \text{Stop-Situation} \mapsto \text{Ausgabewort} \end{cases}$ mit Eingabewort w $\in \Sigma^*$mit$\Sigma \subset \Gamma \setminus \{B\}$

8 Church' sche These

"berechenbar schlechthin" = durch ein Turing-Programm bearbeitbar

8.1 Rechtfertigung(en)

1. alle anderen Formalisierungen wurden **als äquivalent bewiesen** durch konstruktive Simulation (ebenfalls "berechenbar")
2. - minimalistischer Ansatz $\curvearrowright$ tatsächlich berechenbar
   - beliebig anreicherbar[1] $\curvearrowright$ alles erfassbar

8.2 gesucht: Turing Programm UTM mit

$$\delta(UTM) : \underbrace{\underbrace{\text{Turing Programm}}_{\hat{=}\Gamma_1^*} \times \underbrace{\text{Daten}}_{\hat{=}\Gamma_2^*} \rightarrow \underbrace{\text{Daten}}_{\hat{=}\Gamma_3^*}}_{\delta(UTM):\Gamma^* \rightarrow \Gamma^*}$$

geeignet kodieren

$$\delta(UTM)(\overbrace{\underbrace{\text{code(TM)} \in \Gamma^* \quad}_{\in\Gamma^*}, \in \Gamma^* \quad x \in \Gamma^*}^{\text{Eingabe}}) = \overbrace{\delta(TM)(x)}^{\text{Ausgabe}}, \forall TM, \forall x$$

$\delta(UTM)(w, x) := \text{error} \in \Gamma \quad \forall w \in \Gamma^*, \text{die kein korektes Turing Programm sind}$

8.3 gefunden:

- CPU bewirkt "Berechenbares" $\curvearrowright^2 \exists$ UTM mit UTM $\hat{=}$ CPU
- Programmiere CPU als Turing Programm, nutze dabei alle "Tricks"

8.4 Gibt es nicht berechenbares ?

Ja!

weniger Programme: $\| \Gamma^* \|$
als Funktionen: $\| \Sigma^{*\Sigma^*} \|$

Für geeignete Codierung

8.5 genauere Überlegung

- wähle Alphabet geeignet für Codierung : Γ
- w $\in \Gamma^*$ deutbar als

$$\begin{cases} \text{Turing-Programm (ggf. nicht "korrekt")} \\ \text{Datum} \end{cases}$$

- **Selbstanwenung**: $\overbrace{\delta(\underbrace{w}_{\text{Als Programm}})}^{\text{Semantik}}(\underbrace{w}_{\text{Als Datum}}) = \delta(UTM)(\underbrace{w}_{\text{Als Programm}})(\underbrace{w}_{\text{Als Datum}})$
- "Als Akzeptor" : $\delta(w)(w) = 1$ gdw "w akzeptiert w"

Satz 1 ***Diagonalisierung:*** $D=\{w \mid \underbrace{w \textit{ akzeptiert } w \textit{ nicht}}_{\Sigma(w)(w)\neq 1}\}$ *D ist nicht rekursiv*

[1] gilt insbesondere auch für Programmgesteuert
[2] nach Church

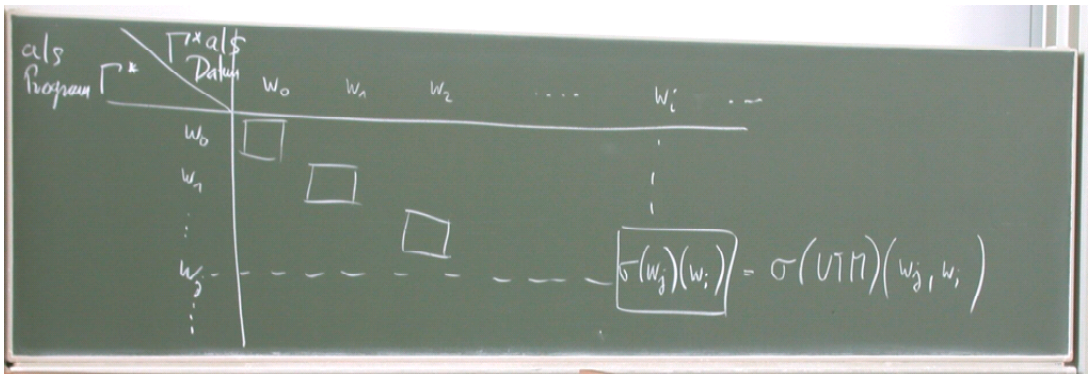

Abbildung 5: UTM als Tabelle

8.6 Diagonalisierung

vgl. Abb. S.14

Definition 1 *$D = \{w \mid w$ akzeptiert w nicht$\}$ (1)*

Satz 2 *D ist nicht rekursiv*

Korollar 1 *$\bar{D}$ ist nicht rekursiv*

8.7 Beweis

(indirekt) andernfalls sei w_j akzept. Turing Programm

$$\delta(w_j)(x) = \mathcal{X}_D(x) = \begin{cases} 1 & \text{falls x} \in D \quad (2) \\ 0 & \text{falls x} \notin D \quad (3) \end{cases}$$

8.8 Fallunterscheidung

8.8.1 1. Fall

$w_j \in D \overset{(2)}{\Rightarrow} w_j$ akzeptiert w_j
$w_j \in D \overset{(1)}{\Rightarrow} w_j$ akzeptiert w_j nicht

8.8.2 2. Fall

$w_j \notin D \overset{(3)}{\Rightarrow} w_j$ verwirft w_j
$w_j \notin D \overset{(1)}{\Rightarrow} w_j$ akzeptiert w_j

8.9 Halteproblem

$$H := \{ \quad \underbrace{\underbrace{TM, w}_{\in \Gamma^*}}_{\text{o.B.d.A. geeignet gewählt}} \quad \mid \quad \underbrace{\delta(TM)(x) \text{ definiert}}_{\text{TM hält bei Eingaben von x}} \quad \}$$

Satz 3 *H ist* nicht *rekursiv*

8.10 Beweis

(durch Reduktion)
angenommen: H rekursiv vermöge TM_h
dann auch: $\bar{D}$ rekursiv mit hilfe von TM_h
also: Widerspruch

Satz 4 *H ist rekursiv aufzählbar*

Korollar 2 $\{ L \mid L \text{ rekursiv}\} \overset{\subset}{\neq} \{L|L \text{ rek. aufzählbar}\}$

TM:
Eingabe: $w \in \Gamma^*$
zu bestimmen: w $\in \bar{D}$?
Verfahren:

w akzeptiert w?

1. bestimme mit TM_H, ob $\delta(w)(w)$ definiert [3]
2. falls nein: "$w \notin \bar{D}$" Ausgabe
   falls ja: berechne (mit UTM) $\delta(w)(w)$
   falls Ergebnis "akzeptiert": Ausgabe: "$w \in \bar{D}$"
   falls Ergebnis "verwirft": Ausgabe: "$w \notin \bar{D}$"

8.11 Eine (berühmte, wichtige Anwendung)

Logik: Sprachen für

Formeln:	$F \subset \Sigma^*$	
Aussagen:	$A \subset F$	
Theoreme mit Beweis:	$TB \subset A \times \Sigma^*$	
Theoreme:	$T \subset F$	
F: rekursiv:		"Compiler" bauen
A: rekursiv:		"leiche Syntaktische Überprüfung" (keine "freien Vorkommen" von Variablen
TB rekursiv:		"korrekte Einzelbeweisschritte" leicht erkennbar; es gibt "nur wenige Grundtypen" davon !
T rek. aufzählbar, nicht rekursiv:		"Beweis finden zu schwer":

Σ^* geeignet gewählt

- simuliere Turing Programm durch Aussagen
- reduziere Halteproblem auf Beweisfinden

[3] hält

22.04.2002

9 Pariellität und Nicht-Rekursivität

$f : \Sigma^* \xrightarrow{p} \Sigma^*$
patriell rekursiv
:gdw ex. TM mit
$\delta(TM) = f$

Haltemenge :=
$\{TM, x | \delta(TM)(x)$definiert
ist nicht rekursiv

f. alle $x : \delta(TM)(x) = f(x)$
f. alle $x : \delta(TM)(x)$definiert
gdw f(x) definiert

(und falls definiert, dann gleich)

f. alle $TM : \delta(TM) \neq \mathcal{X}_H$
f. alle TM ex. x :
$\underbrace{\delta(TM)(x)}_{\text{möglicherweise undefiniert}} \neq \underbrace{\mathcal{X}_H(x)}_{\text{immer definiert}}$

Fall 1: $\delta(TM)(x)$ undef.
Fall 2: $\delta(TM)(x)$ definiert,
aber verschieden von $\mathcal{X}_H(x)$

Fazit: Partiellität lässt sich i. allg. nicht ("berechenbar") beheben !

10 Partiellität und Universalität

es gibt universelle UTM:
$\underbrace{\delta(UTM)(TM, x)}_{\text{möglicherweise undefiniert}} = \underbrace{\delta(TM)(x)}_{\text{(immer beide Seiten zusammen)}}$

(fiktive) Beschränkung
auf alle totalen Funktionen:

$\curvearrowright$ erlaubt kein
universelles Programm in der gleichen Sprache !

Programmiersprace für
aussschließlich totale Fkt:
$(\underbrace{\tau(w)}_{\text{berechnete totale Funktion}})_{w \in \Sigma^*}$
mit τ : Semantik, w : "Programm"

10.1 Beweis

Diagonalisierung

indirekte Annahme: $\overset{\text{"Programm"}}{TU} \in \Sigma^*$ universell für Sprache ex. $\overset{\text{"Programm"}}{d \in \Sigma^*}$: $\tau(d)(x) = \underbrace{\tau(TU)(x,x)}_{=\tau(x)(x)}$"+1"$(\neq \tau(x)(x))$ $\overset{\text{für } x:=d}{\curvearrowright} \tau(d)(d) \neq \tau(d)(d)$, also Widerspruch!

11 rekursiv aufzählbar und rekursiv

$L \subset \Sigma^*$

rekursiv aufzählbar	**rekursiv**
ex. TM mit	ex. TM mit
$\underbrace{\delta(TM)}_{\text{möglicherweise partiell}}(x) = 1$ gdw $x \in L$	$\underbrace{\delta(TM)}_{\text{total}}(x) = \begin{cases} 1 & x \in L \\ 0 & x \notin L \end{cases}$

- Partiellitt i. allg. nicht überwindbar:
  Haltemenge := $\{TM, x | \delta(TM)(x) \text{definiert}\}$
  rek. aufz., aber nicht rekursiv
- "komplementäre Partiellität" übereindbar:

Satz 5 *L rekursiv gdw L und* $\bar{L} := \Sigma^* \setminus L$ *rek. aufz.*

11.1 Beweis

"$\Rightarrow$": gemäß Definition

"$\Leftarrow$": parallel laufen lassen bis Ausgabe "1" erscheint $\begin{cases} \text{sei } TM_L \text{Programm für L} \\ TM_{\bar{L}} \text{ Programm für } \bar{L} \end{cases}$

$\curvearrowright$ total, "entscheidet" L (und $\bar{L}$)

12 L rekursiv aufzählbar

- (1) semi-entscheidbar: ex. TM mit $\delta(TM)(x) = 1$ gdw $x \in L$
- (2) $\mathcal{X}_L$ partiell rekursiv: ex. TM mit $\delta(TM)(x) = \mathcal{X}_L^*(x) := \begin{cases} 1 & \text{falls } x \text{ in } L \\ \uparrow & \text{sonst} \end{cases}$
- (3) Definitionsbereich einer partiell rekursiven Fkt:
  ex. TM mit domain$(\delta(TM)) = L$
- (4) Wertebereich einer partiell rekursiven Fkt:
  ex. TM mit range $(\delta(TM)) = L$
- (5) Wertebereich einer total rekursiven Fkt: ex. TM mit $\delta(TM)$ total und
- (6) [ex. berechenbares Aufzählungsverfahren: $\overbrace{\text{spuckt nacheinander}}^{\text{potentiell unendlicher Vorgang}}$ genau die Elemente von L aus]

1 $\Leftarrow$ 2: nach Def.
1 $\Rightarrow$ 2: Ausgabe $\neq 1$ $\curvearrowright$ ersetzen durch "Endlos-Schleife"
2 $\Rightarrow$ 3: nach Def.
2 $\Leftarrow$ 3: beim "Halten": noch Ausgabe von "1"
3 $\Rightarrow$ 4:

1. rette Eingabe x
2. ersetze beim Halten das eigendliche Ergebnis durch gerettete Eingabe

$3 \Leftarrow 4$: $y \in \text{range}\ (\delta(TM)) \Leftrightarrow \underbrace{\text{ex. x}}_{\text{suche nach solchen x}} : \delta(TM)(x) = y$

Argumente w_i / Schrittzahl j	w_0	w_1	w_2	...
0	•	↙	↙	↙
1	↙	↙	↙	↙
2	↙	↙	↙	↙
3	↙	↙	↙	↙
4	↙	↙	↙	↙
...	↙	↙	↙	↙

↓ Tiefensuche, → Breitensuche
jede Kombination w_i,j kommt vor !
Eingabe: x
Voraussetzung: $x \in L \Leftrightarrow x \in \text{range}(\delta(TM))$
Methode: falls x als Ausgabe erscheint: dann stoppen (und Ausgabe 1) sonst: Endlos-Schleife

13 Berechenbarkeit für Funktionen und Mengen (Sprachen)

Funktionen	**Mengen (Sprachen)**
$f : \Sigma^* \underset{\text{p}}{\rightarrow} \Sigma^*$	$L \subset \Sigma^*$
partiell rekursiv :gdw	rek. aufz.:
ex. TM mit $\delta(TM) = f$	ex. TM mit $\delta(TM) = \mathcal{X}_L^*$
	Menge L $\mapsto$ Funktion $\mathcal{X}_L^*$

Funtkion f $\mapsto$ Funktionsgraph
graph(f) := $\{ \underbrace{(x,y)}_{\approx \tilde{\Sigma^*} \text{ mit geeigneter Codierung}} \mid y = f(x)\} \subset \Sigma^* \times \Sigma^*$

Satz 6 *f partiell rekursiv gdw. graph(f) rek. aufz.*

Satz 7 *Wenn f total rekursiv dann graph(f) rekursiv*

13.1 Beweis

"⇒" benutze "Diagonalsuche" !
"⇐" Eingabe: x
Verfahren: zähle graph(f) auf wenn (x,y) erscheint dann Ausgabe: y

13.2 Entscheidungsverfahren fÜr graph(f)

Eingabe: (x,y) Verfahren: berechne $\underbrace{f(x)}_{\text{prüfe, ob y = f(x)}}$

14 Abschlusseigenschaften

Satz 8 *Die Menge der* rekursiven *Sprachen ist abgeschlossen unter:*

- *Komplement*
- *Vereinigung*
- *Durchschnitt*

Satz 9 *Die Menge der* rek. aufz. *Sprachen ist abgeschlossen unter*

- *Vereinigung*
- *Durchschnitt*

[nicht unter komplement: H ist Gegenbeispiel !]

25.04.2002

15 Formalismus für das "Berechenbare"

(universelle Programmiersprache)

- abstrahiert "reale Rechner / Programmiersprachen";
- als *Turing-Programme* mathematisierbar;
- erlaubt *universelles Programm* und damit
- *Selbstanwenung* und *Diagonalisierung*;
- beinhaltet notwendigerweise *Partiellität* und
- dementsprechend *Nicht-Rekursivität* der Definiertheit;
- erlaubt zu unterscheiden zwischen rekursiv - rekursiv aufzählbar - nicht rekursiv aufzählbar z.B.
  $\{0^n1^n|x \leq 1\}$ - Haltemenge - $\overline{\text{Haltemenge}}$

16 Kann man (Turing-) Programme implizit definieren ? - als Lösung welcher Gleichungen ?

ja, vermöge berechenbarer **"Programmtransformationen"**:

17 Rekursionssatz

Satz 10 *Für alle total rekursiven Funktionen* $g : \Gamma^* \to \Gamma^*$ *gibt es* $w \in \Gamma^*$ *mit*

$$\delta(\underbrace{w}_{Programm}) = \delta(\underbrace{g(w)}_{transformiertes\ Programm})$$

Programm *und* transformiertes Programm *haben gleiche Semantik !*

- w ist die Lösung der Gleichung $\delta(X) = \delta(g(X))$ mit X "Unbekannte" für Programme.
- w ist *"Semantik-Fixpunkt"* der *syntaktischen Programmtransformation*

17.1 Beweis

Beweis: betrachte Funktion τ_a mit Parameter a

$$\tau_a(x) := \begin{cases} \delta(\delta(a)(a))(x) & \text{falls} \delta(a)(a) \downarrow \\ \uparrow & \text{sonst} \end{cases} \tag{0}$$

$$\text{jedes} \tau_a \text{ist patiell rekursiv} \tag{1}$$

$$\text{sei h:} a \mapsto TM_a mit \tau_a = \delta(\underbrace{TM_a}_{h(a)})\text{h ist total rekursiv} \tag{2}$$

$$\delta(h(a))(x) = \begin{cases} \delta(\delta(a)(a))(x) & \text{falls } \delta(a)(a) \downarrow \\ \uparrow & \text{sonst} \end{cases} \tag{3}$$

(3)=(0*)

$$g \circ h \qquad \underbrace{\text{total}}_{\delta(TM)(TM)\downarrow} \qquad \underbrace{\text{rekursiv}}_{(4)} \Leftarrow \begin{cases} \text{g total rek.} \\ \text{h total rek.} \end{cases}$$
$$(5)$$

$$\exists TM \text{ auf } g \circ k = \delta(TM) \tag{4}$$

definiere: w:= h(TM)
z.Z. $\delta(w) = \delta(g(w))$
z.Z. f. alle x: $\delta(w)(x) = \delta(g(w))(x)$

17.1.1 Beweis

$$(\delta(\underbrace{h(TM)}_{w})(x) = \delta(\underbrace{\delta(TM)}(TM))(x) \overset{(4)}{=} \delta(g \circ h(TM))(x) = \delta(g(\underbrace{h(TM)}_{w}))(x) \qquad (5)(3)$$

18 Spielanwendung mit ernsthaltem Hintergrund:

gibt es TM mit $\delta(TM)(x)$ hält gdw x=TM ?
[TM "zählt" genau sich selbst (eigene Beschreibung) auf]
Hintergrund: "Selbstreproduktion möglich ?"
solches TM mit Rekursionssatz bestimmbar:

- betrachte "Programmtransformation" g mit

$$\delta(g(a))(x) := \begin{cases} 1 & \text{falls x=a} \\ \uparrow & \text{sonst} \end{cases}$$

  bei Eingabe von a liefert g(a) ein Programm, das genau für dessen Eingabe x=a die Ausgabe 1 liefert und sonst immer undefiniert bleibt.

- g ist total rekursiv
  $\curvearrowright$ (Rekursionssatz) ex. w mit $\delta(g(w)) = \delta(w)$
  $\curvearrowright \delta(w)(x) = \delta(g(w))(x) = \begin{cases} 1 & \text{falls } x = w \\ \uparrow \end{cases}$
  $\curvearrowright \delta(w)(x)$ hält gdw x=w

19 "Superanwendung" für Kernproblem der Informatik: welche Eigenschaften von Programmen sind entscheidbar?

- ex. x mit $\delta(TM)(x)$ definiert? *TM nicht trivial ?*
- f. alle x: $\delta(TM)(x)$ definiert? *TM total ?*
- ex. unendlich viele x mit $\delta(TM)(x)$ definiert? *TM unendlich ?*
- $\delta(TM_1) = \delta(TM_2)$? f. alle x: $\delta(TM_1)(x) = \delta(TM_2)(x)$? *TM_1 und TM_2 äquivalent ?*
- f. alle x: wenn $\delta(TM_1)(x)$ delta, dann auch $\delta(TM_2)(x)$ definiert ? *TM_1 in TM_2 enthalten?*

Alle genannten Eigenschaften sind *nicht* entscheidbar !
genauer:
Γ^* : alle Programme (o.B.d.A.: jedes Wort über Γ ist "Programm")
$\mathcal{PR} := \{\delta(w) \mid w \in \Gamma^*\}$
alle partiell rekursiven Fkt. (jeweils einstellig gedeutet)

20 Satz von Rice

Sei $\emptyset \neq \gamma \overset{\subset}{\neq} \mathcal{PR}$ nichttriviale Menge partiell rekursiver Fkt.

Dann ist die zugehörige Proramm-Menge

$$L(\gamma) := \{w \mid \underbrace{\delta(w) \in \gamma}_{\text{w ist ein Programm für eineFunktion aus } \gamma}\}$$

nicht rekursiv.

20.1 Beweis

(indirekt): Angenommen $L(\gamma)$ wäre rekursiv.

$\overset{\text{Def.}}{\curvearrowright}$ mit $\delta(TM)(w) = \mathcal{X}_{L(\gamma)}(w) = \begin{cases} 1 & w \in L(\gamma) \\ 0 & w \notin L(\gamma) \end{cases}$

γ nicht trivial /nach Voraus.): ex. w_1, w_2 mit

$$w_1 \in L(\gamma) \quad (\Leftrightarrow \delta(w_1) \in \gamma)$$

$$w_2 \notin L(\gamma) \quad (\Leftrightarrow \delta(w_2) \notin \gamma)$$

definiere (Diagonalisierung)

$$g(x) := \begin{cases} w_2 & \text{falls } x \in L(\gamma) \\ w_1 & \text{falls } x \notin L(\gamma) \end{cases}$$

g total rekursiv

$\overset{\text{Rekursionssatz}}{\rightarrow}$ g hat "Fixpunkt": ex. w mit $\delta(w) = \delta(g(w))$

Fälle für w:

Fall 1:

$$\underbrace{w \in L(\gamma)}_{\delta(w)\in\gamma} \curvearrowright \underbrace{g(w) = w_2 \notin L(\gamma)}_{\delta(g(w))\notin\gamma}$$

Fall 2:

$$\underbrace{w \notin L(\gamma)}_{\delta(w)\notin\gamma} \curvearrowright \underbrace{g(w) = w_1 \in L(\gamma)}_{\delta(g(w))\in\gamma}$$

20.2 Anmerkunken zum Satz von Rice

Satz von Rice liefert negative Antworten für Entscheidbarkeit von (vielen) Programm-Eigenschaften:

1. "nicht trivial" ? "total" ? $\curvearrowright$ direkt "unendlich" ?
2. "äquivalent" ? "enthalten" ? $\curvearrowright$ mit Reduktion z.B.:

angenommen: " $\overbrace{\delta(TM_1) = \delta(TM_2)}^{\text{als Eigenschaft von } TM_1 \text{ und } TM_2}$ wäre entscheidbar
wähle: $\overline{TM_2}$ fest derart, dass $\delta(\overline{TM_2})(x) =\uparrow$ f. alle x
dann: " $\underbrace{\delta(TM) = \delta(\overline{TM_2})}_{\text{als Eigenschaft von TM}}$ " ebenfalls entscheidbar
dann: "f. alle x: $\delta(TM)(x)$ undefiniert" entscheidbar *curvearrowright* Widerspruch zum Satz von Rice

29.04.2002

21 Was ist berechenbar ?

eine "maschinennahe" Abstraktion:
Turing-Maschine mit Turing-Programmen

$$\mathcal{PR} := \{f : \Sigma^* \underset{p}{\to} \Sigma^* \mid ex.TMmit\delta(TM) = f\}$$

TM im Wesentlichen: $\delta : Q \times \Gamma \times \{R, L, N\}$
stelle := q_0;
repeat
inhalt := read()
case
...
if stelle = q and inhalt = a then stelle := q' ; write(a'); address(d')
...
esac
until Stopp-Situation
Welche Funktionen sind es denn nun ?
Was ist berechenbar ?

- $\begin{cases}\text{durch "Programm beschreibbar"}\\ \text{durch "Maschine beschreibbar"}\end{cases}$ $\curvearrowright$ Turing-Programme
  ... (viele andere, stets äquivalente Abstraktionen)
- durch "berechenbare Zusammensetzung" $\curvearrowright$ *Funktionale*
  von "offensichtlich Berechenbarem" $\curvearrowright$ *Grundfunktionen*
  ausgehend von Grundfunktionen den Abschluss unter Funktionalen bilden !

21.1 Exkurs: Funktionsterme versus Funktionen

z.B.:	+(x,y)	soll bedeuten	Additionsfunktion
z.B.:	•(x,+(y,1))	soll bedeuten	Komposition von Nachfolgerfunktion bzg. y mit Multiplikationfunktion

Unterscheidung häufig nur implizit: Syntax versus Semantik

22 Grundfunktionen (über Alphabet Σ)

- Nullfunktion(erase)

$$E : \Sigma^* \to \Sigma^*$$
$$E(x) := \lambda \text{ f. alle } x \in \Sigma^*$$

λ: "leeres Wort", neutrales Element bezüglich Konkatenation

- Nachfolgefunktionen für $a \in \Sigma$:

$$S_a : \Sigma^* \to \Sigma^*$$
$$S_a(x) = \underbrace{ax}_{\text{Konkatenation}} \quad \forall x \in \Sigma^*$$

- Projektionsfunktionen für $1 \leq j \leq n$:

$$P_j^n : \Sigma^* \times ... \times \Sigma^* \times ... \times \Sigma^* \to \Sigma^*$$
$$P_j^n(x_1, ..., x_j, ..., x_n) := x_j \quad \forall x_1, ..., x_j, ..., x_n \in \Sigma^*$$

23 Funktionale (für Funktionen über Σ)

- Substitition (Komposition): seien

$$g : {\Sigma^*}^m \to \Sigma^*$$
$$h_i : {\Sigma^*}^n \to \Sigma^* \qquad \text{für } i = 1, ..., m :$$

f entsteht durch *Substitution* aus $g, h_1, ..., h_m$:

$$f : {\Sigma^*}^n \to \Sigma^*$$
$$f(x_1, ..., x_n) := g(h_1(x_1, ..., x_n), ..., h_m(x_1, ..., x_n)) \quad \forall x_1, ..., x_n \in \Sigma^*$$

kurz: $f := \mathcal{L}(g, h_1, ..., h_m)$

- Primitive Rekursion: seien

$$g : {\Sigma^*}^{n-1} \to \Sigma^*$$
$$h_a : {\Sigma^*}^{n+1} \to \Sigma^* \quad \text{für } a \in \Sigma;$$

f entsteht durch Primitive Rekursion aus g, $(h_a)_{a\in\Sigma}$:

$$f : {\Sigma^*}^n \to \Sigma^*$$
$$f(\underbrace{\lambda}_{\text{"Rekursionsanfang"}}, x_2, ..., x_n) := g(\underbrace{x_2, ..., x_n}_{\text{"Parameter"}})$$

Rekursionsschritt

$$f(y\mathbf{a}, x_2, ..., x_n) := h_a(\underbrace{y}_{\text{"Restargument"}}, \overbrace{f(y, fx_2, ..., x_n)}^{\text{Ergebnis für "Restargument"}}, \underbrace{x_2, ..., x_n}_{\text{"Parameter"}})$$

kurz: $f := \mathcal{PR}(g, (h_a)_{a\in\Sigma})$

speziell n=1:

$$f(\lambda) := g \qquad \text{mit } g \in \Sigma^*$$
$$g(ya) := h_a(y, f(y))$$

Definition 2 $f : {\Sigma^*}^n \to \Sigma^*$ *heißt* primitiv-rekursiv *:gdw f kann aus den Grundfunktionen* E, S_a, P_j^n *durch (endliche) Anwendung der Funktionale L, PR gewonnen werden.*

Satz 11 *Jede primitiv-rekursive Funktion ist total (Turing-) berechenbar (unter geeigneten Verabredungen).*

Beweis-Ansätze:

1. Church'sche These anwenden: "alles programmierbar".
2. (a) Turing-Programme für Grundfunktionen angeben
   (b) Turing-Programme für Funktionale angeben, als "(Turing-) Programmtransformationen"
3. Vergleich mit "üblichen Programmiersprachen":
   $\mathcal{L}$: Assignment, Statement Sequence / Expression
   $\mathcal{PR}$: Bounded Loop (For Statement)

23.1 Beispiele

$$\Sigma := \{1\} \curvearrowright \Sigma^* \underbrace{\approx}_{\text{per Strichcodierung ("Bierdeckel-Codierung")}} \mathcal{N}$$

1. arithmetische Nachfolger-Funktion:

   $$Succ(x) := S_1(x)$$

   S_1 : "Wort-Nachfolgefunktion" für $1 \in \Sigma$

2. arithmetische Addition:
   Vorüberlegung:

   $$(\underbrace{y}_{\text{"Restargument"}} + 1) + \underbrace{w}_{\text{als "Parameter" benutzen}} = \overbrace{(y+w)}^{\text{Ergebnis für "Restargument"}} + 1$$

   formal:

   durch "Primitive Rekursion" $\underbrace{\begin{cases} Add(\lambda, w) = w = P_1^1(w) \\ Add(y1, w) = S_1(Add(y, w)) \\ = S_1(\underbrace{P_2^3(y, Add(y, w), w)}_{Add(y,w)}) \end{cases}}_{\text{durch "Komposition"}}$

3. arithmetische Multiplikation:
   Vorüberlegung:

   $$(\underbrace{y}_{\text{"Restargument"}} + 1) * w = \overbrace{y * w}^{\text{Ergebnis für Restargument}+w}$$

   Wobei + schon als primitive-rekursion nachgewiesen wurde.
   formal:

   $$Mult(\lambda, w) = \overbrace{\lambda}^{\approx 0} = E(w)$$
   $$Mult(y1, w) = Add(Mult(y, w), w)$$
   $$= Add(P_2^3(y, Mult(y, w), w), P_3^3(y, Mult(y, w), w))$$

23.2 Beispiele, Fortsetzung

nunmehr $\Sigma := \{0,1\} \curvearrowright \Sigma^* \underbrace{\approx}_{\text{per Binärkodierung}} \mathcal{N}$

1. arithmentische Nachfolger-Funktion:

$$Succ(\lambda) = 1 \in \Sigma \subset \Sigma^* = \underbrace{S_1(E(\lambda))}_{\text{falls "erwünscht": durch Grundfunktion ausdrückbar}}$$

$$\underbrace{Subb(w0)}_{w0+1=w1} = w1 = S_1(w) = \underbrace{S_1(\underbrace{P_1^2(w, Succ(w))}_{w})}_{\text{"von Form } h_1(w,Succ(w))\text{"}}$$

$$\underbrace{Succ(w1)}_{w_n...w_1 1+1=Succ(w)0} = Succ(w)0 = S_0(Succ(w)) = S_0(P_2^2(w, Succ(w)))$$

mit $w_n...w_1 = w$

24 berechenbar = primitiv rekursiv ?

"berechenbar" $\overset{?}{=}$ **primitiv rekursiv**

natürlich nicht !

$\overbrace{\quad}$ umfasst "Partiellität" — $\overbrace{\quad}$ stets total

was fehlt ?

- "unbeschränkte Suche"
- allgemeine Wiederholung / Rekursion (Repeat Statement, While Statement)

$\curvearrowright$ Abstraktion als weiteres Funktional

25 Minimierung

Minimierung über $\underbrace{\{a\}^*}_{\approx\mathcal{N}}$ **mit a** $\in \Sigma$:

sei $\psi : \Sigma^{*^{n+1}} \underset{p}{\to} \Sigma^*$; (möglicherweise) partiell !

ϕ entsteht durch Minimierung aus ψ: (Anwendung des μ-Operators)

$$\phi : \Sigma^{*^n} \underset{p}{\to} \Sigma^*$$

$$\phi(x_1,...,x_n) := \begin{cases} a^m & \text{mit } m \in \mathcal{N} \text{ minimal bzg:} \\ & -\forall p \le m : \psi(a^p, x_1, .., x_n) \downarrow \\ & -\psi(a^m, x_1, ..., x_n) = \lambda \\ & \text{falls so ein m existuert} \\ \uparrow & \text{sonst} \end{cases}$$

kurz: $\phi := Min_a\psi$

Definition 3 $\phi : \Sigma^{*^n} \underset{p}{\to} \Sigma^*$ *heißt μ-rekursiv :gdw*

ϕ kann aus den Grundfunktionen E, S_a, P_j^n durch endliche Anwendung der Funktionale $\mathcal{L}, \mathcal{PR}$, Min_a gewonnen werden.

Satz 12 *Jede μ-rekursive Funktion ist (Turing-) partiell rekursiv.*

Beweis-Ansatz:
Church'scje These mit $Min_a\psi \hat{=}$ "unbeschränkte Wiederholung":

$$\text{kann "schleifen"} \begin{cases} \textbf{while} & \overbrace{\psi(out, x_1, ..., x_n)}^{\text{kann "versanden"}} \neq \lambda \textbf{ do} \\ & out := S_a(out) \\ \textbf{od} & \end{cases}$$

Satz 13 *Jede (Turing-) partiell rekursive Funktion ist μ-rekursiv.*

Beweis-Ansatz:
betrachte universelles Turing-Programm UTM:

$$\delta(UTM)(TM, W) = \delta(TM)(w)$$

1. UTM stellt Anfangskonfiguration für TM, w her: $\epsilon q_0 w$
2. UTM interpretiert TM schrittweise
3. UTM stoppt, wenn erstmals Stopp-Sitiation auftritt
4. UTM erzeugt Ausgabe

dann nachprüfbar (siehe auch "maschinelle Abstraktion");
(a),(b) für einen Schrit, (d): primitiv-rekursiv
(c): Minimierung
$\curvearrowright$ μ-rekursiv

26 Kleene'scher Normalform-Satz

Satz 14 *Jede* partiell rekursive Funktion *hat eine Darstellung (als Funktionsterm) "mit primitiv-rekursiven Funktionen und **einer** Minimierung".*

02.05.2002

27 (dynamische) Komplexitätsmaße

Klassifikation von $\underbrace{\text{"Problemen"}}_{\text{Funktionen, Sprachen (Wortmengen)}}$:

- ...
- nicht rekursiv aufzählbar
- rekursiv aufzählbar
- rekursiv:
  - ...
  - "eher nicht effizient"
  - ...
  - "eher effizient"
  - ...

27.1 Abstraktion für "Effizenz" ?

z.B. für *Turing-Programme*:

$$Time(TM) : \Sigma^* \underset{p}{\rightarrow}, Time(TM)(w) := \#(\text{ Schritte bis zum Stoppen })$$

$$Space(TM) : \Sigma^* \underset{p}{\rightarrow} \mathcal{N}, Space(TM)(w) := \#(\text{ benutzte Zellen bis zum Stoppen })$$

für Turing-Programme gilt:

- für TM, die "volle Eingabe lesen":

$$Space(TM)(w) \leq Time(TM)(w) \quad \forall w \in \Sigma^*$$

- falls Time(TM)(w) $\downarrow$, dann

$$Time(TM)(w) \leq O(2^{Space(TM)(w)})$$

k := # (benutze Zellen)
l := $\|\Gamma\|$ Alphabetgröße
n := $\|Q\|$

mit k und l $\curvearrowright l^k$ verschiedene Bandinhalte
mit k, l und n $\curvearrowright \leq n * k * l^k$ verschiedene Situationen

$\curvearrowright$ entweder TM stoppt nach höchstens so vielen Schritten, oder TM läuft in "Endlos-Schleife"

27.2 weitergehende Abstraktion

$$\underbrace{w \in \Gamma^*}_{\text{Programme (Syntax)}} \begin{cases} \delta(w) & \text{berechnete Funktion (Semantik)} \\ \gamma(w) & \text{Aufwandsfunktion (dyn. Komplexität)} \end{cases}$$

$\gamma(w)(x)$:= "Berechnungsaufwand" von Programm w bei Eingabe x

27.3 Formalisierung

Blum'sches Komplexitätsmaß für Gödelnummerierung

- $F = (\delta(w))_{w\in\Gamma^*}$ sei Aufzählung aller (einstelligen) partiell rekursiven Funktionen (o.b.d.A. vom Typ $\delta(w) := \Gamma^* \underset{p}{\to})$ mit:
  1. $U(w,x) := \delta(w)(x)$ ist partiell rekursiv
  2. $\exists$ total rekursive Fkt. c mit: $\delta(c(u,v)) = \delta(w) * \delta(v)$
- $\mathcal{CM} = (F, \Phi)$ heißt Komplexitätsmaß :gdw
  - F ist Gödelnummerierung (wie oben beschrieben)
  - $\Phi = (\gamma(w))_{w\in\Gamma^*}$ ist Aufzählung von partiell rekursiven Aufwandsfunktionen vom Typ $\gamma(w) : \Gamma^* \underset{p}{\to} \mathcal{N}$ mit

    (B1) $\delta(w)(c)\downarrow$ gdw $\gamma(w)(x)\downarrow$

    (B2) $g(w,x,b) := \begin{cases} 1 & \text{falls } \gamma(w)(x) = n \\ 0 & \text{sonst} \end{cases}$ ist total rekursiv

    (n mit geeigneter Codierung)

    d.h. $\{(w,x,b) \mid \gamma(w)(x) = n\}$ ist entscheidbar!

Beispiele:

- Time
- Space

Gegenbeispiele

- $bin(\gamma(w)(x)) := \delta(w)(x)$ Widerspruch zu B2, H unendscheidbar.
- $\gamma(w)(x) := 1$ Widerspruch zu B1

überaus mächtige Abstraktion, z.B. "jedes Problem kann beliebig schlecht programmiert werden!"

Satz 15 *Sei f eine total rekursive Funktion ["zu programmieren"], g eine total rekursive (Schranken-* $g \hat{=}$ h !!! *) Funktion ["zu unterbieten"]. Dann $\exists$ [Programm] $w = \Gamma^*$ mit*

1. $f = \delta(w)$
2. $\gamma(w)(x) > \underbrace{h(x)}_{\text{als nat. Zahl gedeutet}} \quad \forall x \in \Gamma^*$

Beweis: definiere h $\hat{=}$g!!!

$$\psi(x,w) := \left\{ \begin{array}{ll} f(x) & \text{falls } \underline{\text{nicht}}\gamma(w)(x) \le h(x) \\ \delta(w)(x) \quad \underbrace{\text{"+1"}}_{\text{Diagonalisierung}} & \text{sonst} \end{array} \right\} \text{entscheidbar mit B2}$$

mit x : "eigendliche Argument", w: Parameter
Church'sche These: ex. "Programmtransformation" als rekursive Fkt. s mit

$$\psi(x,w) = \delta(s(w))(x)$$

s total !

Rekursionssatz:

(3) $\exists w_0$(Fixpunkt) mit $\delta(w_0) = \delta(s(w_0))$ } tut es !

$\curvearrowright$ für w_0 trurr Sonderfall nicht ein denn sonst:

$$\delta(w_0)(x)\text{"+1"} \overset{(1b)}{=} \psi(x, w_0) \overset{(2)}{=} \delta(s(w_0))(x) \overset{(3)}{=} \delta(w_0)(x)$$

also Widerspruch.
"es gibt beliebig aufwändige berechenbare Probleme!"

Satz 16 *Sei h eine total rekursive (Schranken-) Funktion. Dann gibt es eine total rekursive Funktion f mit:* $\forall$*[Programme]*$w \in \Gamma^*$:
wenn $\delta(w) = f$,
dann $\gamma(w)(x) > h(x)$*für fast allex* $\in \Gamma^*$
(bis auf endlich viele Ausnahmen)

Beweis:

$$f(x) := Max\{\delta(w)(x)\text{+"1"} \mid w \leq x \text{ und } \gamma(w)(x) \leq h(x)\}$$

f total rekursiv wegen B1, B2 sei w ?derart?, dass $\delta(w) = f$
$\curvearrowright \gamma(w)(x) > h(x)$ $\forall x \geq w$ (gemäß Def. mit Diagonalisierung)

"Funktions-Ergebnisse lassen sich durch Aufwand abschätzen (aber nicht umgekehrt, i. Allg.)!"

Satz 17 *Es gibt eine total rekursive Funktion* α*, so dass für alle[Programme]* $w \in \Gamma^*$ *gilt:*

$$\delta(w)(x) \leq \alpha(x, \gamma(w)(x)) \qquad \textit{für fast alle } x \in \Gamma^*$$

Beweis:

$$\alpha(x, y) := Max\{\delta(w)(x) \mid w \leq x, \gamma(w)(x) = y\}$$

total rekursiv. α tut es !

28 Komplexitätsklassen

Definition 4 *Sei* α *eine total rekursive Funktion.*
$\mathcal{C}_\alpha := \{f \mid f$ *total rekursiv, ex. [Programm]* $w \in \Gamma^*$ *mit* $\delta(w) = f$ *und* $\gamma(w)(x) \leq \alpha(x)$ *f. fast alle* $x\}$

klar:

1. $\alpha(x) \leq \beta(x)$ $\curvearrowright \mathcal{C}_\alpha \subset \mathcal{C}_\beta$
2. $\forall\alpha\exists\beta$ mit $\mathcal{C}_\alpha \subsetneq \mathcal{C}_\beta$

aber viele "merkwürdige Phänomene", z.B.

28.1 Lückensatz

Sei h total rekursiv mit $h(x) \geq x$. Dann gibt es eine ("beliebig große") total rekursive Fkt α mit $\mathcal{C}_\alpha = \mathcal{C}_{h \circ \alpha}$

28.2 Beschleunigung-Satz

Zu jeder $\underbrace{\text{total rekursiven Fkt. g}}_{\text{"inverse Beschleunigung"}}$ gibt es eine total rekursive Funktion f mit:

$\forall w$ mit $\delta(w) = f \exists u$ mit $\delta(u) = f$ und $g(x, \gamma(u)(x)) < \gamma(w)(x)$ f. fast alle x

06.05.2002

29 Was ist "tatsächlich" / "effizient" berechenbar (entscheidbar) ?

$$\text{TM Turing Programm} \begin{cases} \delta(TM) & \text{semantik} \\ Time(TM) & \text{dynamische Zeit-Aufwandsfunktion} \end{cases}$$

Funktion $S : Q \times \Gamma \to Q \times \Gamma \times \{R, L, N\}$, d.h. deterministisch.
als "worst-case" - Verhalten, überladen bezeichnet:

$$Time(TM)(n) := Max\{Time(TM)(w) \mid w \in \Sigma^n\}$$

"Längste Rechenzeit bei Eingabe mit Gräße n", wobei stehts $\|\Sigma\| \leq 2$ vorausgesetzt, d.h. für Zahlen: "Länge der Zahldarstellung logarithmisch in Zahlwert

2+1	100	hundert
3+1	1000	tausend
4+1	10000	zehntausend

30 Ansatz für "tatsächlich" / "errifzient" berechenbar / entscheidbar

polynomiell (Zeit-) beschränkt, d.h. ex. Polynom $P : \mathcal{N} \to \mathcal{N}$ mit

$$Time(TM)(n) \leq P(n) \qquad \underbrace{\text{für fast alle n} \in \mathcal{N}}_{\text{für "hinreichend große" n}}$$

zugehörige "Problemklasse" (für Sprachen über Σ)

TM entscheidet L

$$\mathcal{P} := \{L \mid \exists TM \text{ mit}$$

$$(1) \qquad \mathcal{X}_L = \delta(TM)$$

$$(2) \qquad Time(TM)\text{polynomiell beschränkt}\}$$

sind *"praktische Probleme"*
tatsächlich / effizient $\curvearrowright$ polynomiell Zeit-beschränkt
berechenbar / entscheidbar ? welche ?

- Addition
- Multiplikation
- ...
- Klammerstrukturen
- ...

} Ja!

- Graph-Suchprobleme
- ...
- Optimierungsprobleme
- ...

} ?

$\curvearrowright$ weit geteilte Vermutung: nein !

30.1 Beispiele

30.1.1 Beispiel 1: CLIQUE

ungerichteter Graph: $G = (V, E)$ mit $E? \subset ?\{\{v_1, v_2\} \mid v_1 \in V, v_2 \in V\}$

$CLIQUE = \{(V, E), k, V' \mid$ "V' ist Clique der Größe k in Graph (V,E)": $V' \subset V$ und $\forall v_1, v_2 \in V' : \{v_1, v_2\} \in E$ und $\|V'\| = k\}$

polynomiell (Zeit-) beschränkt entscheidbar !

drei abgeleitete "Probleme":

$CLIQUE_1 := \{(V, E), k \mid \exists V' : [(V, E), k, V'] \in CLIQUE\}$
"Clique-Größe k in (V,E) erreichbar ?"

$CLIQUE_2 : (V, E) \mapsto Max\{k \mid \exists V' : [(V, E), l, V'] \in CLIQUE\}$
"berechne optimale Clique-Größe in (V,E)!"

$CLIQUE_3 : (V, E) \mapsto V'$ mit $''$

1. V' Clique in (V,E)
2. mit optimaler Größe"

- folgende *Programmtransformationen* sind offensichtlich:
  Programm für $CLIQUE_3$ (optimale Clique!) $\underset{\text{abzählen!}}{\curvearrowright}$ Programm für $CLIQUE_2$ (optimale Clique-Größe!) $\underset{k \leq \text{ optimale Clique-Größe?}}{\curvearrowright}$ Programm für $CLIQUE_1$ (Clique-Größe k erreichbar?)
- Umkehrungen gelten auch!
- alle "Probleme" erscheinen "aufwändig":
  unmittelbared Überprüfen der Definition erfordert *Suche* in einem "exponentiell großen Suchraum"
  $[V' \in \mathcal{P}V$ mit $\|\mathcal{P}V\| = 2^{\|V\|}]$

zu "Umkehrung gelten auch":

Programm für $CLIQUE_1$ (Clique-Größe l erreichbar ?)
$\curvearrowright$ Programm für $CLIQUE_2$ (optimale Clique-Größe!)
Eingabe: (V,E)
Methode:

1. bestimme $n := \|V\|$ $[CLIQUE_2(V, E) \leq n]$
2. für k := 1,...,n:
   - wende Programm für $CLIQUE_1$ auf Eingabe (V,E),k an
   - bestimme das Maximum der "positiven Antworten"

Ausgabe: bestimmtes Maximum
Beobachtung:
wenn Programm für $CLIQUE_1$ polynomiell (Zeit-) beschränkt
dann auch Programm für $CLIQUE_2$ polynomiell (Zeit-) beschränkt

Programm für $CLIQUE_2$ (optimale Clique-Größe l!)
$\curvearrowright$ Programm für $CLIQUE_3$ (optimale Clique)
Eingabe: (V,E)
Methode:

1. wende Programm für $CLIQUE_2$ auf Eingabe (V,E) an: liefert k_{opt} [Optimale Clique-Größe]
2. betrachte nacheinander Kanten e aus E:
   - Kante e "probeweise entfernen'
   - falls "optimale Clique-Größe im Restgraph" $= k_{opt}$
     dann e endgültig entfernen
     sonst e behalten

Ausgabe: "Restgraph" [bildet optimale Clique]
Beobachtung: entsprechend !

Satz 18 *Wenn $CLIQUE_i$ polynomiell (Zeit-) beschränkt berechenbar / entscheidbar ist, so auch $CLIQUE_j$ für $j \neq i$.*

Beweis: Obige Programmtransformationen mit "Beobachtungen"

"Lehre": um die "Aufwändigkeit" der Suchprobleme zu bestimmen [tatsächlich exponentiell?] reicht es in diesem Beispiel (und vielen anderen), das *Entscheidungsproblem* zu untersuchen:

$$CLIQUE_1 := \{(V,E), k \mid \exists V' : [(V,E), k, V'] \in CLIQUE\} = proj_{(v,E),K}(CLIQUE)$$

30.1.2 Beispiel 2: Rucksack packen

n Objekte 1,...,n
mit Gewichten $g_1, .., g_n \in \mathcal{N}$
und Nutzen $a_1, ..., a_n \in \mathcal{N}$

1 Rucksack mit
Gewichtsschranke $G \in \mathcal{N}$
gefordertem Nutzen $A \in \mathcal{N}$

$$KNAPSACK := \{g_1, ..., g_n, a_1, ..., a_n, G, A, I \mid I \subset \{1, ..., n\} \text{ "Packung" mit } \underbrace{\sum_{i \in I} g_i \leq G}_{\text{"packbar"}} \text{ und } \underbrace{\sum_{i \in I} a_i \geq A}_{\text{"nützlich"}}\}$$

$$KNAPSACK_1 := \{g_1, ..., g_n, a_1, ..., a_n, G, A \mid \exists I \text{ mit}$$
$$g_1, ..., G, A, I \in KNAPSACK\} = proj_{g_1,...,g_n,a_1,...,a_k,G,A}(KNAPSACK)$$
"Nützlichkeit A erreichbar ?"

$$KNAPSACK_2 : g_1, ..., g_n, a_1, ..., a_n, G \mapsto Max\{A \mid ...\}$$
"maximale Nützlichkeit!"

$KNAPSACK_3 : g_..., k_n, a_1, ..., a_n, G \mapsto I$ mit "I packbar und maximal nützlich"
"maximale nützliche Packung!"

30.1.3 Beispiel 3: Handelsreisender (Traveling Salesman)

n Orte 1,...,n
$\underbrace{\text{mit "Reisekosten" c(i,j)} \in \mathcal{N}}$
Graph mit Knoten {1,...,n}
gerichteten Kanten (i,j),
jeweils mit c(i,j) bewertet

Rundreise
$\underbrace{\text{mit Kostenbeschränkung } B \in \mathcal{N}}$
Hamiltonkreis, d.h. Permutation

$$\pi = \begin{pmatrix} 1 & & n \\ & ... & \\ \pi(1) & & \pi(n) \end{pmatrix} \text{ mit}$$

$$\sum_{j=1,...,n} c(\pi(j), \underbrace{\pi(j+1)}_{\text{"mod n" gerechnet}}) \leq B$$

$$TRAVSALE := \{(c(i,j))_{\substack{1 \leq i \leq n \\ 1 \leq j \leq n}}, B, \pi \mid ...\}$$

$$TRAVSALE_1 := proj_{(c(i,j))_{\substack{1 \leq i \leq n \\ 1 \leq j \leq n}}, B}(TRAVSALE)$$

"Kostenbeschränkung B erreichbar"

$$TRAVSALE_2 : (c(i,j))_{\substack{1 \leq i \leq n \\ 1 \leq j \leq n}} \mapsto Min\{B \mid ...\}$$

"minimale kosten!"

$$TRAVSALE_3 : (c(i,j))_{\substack{1 \leq i \leq n \\ 1 \leq j \leq n}} \mapsto \pi \text{ mit "}\pi \text{ kostenminimale Rundreise"}$$

kostenminimale Rundreise !"

30.2 Beobachtungen

- Alle "Grund-Probleme" $PROBLEM$ sind polynomiell (Zeit-) beschränkt entscheidbar
- Alle "Erreichbarkeits-Probleme" $PROBLEM_1 := proj_{...}(PROBLEM)$ sind *Suchprobleme*:
  - ex. Clique V' ...?
  - ex. Packung I ...?
  - ex. Permutation π ... ?

  mit einem gemäß Definition "exponentiell großen Suchraum ":
  - $v' \in \mathcal{P}V$ mit $\|\mathcal{P}V\| = 2^{\|V\|}$
  - $I \in \mathcal{P}\{1,...,n\}$ mit $\|\mathcal{P}\{1,...,n\}\| = 2^n$
  - $\pi \in Perm(1,...,n)$ mit $\|Perm(1,...,n)\| = n!$

31 generische Struktur von (deterministischen) Suchprogrammen

- erzeuge Suchraum: alle Kandidaten, die gemäß Definition überprüft werden müssen [in Beispielen jeweils "exponentiell"]
- entscheide "Grund-Problem" für je einen Kandidaten
  [in Beispielen jeweils polynomiell (Zeit-) beschränkt]
  [in Beispielen insgesamt: exponentieller Zeitaufwand !]

31.1 Abstraktion für "Suchraum durchlaufen"

- einen Kandidaten nichtdeterministisch wählen ("raten")
- falls überhaupt möglich: Auswahl stets "richig" d.h. ein "positiver" Kandidat wird gewählt

32 generische Struktur von nichtdeterministischen Suchprogrammen (als Abstraktion!)

- wähle nichtdeterministisch einen Kandidaten derart, dass -falls überhaupt möglich- dieser "positiv" ist [Wahl hinschreiben: polynomiell (Zeit-)beschränkt]
- überprüfe für gewählten Kandidaten, ob er "tatsächlich" positiv ist

  [in Beispielen: polynomiell (Zeit-) beschränkt]

  [in Beispielen insgesamt: polynomieller Zeitaufwand !]

33 Abstraktion im Kontext von Turing-Programmen

bislang: deterministisch $\curvearrowright \delta$ Funktion

nunmehr: nichtdeterministisch $\curvearrowright \delta$ Relation, d.h.

$$\delta \subset (Q \times \Sigma) \times (Q \times \Sigma \times \{L, R, N\});$$

dann $\vdash$ entsprechend abändern:

Folgesituation von $\alpha q \overbrace{a\beta}^{\beta}$ gemäß irgend einem q',a',d mit $(q, a, q', a', d) \in \delta$

[o.B.d.A. nur Akzeptoren von Sprachen betrachten]

L(NTM) := $\{w \mid$ Eingabewort w $\mapsto$ Ausgangssituation $\vdash$ (für mindestens eine Rechnung) Stopp-Situation $\mapsto$ Akzeptieren $\}$

$$NTIME(NTM)(w) := \begin{cases} \#(\text{Schritte in kürzester akzeptierender Rechnung} & \text{falls } e \in L(NTM) \\ 0 & \text{sonst} \end{cases}$$

13.05.2002

34 NP

34.1 Definition

Definition 5 **NP**$= \{L \mid \exists NTM\, mit$
(1) $L = L(NTM)$
(2) $NTIME(NTM)$ *polynomiell beschränkt*$\}$

34.2 Satz

Satz 19 $CLIQUE_1, KNAPSACK_1$ *und* $TRAVSALE_1$ *sind Elemente von* **NP**

Beweis:
siehe obige Beobachtung

34.3 Satz

Satz 20 **P** $\subset$ **NP**

Beweis:
Gemäß Definition (jedes deterministische Programm auch als formal "nichtdeterministisch" deutbar)

$$\mathbf{P} := \{L \mid \exists TM \text{ mit } \mathcal{X}_L = \delta(TM), Time(TM) \text{ polynomiell beschränkt } \}$$
$$\mathbf{NP} := \{L \mid \exists NTM \text{ mit } L = L(NTM), NTIME(NTM) \text{ polynomiell beschränkt } \}$$

34.4 Satz

Satz 21 *Für alle* $L \in$ **NP**$\exists$ *Polynom p,* $\exists$ *(deterministische) TM mit*
(1) $\mathcal{X}_L = \delta(TM)$
(2) $TIME(TM)(n) \leq 2^{p(n)}$

Beweis:
sei $L \in$ **NP**
$\curvearrowright \exists NTM, \exists$ Polynom q mit $L = L(NTM), NTIME(NTM)(n) \leq q(n)$
$\curvearrowright \forall w \in L, \exists$ akzeptierende nichtdeterministische Rechnung mit
#(Schritte für w) $\leq q($ $\underbrace{\mid w \mid}_{\text{Länge von w}}$ $) =: n$
für jeden Schritt gilt: nichtdeterministische Auswahl erfolgt aus höchstens

$$\|Q\| * \|\Gamma\| * \|\{R, L, N\}\| =: k$$

vielen Möglichkeiten (k Eigenschaft von NTM, unabhängig von w)
$\curvearrowright$ ("zulässige") Rechnung von NTM der Länge m

$$\hat{=}(i_1, ..., i_m) \in (Q \times \Gamma \times \{R, L, N\})^m$$
$$\text{mit } : i_t \in Q \times \Gamma \times \{R, L, N\}$$
$$\text{ist Tripel } (q' \in Q, a' \in \Gamma, d \in \{R, L, N\}),$$

das in Schritt t, t=1,...,m, bei dann gegebenem Zustand q und gelesenem Zeichen a geählt werden kann, d.h.

$$(q, a, \underbrace{q', a', d}_{=i_t}) \in \delta$$

es gibt $\leq k^m$ solcher "zulässigen" Rechnungen !

34.5 Konstruktion der Behaupteten TM

diese simuliert alle möglichen "zulässigen" Rechnungen von NTM.

Eingabe: w

Methode:

1. berechne $n :=\mid w \mid$
2. berechne $m := q(n)$
3. simuliere nacheinander jede "zulässige" Rechnung von NTM
4. falls eine (von NTM) akzeptierende Rechnung gefunden
   dann akzeptiere
   sonst verwerfe

$\curvearrowright$ insgesamt: " poly + $k^{\overbrace{m}^{q(n) \text{ mit q poly}}} * poly + poly$"$\leq 2^{p(n)}$ für geeignetes Polynom P

35 Reduktion von Problemen

transformiere je eine Aufgabe w_1 bezüglich $Problem_1$ (Sprache L_1) in eine Aufgabe w_2 bezüglich $Problem_2$ (Sprache L_2) vermöge "Reduktionsfunktion" f:

$$w \in L_1 gdw f(w) \in L_2$$

$$\mathcal{X}_{L_1} = \mathcal{X}_{L_2} \circ f$$

für Entscheidungserfahren:
wenn f total rekursiv und L_2 rekursiv,
dann L_1 rekursiv
[denn: Menge der total rekursiven Fkt. abgeschlossen unter $\circ$]
für "effiziente" Entscheidungsverfahren:
wenn f polynomiell (Zeit-) beschränkt und $L_2 \in \mathbf{P}$
dann $L_1 \in \mathbf{P}$
[denn: Menge der polynomiell (Zeit-) beschränkten Fkt. abgeschlossen unter $\circ$: $Time(c(TM_2, TM_1))(w) = Time(TM_1)(w) + Time(TM_2)(\delta(TM_1)(w))$]

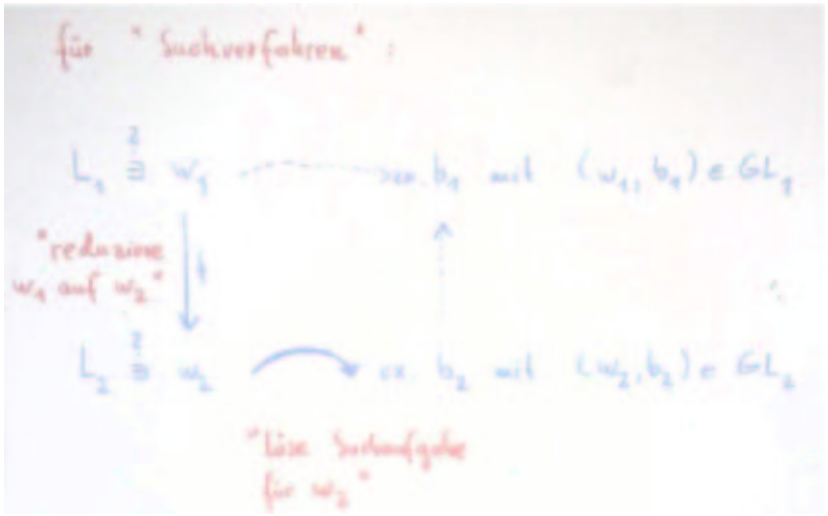

Abbildung 6: Suchverfahren

Definition 6 *$\leq_p$ definiere eine zweistellige Relation auf Sprachen wie folgt:*
$L_1 \leq_p L_2 := gdw \exists$ *polynomiell (Zeit-) beschränkte Funktion f mit* $w \in L_1 gdw f(w) \in L_2$
"L_1 im Wesentlichen nicht aufwändiger als L_2"

Satz 22 *$\leq_p$ ist transitiv (und reflexiv)*

Satz 23 **P** *ist bezüglich $\leq_p$ "nach unten abgeschlossen"*

Satz 24 **NP***ist bezüglich $\leq_p$ "nach unten abgeschlossen"*

$L \in \mathbf{P} \curvearrowright L \leq_p L_1$ f. fast alle L_1
P "unterste" Problemklasse bez. $\leq_p$
(vgl. Abb. 7 Seite 39)

Abbildung 7: NP oder P

36 NP-vollständig

Definition 7 *L heißt NP_vollständig :gdw*

- $L \in$**NP**
- $\forall L_1$ *in***NP** *gilt:* $L_1 \leq_p L$

Definition 8 *L heißt* NP-hart *:gdw*

- $\forall L_1 \in$ **NP** *gilt:* $L_1 \leq_p L$

Satz 25 *Sei L NP-vollständig. Dann gilt:*

1. *Falls* $L \in \mathbf{P}$*, dann* $\mathbf{P} =$**NP**
2. *Falls* $L \notin \mathbf{P}$*, dann:* $\forall$ *NP-vollständigen* $L_1 : L_1 \notin \mathbf{P}$

Beweis:
(zu 1.): "$\subseteq$" $\mathbf{P} \subset$**NP** nach Definition
"$\subseteq$": $L_1 \in$**NP** $\underset{\text{L NP-vollständig}}{\curvearrowright}$ $L_1 \leq_p L$ $\underset{\text{L} \in \mathbf{P}}{\curvearrowright}$ $L_1 \in \mathbf{P}$
gibt es eine NP-vollständige Sprache ?

- währe "schwierigstes Suchproblem"
- reichte aus, um "$\mathbf{P} \stackrel{=}{?} \mathbf{NP}$" zu untersuchen

...
ja!

- sogar "jede Menge"
- mit "Prototyp" SAT_1

37 Aussagenlogik (Schaltfunktionen)

Aussagenvariablen: $x_1, ..., x_n, ...$

Literale: $..., x_k, \bar{x_k}, ...$ (Negation)

Klausel: Disjunktion von Literalen (von n Variablen)

Formel: Konjunktion von Klauseln (m viele)

$$SAT = \{\Phi, \mathcal{L} \mid \Phi = \bigwedge_j \bigvee_i {}^{(-)}x_{i,j} \text{ und } \mathcal{L} : \{x_1, ..., x_n\} \to \{0,1\} \text{ mit } \mathcal{L}(\Phi) = 1\}$$

$$SAT_1 = \{\Phi \mid \exists \mathcal{L} : [\Phi, \mathcal{L}] \in SAT\}$$

Menge der erfüllbaren aussagenlogischen Formeln

"Schaltfunktionen, die für geeignete Eingabe den Wert 1 ausgeben können"

Suchproblem: .. ex. L ?

exponentiell großer Suchraum: $\{0,1\}^{\{x_1,...,x_n\}}$

Satz 26 $SAT_1 \in$ **NP**

Beweis:

(1) rate nichtdeterministisch eine erfüllende Belegung L

(2) werte Formel Φ unter L aus

(3) falls Auswertung liefert 1

dann akzeptiere

sonst verwerfe

alle Schritte sind polynomiell (Zeit-) beschränkt

Satz 27 *SAT_1 ist NP_vollständig, d.h.*

- $SAT_1 \in$**NP**
- $\forall L_1 \in$**NP***: $L_1 \leq SAT_1$:*
  - *simuliere polynomiell beschränkte Rechnungen durch Formeln*
  - *reduziere "akzeptieren" auf "erfüllende Belegung finden"*

16.05.2002

Beh:

$\underbrace{\forall L \in \mathbf{NP}}$: $\underbrace{\mathcal{L} \leq_p SAT_1}$

ex. NTM, ex. Polynom p: ex. polynomiell (Zeit-) beschränkte Funktion

N ∈ L gdw: ex. nichtdet. akzeptierende Rechnung von NTM der Länge $\leq p(\mid w \mid)$ f mit $w \in L gdw f(w) \in SAT_1$

$W \mapsto$ akzeptierende Rechnung $\mapsto$ simulierende Formel Φ_w

Idee:

- beschreibe Arbeitsweise von NTM unter Eingabe von w durh Formel Φ_w der Aussagenlogik, so dass $w \in L \quad gdw \quad \underbrace{\Phi_w \in SAT_1}_{\Phi_w \text{ erfüllt}}$
- Kontruktion $w \overset{f}{\mapsto} \Phi_w$ ist polynomiell (Zeit-) beschränkt

sei $w \in \Sigma^*$ mit $\mid w \mid = n$ gegeben $m := p(n)$
eine akzeptierende Rechnung muss berücksichtigen:
Zeitpunkt: t=1,...,m
Zellen: $z = -m, -(m-1), ..., -1, 0, 1, ..., m$
Zeichen: $a \in \Gamma$
Zustände: $q \in Q \cup \{x\}$
elementare Aussagen: [z,t,a,q]
mit Bedeutung: auf Zelle z befindet sich zur Zeit t das Zeichen a, und der Kopf schaut im Zustand q auf diese Zelle / im Quasizustand X nicht auf diese Zelle
für jede elementare Aussage: eine Aussagenvariable.
insgesamt: $m * \underbrace{(2m+1) * \|\Gamma\| * (\|Q\|+1)}_{\leq q(n) \text{ für geegnetes Polynom}}$ Aussagenvariablen
mit Hilfe der Aussagenvariablen, [z,t,a,q] durch geeignete Formeln beschreiben:

- Anfangssituation
- akzeptierende (Stopp-) Situation
- Situations-übergänge
- Eindeutigkeit der Situation

37.1 Beschreibung der Anfangssituation

für t=1 steht auf Zellen $z = 0, ..., n-1$ das Wort $w = a_1 ... a_n$; auf alen übrigen Zellen B; Kopf schaut im Anfangszustand q_0 auf Zelle 0:

$\alpha_{A_w} \equiv [-m, 1, B, X] \wedge ... \wedge [-1, 1, B, X] \wedge [0, 1, a_1, q_0] \wedge [1, 1, a_2, X] \wedge ... \wedge [n-1, 1, a_n, X] \wedge [\underbrace{n}_{z}, \underbrace{1}_{t}, \underbrace{B}_{a}, \underbrace{X}_{q}] \wedge ... \wedge [m, 1, B, X]$

mit z: Zelle, t: Zeit, a: Zeichen, q: Zustand (Quasizustand)

37.2 (o.B.d.A. vereinfachte) Beschreibung einer akzeptierenden Stopp-Situation

für t=n schaut der Kopf im akzeotierenden Zustand q_F auf eine der Zellen:

$$\alpha_f \equiv \bigvee_{-m \leq z \leq m, a \in \Gamma} [z, m, a, q_F]$$

(Disjunktion ürber alle möglichen Werte von z und a (polynomiell viele in n)

37.3 Beschreibung der Situations-Übergänge

37.3.1 Fall 1

für
$-m \leq z \leq m$, $1 \leq t \leq m-1$, $x \in \Gamma$, $[z,t,x,X]$ $\underbrace{X}_{\text{Kopf-schaut nicht auf Zelle z}}$:

gilt zum Zeitpunkt t +1

37.3.2 Fall 1.1

Kopf schaut nicht auf Zelle z [z,t+1,x,X]

37.3.3 Fall 1.2

Kopf schaut auf Zelle z, mit Folgezustand q' [z,t+1,x,q']

37.3.4 Fall 1.2.1

vermöge Linkschritt von Zelle z+1, mit q,a gemäß δ [z+1,t,a,q] $\vee$...

37.3.5 Fall 1.2.2

vermöge Rechtsschritt von Zelle z-1 mit q,a gemäß δ [z-1,t,a,q] $\vee$

also:

$\underbrace{\overline{[z,t,x,X]}}_{\text{wenn Fall 1}} \underbrace{\vee}_{\text{dann}} (\underbrace{[z,t+1,x,X]}_{\text{Fall 1.1}} \vee ([z,t+1,x,q'] \wedge [z,1,t,a,q] \vee ...) \vee ([z,t+1,x,q'1] \vee [z-1,t,a,q] \vee$
...))

37.3.6 Fall 2

für
$-m \leq t \leq m$,
$1 \leq t \leq m-1$
$a \in \Gamma$
$\underbrace{q \in Q}_{\text{Kopf schaut auf Zelle z}}$ [z,t,a,q]

gilt zum Zeitpunkt t+1

37.3.7 Fall 2.1

Kopf schaut auf Zelle , mit q', a' gemmaß δ [z,t+1,a',q'] $\vee$...

37.3.8 also

$\overline{[z,t,a,q]} \vee ([z,t+1,a',q'] \vee ... \vee [z,t+1,a',X] \vee)$

$\alpha_{\ddot{u}} \equiv \bigwedge$ "alle solchen Formeln"

37.4 Beschreibung der Eindeutigkeit der Situation

- für $-m \leq z \leq m, 1 \leq t \leq m$ sind $x \in \Gamma, q \in Q \cup \{x\}$ eindeutig bestimmt:

$$\bigvee_{x \in \Gamma, q \in Q \cup \{x\}} [z,t,x,q] \wedge \bigwedge_{(x_1,q_1) \in \Gamma \times Q \cup \{X\} \neq (x_2,q_2) \in \Gamma \times Q \cup \{X\}}$$

- für $1 = t \leq m$ schaut der Kopf auf mindestens / höchstens eine Zelle:

$$\bigvee_{-m \leq z \leq m, x \in \Gamma, q \in Q} [z,t,x,q] \qquad \text{“mindestens”}$$

“wenn Kopf auf Zelle z kommt, so nicht auf z+1, z+2: höchstens”

$$\overline{[z,t,x,q]} \vee (\bigvee_{x \in \Gamma} [z+1,t,x,X] \wedge \bigvee_{x \in \Gamma} [z+2,t,x,X]$$

$\alpha_E \equiv \bigwedge$ “alle solchen Formeln”

(abgesehen von Ungenauigkeiten, ...) kann man nachprüfen:
$w \in L = L(NTM) gdw$:

$$\Phi_w :\equiv \overbrace{\alpha_{A_w} \vee \alpha_F \vee \alpha_{\ddot{u}} \vee \alpha_E}^{\text{evtl. noch in “Normalform” bringen}} \quad \underbrace{\in SAT_1}_{\text{erfüllbar}}$$

(mit α_{A_w}: Anfangssituation f”ur w,
α_F: akzeptierende Stopp-Situation,
$\alpha_{\ddot{u}}$: Situationsübergänge
α_E: Eindeutigkeit)
dann (immer noch):

- von polynomiell (in $\mid w \mid$) beschränkter Länge
- polynomiell (Zeit-) beschränkt berechenbar aus w (bei gegebener NTM)

bislang: für $\Phi \equiv \bigvee_j \bigwedge_i \overline{i_{i,j}}$ beliebig lange Disjunktionen erlaubt $\curvearrowright$ jeweils “viel Auswahl” für erfüllbares Literal
nunmehr: Auswahl einschränken: nur Disjunktionen mit Länge $\leq$
3: ?
2: ?
1: offensichtlich polynomiell (Zeit-) beschränkt entscheidbar [keine komplementären Literale wie z.B. $... \vee x_i \vee ... \vee \overline{x_j}$]

Satz 28 $3 - SAT_1$ *ist* **NP**-*vollständig*

Satz 29 $2 - SAT_1 \in \mathbf{P}$

Beh: jede Klausel $\alpha = l_1 \wedge ... \wedge l_k$ mit $k \geq 4$ lässt sich “ersetzten” mit Hilfe “neuer” Aussagenvariablen $y_1, ..., y_{K-3}$ durch
$\beta = (l_1 \vee l_2 \vee y_1) \wedge (l_3 \vee \bar{y_1} \vee y_2) \wedge (l_4 \vee \bar{y_2} \vee y_3) ... \wedge (l_{K-2} \vee \bar{y_{K-4}} \vee y_{k-3}) \wedge (l_{k_1} \vee \bar{y_{k-3}} \vee l_k)$
derart, dass gilt: $\forall$ bel. $\mathcal{L}$
$\mathcal{L}(\alpha) = 1$ gdw. ex. Erweiterung $\tilde{\mathcal{L}}$ von $\mathcal{L}$ mit $\tilde{\mathcal{L}}(\beta) = 1$
“$\Rightarrow$” Sei $\mathcal{L}(\alpha) = 1 \underset{\alpha\text{Disjunktion}}{\curvearrowright}$ ex. i mit $\mathcal{L}(l_i)$=1 definiere $\tilde{\mathcal{L}}(y_j) = \begin{cases} 1 & j = 1, ..., i-2 \\ 0 & j = i-1, ..., k-3 \end{cases}$
“$\Leftarrow$”: Sei $\tilde{\mathcal{L}}$ Erweiterung von $\mathcal{L}$ mit $\bar{\mathcal{L}}(\beta) = 1$
zu zeigen: ex. i mit $\mathcal{L}(l_i) = 1$
Beweis:(indirekt):
angenommen: f. alle i: $\mathcal{L}(l_i) = 0 \curvearrowright \tilde{\mathcal{L}}(y_1) = 1$ [sonst $l_1 \wedge l_2 \wedge y_1$ nicht erfüllt]
$\curvearrowright \tilde{\mathcal{L}}(y_2) = 1$ [sonst $l_3 \vee \bar{y_1} \vee y_2$ nicht erfüllt]
... (Induktion)
$\curvearrowright \mathcal{L}(y_{k-3} = 1$ [sonst ...]

$\curvearrowright \mathcal{L}(l_{k-1} \vee y_{k-3} \vee l_k) = 0$
$\underset{\beta \text{ Konjunktion}}{\curvearrowright} \mathcal{L}(\beta) = 0$ also Widerspruch...
$(x_0 \wedge x_3) \vee (x_1 \wedge \bar{x_2}) \vee (\bar{x_1} \vee \bar{x_2} \vee \bar{x_3}) \wedge x_3$ (vgl. Abb 8)
entferne Einerklausen: $(x_1 \wedge \bar{x_2}) \vee (\bar{x_1} \vee x_2) \wedge (\bar{x_1} \vee \bar{x_2})$

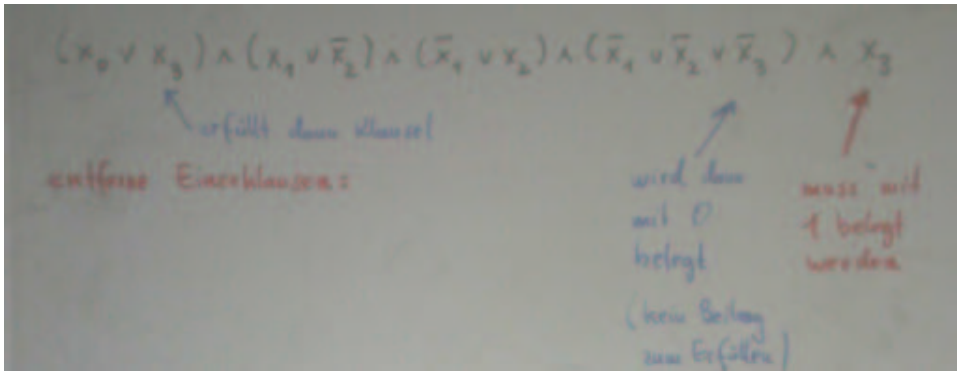

Abbildung 8: Anmerkungen zur Formel

splitte, d.h. Fallunterscheidung für z.B. x_1 (vgl. Abb. 9)

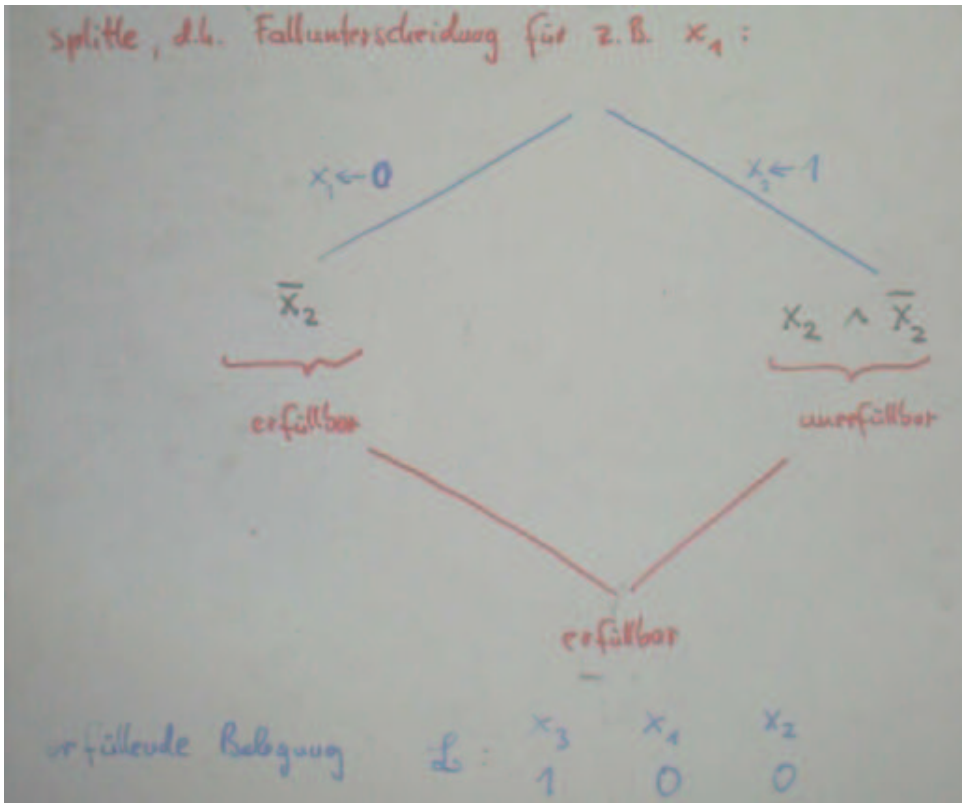

Abbildung 9: Fallunterscheidung

Index