Joachim Biskup

ISSI, Informationssysteme und Sicherheit

Grundbegriffe? "auf den Begriff bringen"

der

theoretischen Informatik?

Informatik beschreibt / schafft

"Formalismus - Wirklichkeit"

- algorithmisch
- maschinell berechenbar

⋮

Uni Do : "Krönung" des Grundstudiums, 4. Sem.

GTI

Software-Praktikum

← grundlegende Abstraktionen,
deren Formalisierung (Automatisierung)

→ grundlegende { Strukturen
Verfahrensweisen

computer (computing)

science ⟷ engineering

- Welche Aufgabenklassen maschinell bearbeitbar ?
- Welche davon effizient ?
- Welche sogar mit endlichem Speicher ?

- Einordnung / Behandlung von:
  - Festlegung von Programmiersprachen
  - Übersetzung von Programmen

- zu schwierig
- gedanklich
- tatsächlich
- einfach

was ist "berechenbar" ?

zur Erinnerung:

- von-Neumann Universalrechner
- programm-gesteuert : kann beliebige Programme ausführen

  digital : operiert auf Worten (über {0,1})

- grundlegende Fähigkeiten:
  - Speicherzellen adressieren
  - Worte (Datum / Befehl) holen (lesen), wegbringen (schreiben)
  - Steuerung durch Sequenzbildung, bedingte Sprünge
  - Arithmetik

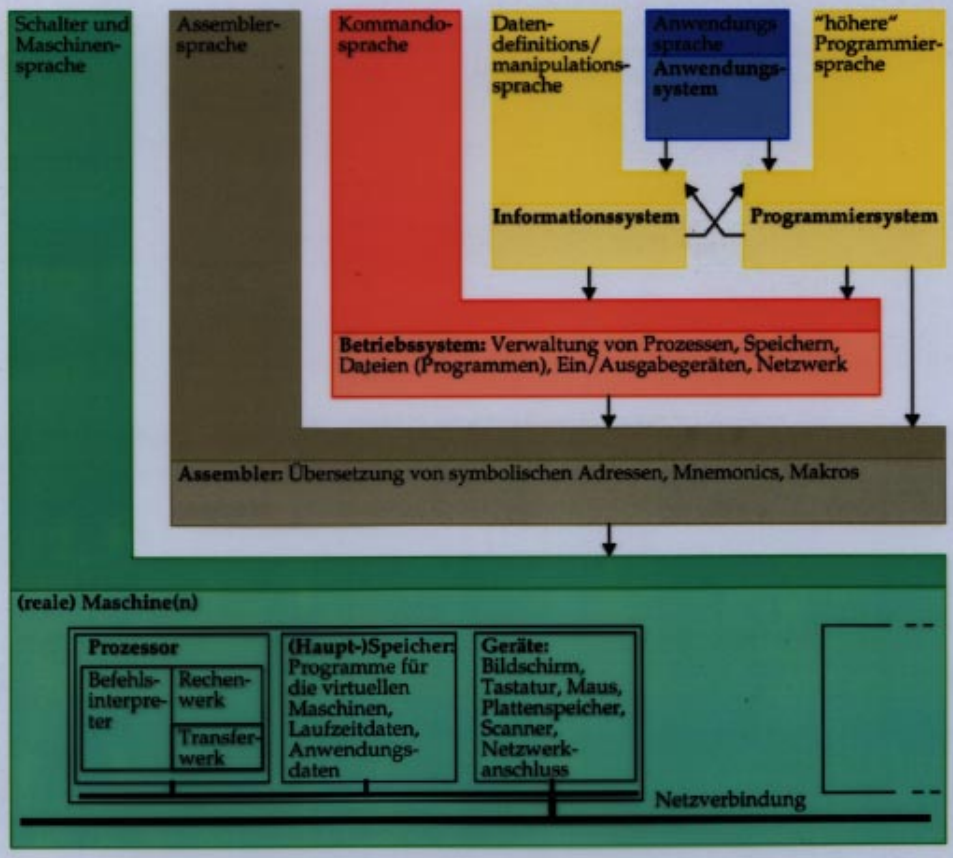


ABBILDUNG 0.1

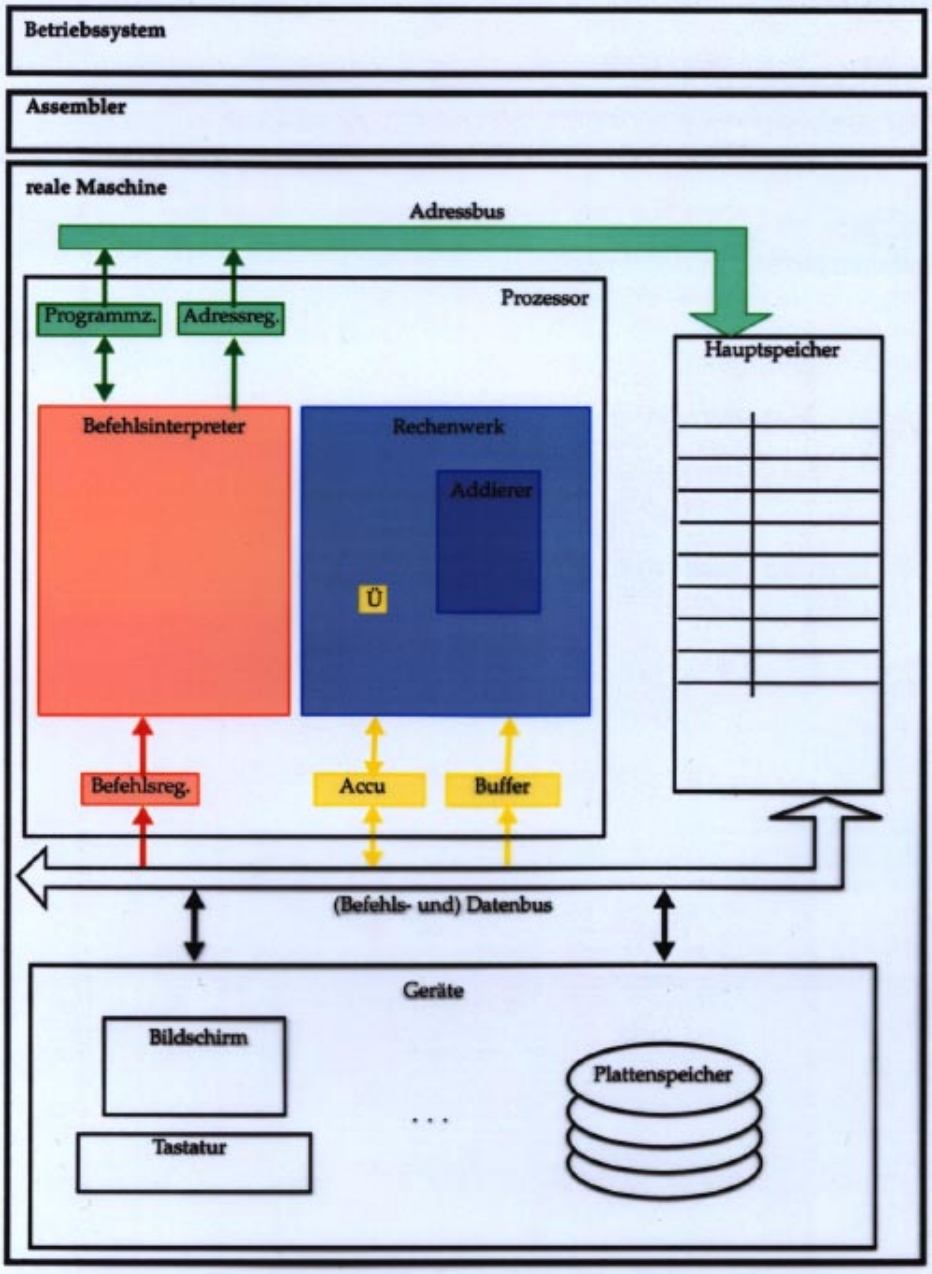


ABBILDUNG 0.5

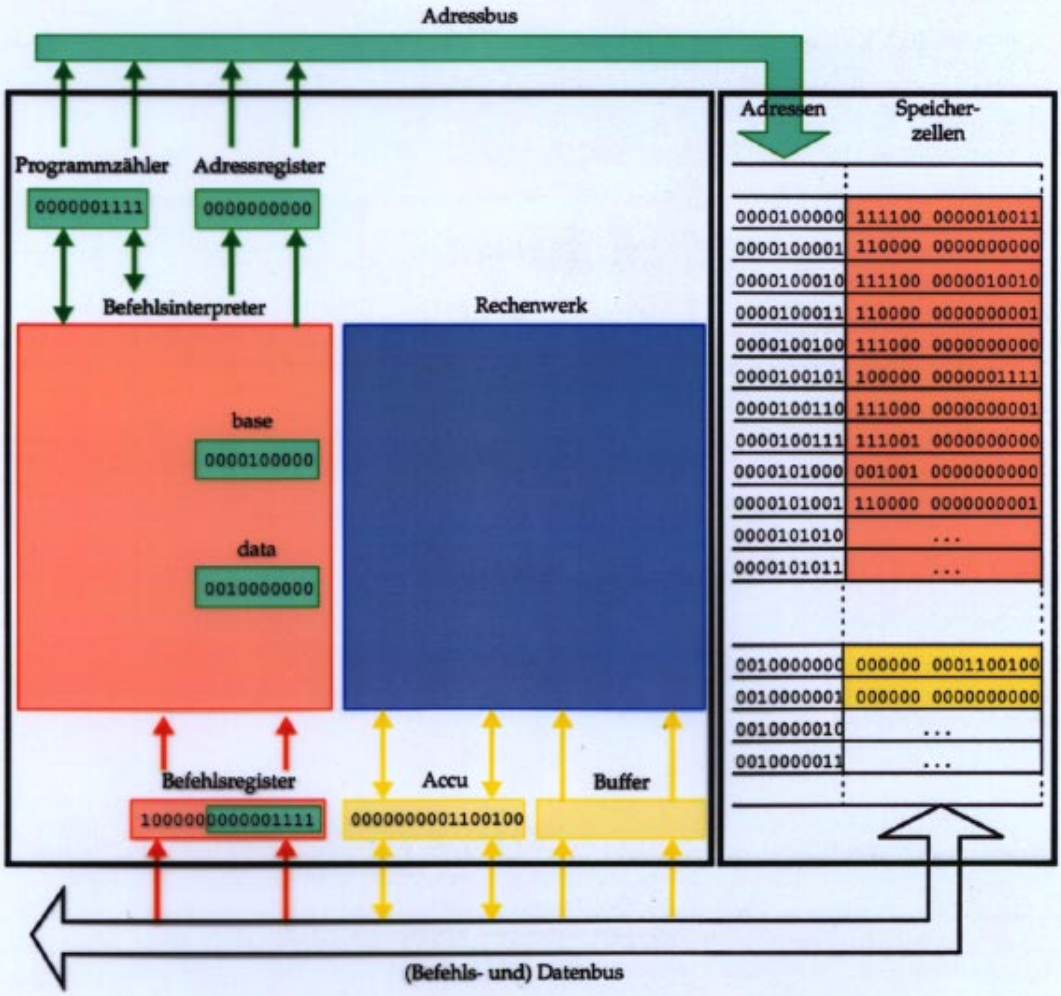


ABBILDUNG 0.2

ein Turing-Programm mit Befehlsinterpreter

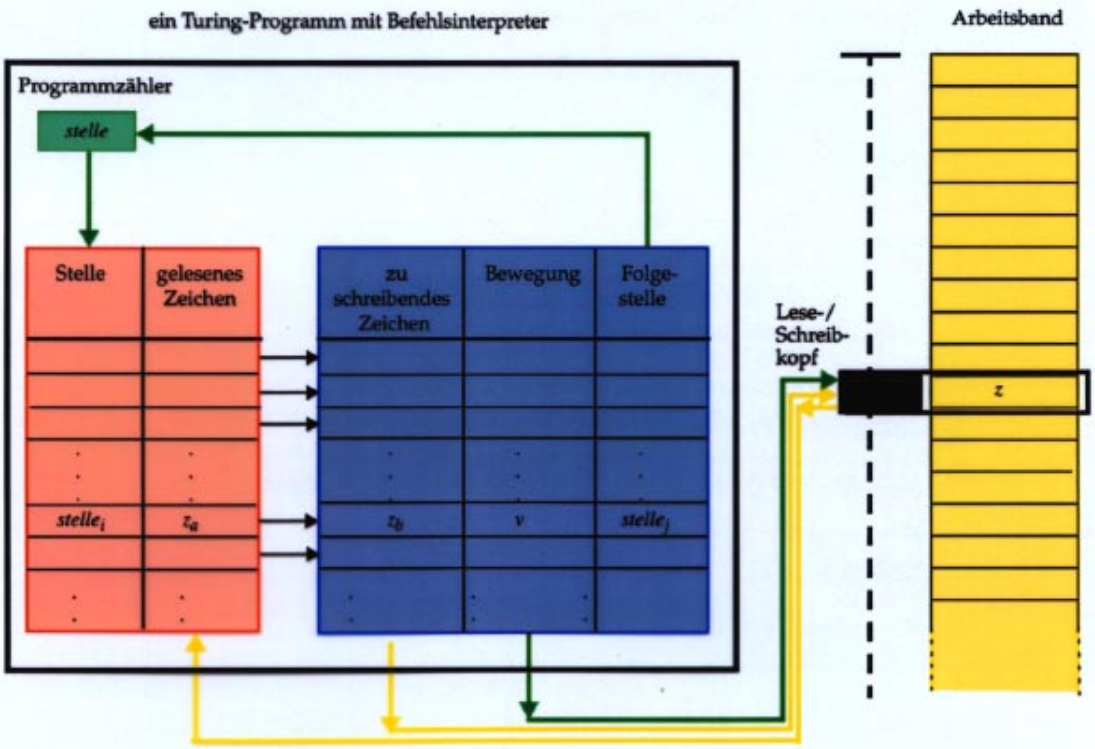


ABBILDUNG 0.4

- Turing-Programme TM **berechnet** Funktion

$\sigma(TM) : \Sigma^* \xrightarrow{\text{partiell}} (\Gamma \setminus \{B\})^*$ vermöge:

Eingabewort $\mapsto$ Ausgangssituation $\mapsto$ $\begin{cases} \uparrow \text{ (undefiniert)} \\ \text{Stopp-Situation} \mapsto \text{Ausgabewort} \end{cases}$

- Turing-Programme TM **semi-entscheidet** Sprache

$\mathcal{L}(TM) \subset \Sigma^*$ vermöge: $w \in \mathcal{L}(TM)$:gdw

Eingabewort $w \mapsto$ Ausgangssituation $\mapsto$ $\begin{cases} \\ \text{Stopp-Situation} \mapsto \text{Ausgabewort} = 1 \end{cases}$

- Turing-Programme TM hat **Laufzeit (-Verhalten)**

$Time(TM) : \Sigma^* \xrightarrow{\text{partiell}} \mathbb{N}$, Eingabewort $w \mapsto \#(\text{Schritte bis Stoppen})$

häufig auch $: \mathbb{N} \xrightarrow{\text{partiell}} \mathbb{N}$, Eingabegröße $n \mapsto \max_{w \in \Sigma^n} Time(TM)(w)$

(eine ansatzweise) formale Definition des Berechenbaren

- $f: \Sigma^* \xrightarrow{p} \Sigma^*$ heißt (partiell) rekursiv / berechenbar

  :gdw ex. TM mit $\sigma(TM) = f$

- $f: \mathbb{N}^k \xrightarrow{p} \mathbb{N}$ heißt (partiell) rekursiv / berechenbar

  :gdw ex. TM mit $\sigma(TM)(\,bin(i_1) \# bin(i_2) \ldots \# bin(i_k)\,)$

  ↑ Binärdarstellung (o.B.d.A)

  ↑ Trennzeichen (o.B.d.A)

  $= bin(\,f(i_1, \ldots, i_k)\,)$ f.alle $(i_1, \ldots, i_k) \in \mathbb{N}^k$

- $L \subset \Sigma^*$ heißt **rekursiv / entscheidbar**

  :gdw ex. TM mit

  $$\underbrace{\sigma(TM)(x)}_{\text{totale Funktion!}} = \chi_L(x) := \begin{cases} 1 & \text{falls } x \in L \\ 0 & \text{falls } x \notin L \end{cases}$$

- $L \subset \Sigma^*$ heißt **rekursiv aufzählbar / semi-entscheidbar**

  :gdw ex. TM mit

  $$\underbrace{\sigma(TM)(x)}_{\text{möglicherweise partielle Funktion!}} = 1 \quad \text{gdw} \quad x \in L$$

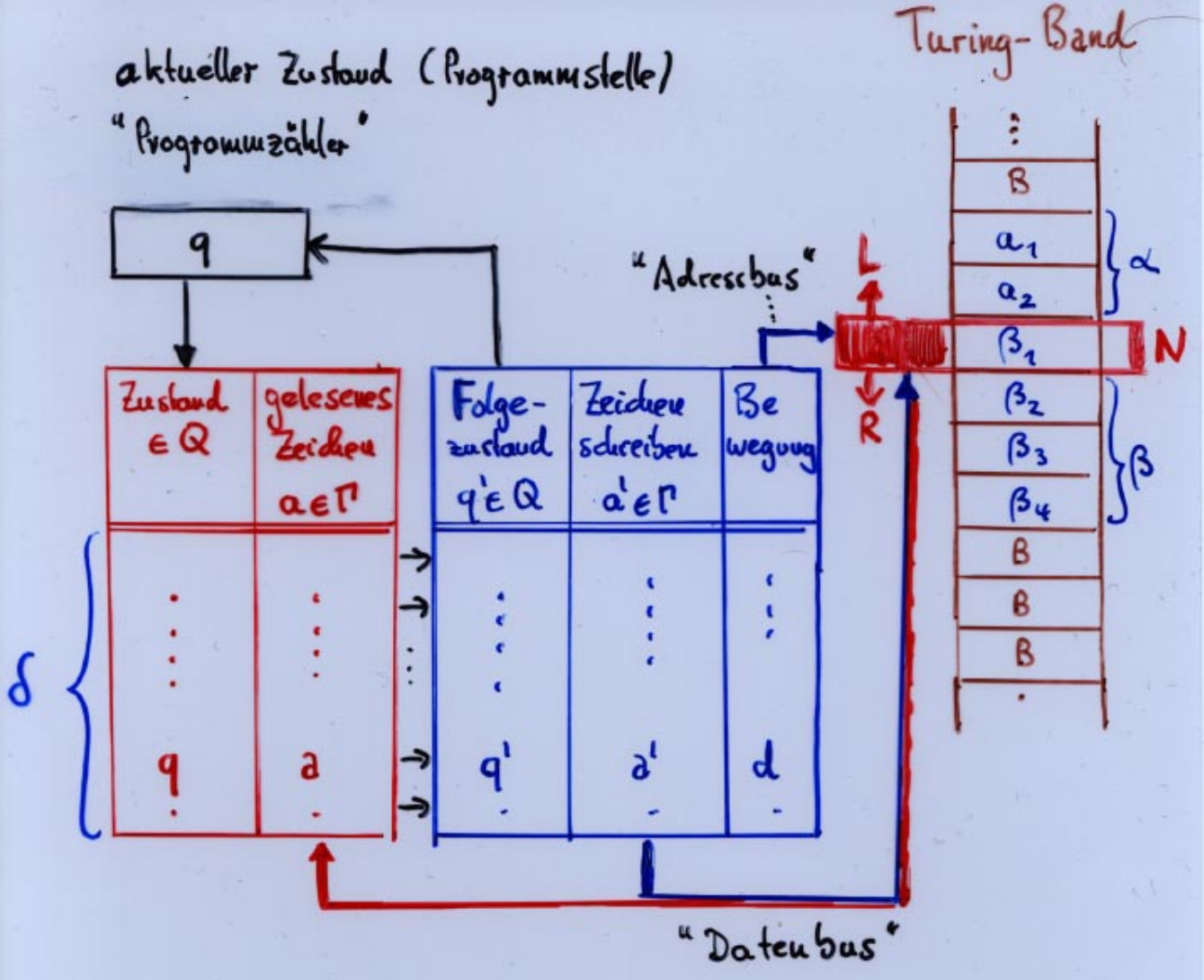
Turing-Programmierung und Varianten

"alle Tricks" der üblichen Programmierung:

- Makros (als Abkürzung)
- Unterprogramm-Aufruf
- komplexe Datentypen ⇝ **Mehrspur-Maschine** für $\Gamma \subset \Sigma_1 \times \Sigma_2 \times \ldots \times \Sigma_\ell$ mit **beliebigen Alphabeten**
- Kontrollstrukturen, insbesondere Wiederholungen
  [bedingte Sprünge bereits nach Definition!]
- tabellenartige Speicher [wie CASE-Kontrollstruktur]

⇝ **Mehrband-Maschinen**

⋮



Syntax von Turing-Programme TM

$$\delta : Q \times \Gamma \longrightarrow Q \times \Gamma \times \{R, L, N\}$$ Zustandsüberführung

$q_0 \in Q$ Anfangszustand

Semantik von TM

$$\sigma(TM) : \Sigma^* \xrightarrow{p} (\Gamma \setminus \{B\})^*$$

Eingabewort $\longmapsto$ Ausgangssituation $\longmapsto$ { ↑ ; Stopp-Situation $\longmapsto$ Ausgabewort }

$w \in \Sigma^*$ mit $\Sigma \subset \Gamma \setminus \{B\}$

Church'sche These

"berechenbar schlechthin" = durch ein Turing-Programm bearbeitbar

Rechtfertigung(en):

① alle anderen Formalisierungen wurden

als äquivalent bewiesen

durch konstruktive Simulation (ebenfalls 'berechenbar')

② - minimalistischer Ansatz ⇝ tatsächlich berechenbar

- beliebig anreicherbar ⇝ alles erfasst

⋮

gilt insbesondere auch

für "programm-gesteuert"

gesucht: Turing-Programm UTM mit

$\sigma(UTM)$: "Turing-Programme" $\times$ "Daten" $\longrightarrow$ "Daten"

$\hat{=} \Gamma_1^*$ $\hat{=} \Gamma_2^*$ $\hat{=} \Gamma_3^*$ geeignet kodieren

$\sigma(UTM)$: $\Gamma^* \longrightarrow \Gamma^*$

Eingabe Ausgabe

$\sigma(UTM)(\underbrace{code(TM), x}_{\in \Gamma^*}) = \sigma(TM)(x)$

$\in \Gamma^* \quad \in \Gamma \quad \in \Gamma^*$

f.alle TM, f.alle x

$\sigma(UTM)(w, x) := \underline{error}_{\in \Gamma}$ f.alle $w \in \Gamma^*$, die kein "korrektes" Turing-Programm sind]

gefunden: • CPU bewirkt "Berechenbares"

Church ↝ ex. UTM mit UTM $\hat{=}$ CPU

• programmiere CPU als Turing-Programm, nutze dabei alle "Tricks"

Gibt es Nicht-Berechenbares ?

ja ! weniger Programme : $\| \Gamma^* \|$ (für geeignete Codierung)

als Funktionen : $\| \Sigma^{\Sigma^*} \|$

genauere Überlegung:

- wähle Alphabet, geeignet für Codierung : Γ
- jedes $w \in \Gamma^*$ deutbar als
  - Turing-Programm (ggf. nicht "korrekt")
  - Datum
- "Selbstanwendung" : $\underbrace{\sigma(w)}_{\text{Semantik}}(w)$

  $\sigma(w)$ als Programm, w als Datum

  $= \sigma(UTM)(w,w)$
- als "Akzeptor" : $\sigma(w)(w) = 1$ gdw "w akzeptiert w"

Selbstanwendung:

Diagonalisierung: $D = \{ w \mid \underbrace{w \text{ akzeptiert } w \text{ nicht}}_{\sigma(w)(w) \neq 1} \}$ (1)

Satz: D ist nicht rekursiv

Korollar: $\overline{D}$ ist nicht rekursiv

Halteproblem:

$$H := \{ \underbrace{TM, x}_{\in \Gamma^*} \mid \underbrace{\sigma(TM)(x) \text{ definiert}}_{TM \text{ hält bei Eingabe von } x} \}$$

(o.B.d.A geeignet gewählt)

Satz: H ist nicht rekursiv

Beweis: durch **Reduktion:**

angenommen: H rekursiv vermöge TM_H

dann auch: $\overline{D}$ rekursiv mit Hilfe von TM_H

also: ↯

Satz: H ist rekursiv aufzählbar

Korollar: $\{ L \mid L \text{ rekursiv} \} \subsetneq \{ L \mid L \text{ rek. aufz.} \}$

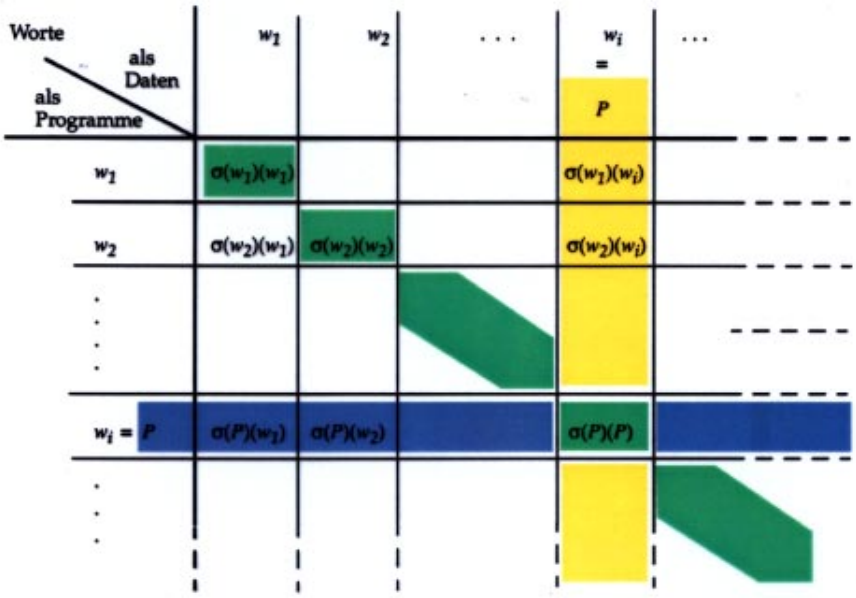


ABBILDUNG 0.3

eine (berühmte, wichtige) Anwendung:

Logik: Sprachen für

- Formeln: $F \subset \Sigma^*$ (geeignet gewählt)
- Aussagen: $A \subset F$
- Theoreme mit Beweis: $TB \subset A \times \Sigma^* \times \Sigma^*$ (geeignet gewählt)
- Theoreme: $T \subset A$

man kann beweisen:

F rekursiv: "Compiler" bauen

A rekursiv: "leichte syntaktische Überprüfung" (keine "freien Vorkommen" von Variablen)

TB rekursiv: "korrekte Einzelbeweisschritte" leicht erkennbar; es gibt "nur wenige Grundtypen" davon!

T rek. aufz., nicht rekursiv: "Beweis finden: zu schwer":

- simuliere Turing-Programme durch Aussagen
- reduziere "Halteproblem" auf "Beweis finden"

Partiellität und Nicht-Rekursivität

$f: \Sigma^* \xrightarrow{p} \Sigma^*$

partiell rekursiv

:gdw ex. TM mit

$\sigma(TM) = f$

f.alle x: $\sigma(TM)(x) = f(x)$

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

(und falls definiert, dann gleich)

Haltemenge :=

$\{TM, x \mid \sigma(TM)(x) \text{ definiert}\}$

ist nicht rekursiv

f.alle TM: $\sigma(TM) \neq \chi_H$

f.alle TM ex. x:

$\sigma(TM)(x) \neq \chi_H(x)$

möglicherweise undefiniert — immer definiert

Fall 1: $\sigma(TM)(x)$ undef.

Fall 2: $\sigma(TM)(x)$ definiert, aber verschieden von $\chi_H(x)$

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

Partiellität und Universalität

es gibt universelle UTM:

$$\underbrace{\sigma(UTM)(TM,x)}_{\text{möglicherweise undefiniert}} = \underbrace{\sigma(TM)(x)}_{\text{(immer beide Seiten zusammen)}}$$

(fiktive) Beschränkung auf alle totalen Funktionen: ↝ erlaubt kein universelles Programm in der gleichen Sprache!

Programmiersprache für ausschließlich totale Fkt:

$$\left(\underbrace{\tau(w)}_{\text{berechnete totale Funktion}}\right)_{w \in \Sigma^*}$$

(Semantik; w: „Programm“)

Beweis: Diagonalisierung

indirekte Annahme: $TU \in \Sigma^*$ („Programm“) universell für Sprache

ex. $d \in \Sigma^*$ („Programm“): $\tau(d)(x) = \underbrace{\tau(TU)(x,x)}_{=\tau(x)(x)} \text{“}+1\text{“} \neq \tau(x)(x)$

für $x := d$ ↝ $\tau(d)(d) \neq \tau(d)(d)$ ↯

$L \subset \Sigma^*$

rekursiv aufzählbar

ex. TM mit

$\sigma(TM)(x) = 1$ gdw $x \in L$

$\sigma(TM)(x)$: möglicherweise partiell

rekursiv

ex. TM mit

$$\sigma(TM)(x) = \begin{cases} 1 & x \in L \\ 0 & x \notin L \end{cases}$$

$\sigma(TM)(x)$: total

- Partiellität i. allg. nicht überwindbar:

  $H_{\text{alternative}} := \{ TM, x \mid \sigma(TM)(x) \text{ definiert} \}$

  rek. aufz., aber nicht rekursiv

- "komplementäre Partiellität" überwindbar:

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

Beweis:

"$\Rightarrow$": gemäß Definitionen

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

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

L rekursiv aufzählbar

1. semi-entscheidbar: ex. TM mit: $\sigma(TM)(x) = 1$ gdw $x \in L$

2. χ_L^* partiell rekursiv: ex. TM mit: $\sigma(TM)(x)$

$$= \chi_L^*(x) := \begin{cases} 1 & \text{falls } x \in L \\ \uparrow & \text{sonst} \end{cases}$$

↖ "undefiniert"

3. Definitionsbereich einer partiell rekursiven Fkt: ex. TM mit domain $(\sigma(TM)) = L$

4. Wertebereich einer partiell rekursiven Fkt: ex. TM mit range $(\sigma(TM)) = L$

5. Wertebereich einer total rekursiven Fkt: ex. TM mit $\sigma(TM)$ total und π

6. ex. berechenbares Aufzählungsverfahren: "spuckt nacheinander (potentiell unendlicher Vorgang) genau die Elemente von L aus"

Berechenbarkeit für

Funktionen

$f: \Sigma^* \xrightarrow{p} \Sigma^*$

partiell rekursiv : gdw

ex TM mit $\sigma(TM) = f$

Mengen (Sprachen)

$L \subset \Sigma^*$

rek. aufz. :

ex. TM mit $\sigma(TM) = \chi_L^*$

Menge $L \mapsto$ Funktion χ_L^*

Funktion $f \mapsto$ Funktionsgraph $\{(x,y) \mid y = f(x)\} \subset \Sigma^* \times \Sigma^*$

graph(f) :=

$\{(x,y)\} \approx \widetilde{\Sigma}^*$

mit geeigneter Codierung

Satz f partiell rekursiv

gdw graph(f) rek. aufz.

Satz wenn f total rekursiv

dann graph(f) rekursiv

Formalismus für das "Berechenbare"

("universelle" Programmiersprache)

- abstrahiert "reale Rechner / Programmiersprachen";
- als Turing-Programme mathematisierbar;
- erlaubt universelles Programm und damit
- Selbstanwendung und Diagonalisierung;
- beinhaltet notwendigerweise Partiellität und
- demensprechend Nicht-Rekursivität der Definiertheit;
- erlaubt zu unterscheiden zwischen

rekursiv	–	rekursiv aufzählbar	–	nicht rekursiv aufzählbar

z.B.

$\{0^n 1^n \mid n \geq 1\}$ – HalteMenge – $\overline{\text{HaltMenge}}$

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

ja, vermöge berechenbarer "Programmtransformationen"

Rekursionssatz

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

$$\sigma(w) = \sigma(g(w))$$

Programm und transformiertes Programm haben gleiche Semantik!

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

"Spielanwendung" mit ernsthaftem Hintergrund:

gibt es TM mit $\sigma(TM)(x)$ hält gdw $x = TM$?

[TM "zählt" genau sich selbst (eigene Beschreibung) auf]

Hintergrund: "Selbstreproduktion möglich?"

solches TM mit Rekursionssatz bestimmbar:

- betrachte "Programmtransformation" g mit

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

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

↝ (Rekursionssatz) ex. w mit $\sigma(\, g(w)\,) = \sigma(w)$

↝ $\sigma(w)(x) = \sigma(\, g(w))(x) = \begin{cases} 1 & \text{falls } x = w \\ \uparrow \end{cases}$

↝ $\sigma(w)(x)$ hält gdw $x = w$

"Superanwendung" für Kernprobleme der Informatik:

welche Eigenschaften von Programmen sind entscheidbar?

- ex. x mit $\sigma(TM)(x)$ definiert ?
  "TM nichttrivial ?"
- f. alle x : $\sigma(TM)(x)$ definiert ?
  "TM total" ?
- ex. unendlich viele x mit $\sigma(TM)(x)$ definiert ?
  "TM unendlich" ?

- $\sigma(TM_1) = \sigma(TM_2)$?

  f. alle x : $\sigma(TM_1)(x) = \sigma(TM_2)(x)$?
  "TM_1 und TM_2 äquivalent" ?
- f. alle x : wenn $\sigma(TM_1)(x)$ definiert, dann auch $\sigma(TM_2)(x)$ 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")

$PR := \{ \sigma(w) \mid w \in \Gamma^* \}$ alle partiell rekursiven Fkt. (jeweils einstellig gedeutet)

Satz von Rice:

Sei $\emptyset \neq \gamma \subsetneq PR$ nichttriviale Menge partiell rekursiver Fkt.

Dann ist die zugehörige Programm-Menge

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

nicht rekursiv.

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

1) "nichttrivial"?
   "total"?
   "unendlich"?
   ↝ direkt

2) "äquivalent"?
   "enthalten"?
   ↝ mit Reduktion, z.B.:

als Eigenschaft von TM_1 und TM_2

angenommen: "$\sigma(TM_1) = \sigma(TM_2)$" wäre entscheidbar

wähle: $\overline{TM_2}$ fest derart, dass $\sigma(\overline{TM_2})(x) = \uparrow$ f.alle x

dann: "$\sigma(TM) = \sigma(\overline{TM_2})$ ebenfalls entscheidbar

als Eigenschaft von TM

dann: "f.alle x: $\sigma(TM)(x)$ undefiniert" entscheidbar

↝ ↯ Satz von Rice

was ist berechenbar ?

eine "maschinennahe" Abstraktion:

Turing-Maschine mit Turing-Programmen

$PR := \{ f : \Sigma^* \rightharpoonup \Sigma^* \mid$ ex. TM mit $\sigma(TM) = f \}$

im Wesentlichen: $\delta : Q \times \Gamma \to Q \times \Gamma \times \{R, L, N\}$

mit $q_0 \in Q$

```
stelle := q0;
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 ?

- { durch "Programme beschreibbar" / durch "Maschine bearbeitbar" } ↝ Turing-Programme

⋮ (viele andere, stets äquivalente Abstraktionen)

- durch "berechenbare Zusammensetzung" ↝ Funktionale
  von "offensichtlich Berechenbarem" ↝ Grundfunktionen

---

ausgehend von Grundfunktionen
den Abschluss unter
Funktionalen bilden !

Exkurs: Funktionsterme versus Funktionen

z.B.: $+(x,y)$ soll bedeuten Additionsfunktion

z.B. $\bullet(x,+(y,1))$ " Komposition von Nachfolgerfunktion bzg. y mit Multiplikationsfunktion

Unterscheidung häufig nur implizit: Syntax versus Semantik

Grundfunktionen (über Alphabet Σ)

- **Nullfunktion** (erase)

$$E : \Sigma^* \to \Sigma^*$$

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

λ: "leeres Wort", neutrales Element bzg. Konkatenation

- **Nachfolgefunktionen** für $a \in \Sigma$:

$$S_a : \Sigma^* \to \Sigma^*$$

$$S_a(x) = x\,a \quad \text{f.alle } x \in \Sigma^*$$

$x\,a$: Konkatenation

- **Projektionsfunktionen** für $1 \le j \le n$:

$$P_j^n : \Sigma^* \times \dots \times \Sigma^* \times \dots \times \Sigma^* \to \Sigma^*$$

$$P_j^n(x_1, \dots, x_j, \dots, x_n) := x_j$$

f.alle $x_1, \dots, x_j, \dots x_n \in \Sigma^*$

Funktionale (für Funktionen über Σ)

- **Substitution** (Komposition): seien

$$g: \Sigma^{*m} \to \Sigma^*$$
$$h_i: \Sigma^{*n} \to \Sigma^* \quad \text{für } i = 1, \dots, m;$$

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

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

kurz: $f := S(g, h_1, \dots, h_m)$

Funktionale, Fortsetzung:

- **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 $g, (h_a)_{a\in\Sigma}$:

$$f: \Sigma^{*\,n} \to \Sigma^*$$

$$f(\lambda, x_2, \dots, x_n) := g(\underbrace{x_2, \dots, x_n}_{\text{"Parameter"}})$$

↑ "Rekursionsanfang"

$$f(ya, x_2, \dots, x_n) :=$$

"Rekursionsschritt"

Ergebnis für "Restargument"

$$h_a(y, \overbrace{f(y, x_2, \dots, x_n)}, \underbrace{x_2, \dots, x_n}_{\text{"Parameter"}})$$

↑ "Restargument"

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

speziell n = 1: $f(\lambda) := g$ mit $g \in \Sigma^*$

$$f(ya) := h_a(y, f(y))$$

Definition

$f : \Sigma^{*n} \to \Sigma^*$ heißt primitiv-rekursiv : gdw

f kann aus den Grundfunktionen E, S_2, P^n_j

durch (endliche) Anwendung der Funktionale $\mathcal{L}$, $\mathcal{PR}$

gewonnen werden.

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

Beweis-Ansätze:

① Church'sche These anwenden: "alles programmierbar".

② (i) Turing-Programme für Grundfunktionen angeben

(ii) Turing-Programme für Funktionale angeben, als "(Turing-) Programmtransformationen"

③ Vergleich mit "üblichen Programmiersprachen":

$\mathcal{L}$: Assignment, StatementSequence / Expression

$\mathcal{PR}$: BoundedLoop (ForStatement)

Beispiele:

$\Sigma := \{1\} \quad \rightsquigarrow \quad \Sigma^* \approx \mathbb{N}$

$\Sigma^* \approx \mathbb{N}$ per Strichcodierung ("Bierdeckel-Codierung")

a) arithmetische Nachfolger-Funktion:

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

↑ "Wort-Nachfolgefunktion" für $1 \in \Sigma$

b) arithmetische Addition:

Ergebnis für "Restargument"

Vorüberlegung: $(y+1) + w = (y+w) + 1$

"Restargument" — als "Parameter" benutzen

formal:

durch "Primitive Rekursion":

$$Add(\lambda, w) = w = P^1_1(w)$$

$$Add(y1, w) = S_1(Add(y, w))$$

$$= S_1\big(\underbrace{P^3_2(y, \underbrace{Add(y,w)}_{\text{Add}(y,w)}, w)}_{\text{Add}(y,w)}\big)$$

durch "Komposition"

c) arithmetische Multiplikation:

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

↑ schon als primitiv-rekursiv nachgewiesen

formal:

$$\text{Mult}(\lambda, w) = \overbrace{\lambda}^{\approx 0} = E(w)$$

$$\text{Mult}(y1, w) = \text{Add}(\text{Mult}(y,w), w)$$

$$= \underline{\text{Add}}\left(\underline{P_2^3(y, \underline{\text{Mult}(y,w)}, w)}, P_3^3(y, \text{Mult}(y,w), \underline{w})\right)$$

Beispiele, Fortsetzung:

nunmehr $\Sigma := \{0,1\} \rightsquigarrow \Sigma^* \approx \mathbb{N}$

$\uparrow$ per Binärcodierung

a') arithmetische Nachfolger-Funktion:

$$Succ(\lambda) = 1 = S_1(E(\lambda))$$

$\in \Sigma \subset \Sigma^*$

falls "erwünscht": durch Grundfunktionen ausdrückbar

$$\underbrace{Succ(w0)}_{\begin{array}{r} w\,0 \\ +\quad 1 \\ \hline w\,1 \end{array}} = w1 = S_1(w) = S_1\big(\underbrace{P_1^2(w, Succ(w))}_{w}\big)$$

"von Form $h_1(w, Succ(w))$"

$$\underbrace{Succ(w1)}_{\begin{array}{r} w \\ +\quad 1\,1 \\ \hline Succ(w)\,0 \end{array}} = Succ(w)\,0 = S_0(Succ(w)) = S_0\big(P_2^2(w, Succ(w))\big)$$

($w = w_n \ldots w_1$)

"berechenbar" $\stackrel{?}{=}$ primitiv-rekursiv

natürlich nicht !

umfosst "Partiellität" — stets total

was fehlt ?

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

↝ Abstraktion als weiteres Funktional

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

sei $\quad \psi : \Sigma^{*\,n+1} \xrightarrow[p]{} \Sigma^*$;

↑ (möglicherweise) partiell !

ϕ entsteht durch Minimierung aus ψ :

("Anwendung des μ-Operators")

$$\phi : \Sigma^{*\,n} \xrightarrow[p]{} \Sigma^*$$

↖ !

$$\phi(x_1,\dots,x_n) := \begin{cases} a^m & \text{mit } m \in \mathbb{N} \text{ minimal bzg:} \\ & - \text{f.alle } \rho \leq m : \\ & \quad \psi(a^\rho, x_1,\dots,x_n)\downarrow \\ & - \psi(a^m, x_1,\dots,x_n) = \lambda \\ & \text{falls so ein } m \text{ existiert} \\ \uparrow & \text{sonst} \end{cases}$$

kurz : $\phi := \mathrm{Min}_a\, \psi$

Grundfunktionen:

Nullfunktion: $E(x) := \lambda$

Nachfolgefunktionen: $S_a(x) := xa$, für $a \in \Sigma$

Projektionsfunktionen: $P^n_j(x_1, \dots, x_j, \dots, x_n) := x_j$, für $1 \leq j \leq n$

Funktionale:

Substitution (Komposition): $f = K(g, h_1, \dots, h_m)$

$$f(x_1, \dots x_n) := g\left(h_1(x_1, \dots, x_n), \dots, h_m(x_1, \dots, x_m)\right)$$

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

$$f(\lambda, x_2, \dots, x_n) := g(x_2, \dots, x_n)$$

$$f(ya, x_2, \dots, x_n) := h_a(y, f(y, x_2, \dots, x_n), x_2, \dots, x_n)$$

Minimierung (über $\{a^x\}$): $\phi = Min_a\, \psi$

$$\phi(x_1, \dots, x_n) := \begin{cases} a^m & \text{mit } m \in \mathbb{N} \text{ minimal bzg:} \\ & - \text{f. alle } p \leq m: \\ & \quad \psi(a^p, x_1, \dots x_n) \downarrow \\ & - \psi(a^m, x_1, \dots, x_n) = \lambda \\ \uparrow & \text{sonst, d.h. solch } m \text{ existiert } \underline{\text{nicht}} \end{cases}$$

Definition

$\phi : \Sigma^{*n} \underset{\rho}{\rightarrow} \Sigma^*$ heißt μ-rekursiv : gdw

ϕ kann aus den Grundfunktionen E, S_a, P^n_j

durch endliche Anwendung der Funktionale $\mathcal{L}$, $\mathcal{PR}$, Min_a

gewonnen werden.

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

Beweis-Ansatz:

Church'sche These mit

$Min_a \psi \triangleq$ "unbeschränkte Wiederholung":

```
out := λ ;
while ψ(out, x_1, ..., x_n) ≠ λ do      (kann "versanden")
    out := S_a(out)
od                                      (kann "schleifen")
```

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

Beweis-Ansatz:

betrachte universelles Turing-Programm UTM:

$$\sigma(UTM)(TM, w) = \sigma(TM)(w)$$

(a) UTM stellt Anfangskonfiguration für TM, w her: $\varepsilon\, q_0\, w$

(b) UTM interpretiert TM schrittweise

(c) UTM stoppt, wenn erstmals Stopp-Situation auftritt

(d) UTM erzeugt Ausgabe

dann nachprüfbar (siehe auch "maschinennahe Abstraktion"):

(a), (b) für einen Schritt, (d): primitiv-rekursiv

(c): Minimierung

↷ μ-rekursiv

Kleene'scher Normalform-Satz

Jede partiell rekursive Funktion

hat eine Darstellung (als Funktionsterm)

"mit primitiv-rekursiven Funktionen

und einer Minimierung".

(dynamische) Komplexitätsmaße

Klassifikation von "Problemen":

- Funktionen
- Sprachen (Wortmengen)

⋮

nicht rekursiv aufzählbar

rekursiv aufzählbar

rekursiv: ⋮

"eher nicht effizient"

⋮

"eher effizient"

⋮

Abstraktion für "Effizienz"?

z.B. für Turing-Programme:

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

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

für Turing-Programme gilt:

- für TM, die "volle Eingabe lesen":

  $\text{Space}(TM)(w) \leq \text{Time}(TM)(w)$ f. alle $w \in \Sigma^*$

- falls $\text{Time}(TM)(w) \downarrow$,

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

$k := \#(\text{benutzte Zellen})$
$\ell := \| \Gamma \|$ Alphabetgröße
$n := \| Q \|$

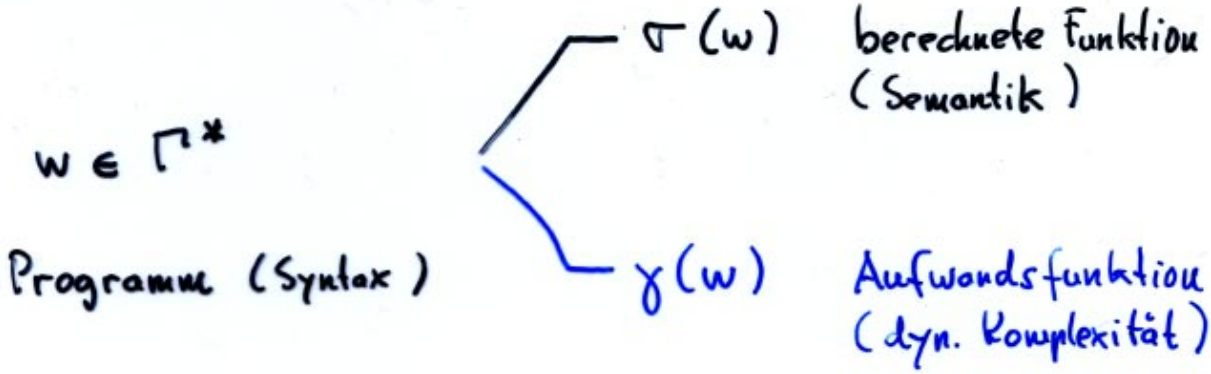
$\} \sim \ell^k$ verschiedene Bandinhalte

$\sim \leq n \cdot k \cdot \ell^k$ verschiedene Situationen

$\rightsquigarrow$ <u>entweder</u> TM stoppt nach höchstens so viel Schritten

<u>oder</u> TM läuft in "Endlos-Schleife"

weitergehende Abstraktion:



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

Formalisierung: Blum'sches Komplexitätsmaß für Gödelnumerierung

- $F = (\sigma(w))_{w \in \Gamma^*}$ sei Aufzählung aller (einstelligen) partiell rekursiven Funktionen (o.B.d.A. vom Typ $\sigma(w): \Gamma^* \xrightarrow{p} \Gamma^*$) mit:

  (i) $U(w,x) := \sigma(w)(x)$ ist partiell rekursiv

  (ii) ex. total rekursive Fkt c mit: $\sigma(c(u,v)) = \sigma(u) \circ \sigma(v)$

- $\mathcal{M} = (F, \Phi)$ heißt **Komplexitätsmaß** : gdw

  - F ist Gödelnumerierung (wie oben beschrieben)
  - $\Phi = (\gamma(w))_{w \in \Gamma^*}$ ist Aufzählung von (partiell rekursiven) Aufwands-funktionen vom Typ $\gamma(w): \Gamma^* \xrightarrow{p} \mathbb{N}$ mit

(B1) $\sigma(w)(x)\downarrow$ gdw $\gamma(w)(x)\downarrow$

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

(mit geeigneter Codierung)

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

Beispiele: · Time
· Space

Gegenbeispiele · $\gamma^{bin}(w)(x) := \sigma(w)(x)$ ↯ B2, H unentscheidbar

· $\gamma(w)(x) := 1$ ↯ B1

überaus mächtige Abstraktion,

z. B.

"jedes Problem kann beliebig schlecht programmiert werden!"

Satz Sei f eine total rekursive Funktion ["zu programmieren"],

h eine total rekursive (Schranken-) Funktion. ["zu unterbieten"]

Dann ex. [Programm] $w \in \Gamma^*$ mit

a) $f = \sigma(w)$

b) $\gamma(w)(x) > \underbrace{h(x)}_{\text{als nat. Zahl gedeutet}}$ f. alle $x \in \Gamma^*$

"es gibt beliebig aufwändige berechenbare Probleme!"

Satz Sei h eine total rekursive (Schranken-) Funktion.

Dann gibt es eine total rekursive Funktion f mit:

für alle [Programme] $w \in \Gamma^*$:

wenn $\sigma(w) = f$,

dann $\gamma(w)(x) > h(x)$

für fast alle $x \in \Gamma^*$

(bis auf endlich viele Ausnahmen)

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

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

$$\sigma(w)(x) \leq \alpha(x, \gamma(w)(x))$$

für fast alle $x \in \Gamma^*$.

Komplexitätsklassen

Definition Sei α eine total rekursive Funktion.

$$\mathcal{C}_\alpha := \{ f \mid f \text{ total rekursiv},$$

$$\text{ex. [Programm] } w \in \Gamma^* \text{ mit } \sigma(w) = f$$

$$\text{und } \gamma(w)(x) \leq \alpha(x) \text{ f. fast alle } x \}$$

Klar: (i) $\alpha(x) \leq \beta(x) \rightsquigarrow \mathcal{C}_\alpha \subset \mathcal{C}_\beta$

(ii) f. alle α ex. β mit $\mathcal{C}_\alpha \subsetneq \mathcal{C}_\beta$

aber viele "merkwürdige Phänomene", z. B.:

Lückensatz: Sei h total rekursiv mit $h(x) \geq x$.

Dann gibt es eine (beliebig große) total rekursive Fkt α

mit $\mathcal{C}_\alpha = \mathcal{C}_{h \circ \alpha}$.

Beschleunigungs-Satz: Zu jeder total rekursiven Fkt g ("inverse Beschleunigung")

gibt es eine total rekursive Funktion f mit:

f. alle w mit $\sigma(w) = f$

ex. u mit $\sigma(u) = f$ und

$$g(x, \gamma(u)(x)) < \gamma(w)(x) \text{ f. fast alle } x$$

was ist "tatsächlich" / "effizient" berechenbar (entscheidbar) ?

TM Turing-Programme

- σ(TM) — Semantik
- Time(TM) — dynamische Zeit-Aufwands-funktion

Funktion

$\delta: Q \times \Gamma \to Q \times \Gamma \times \{R, L, N\}$,

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 Größe n",

wobei stets $\|\Sigma\| \geq 2$ vorausgesetzt,

d.h. z.B. für Zahlen:

"Länge der Zahldarstellung logarithmisch in Zahlwert

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

Ansatz für "tatsächlich" / "effizient" berechenbar / entscheidbar:

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

$$\text{Time}(TM)(n) \leq P(n) \quad \text{für fast alle } n \in \mathbb{N}$$

für "hinreichend große" n

zugehörige "Problemklasse" (für Sprachen über Σ)

$$\mathbb{P} := \{ L \mid \text{ex. TM mit}$$
$$(1)\ \chi_L = \sigma(TM) \quad \text{TM entscheidet } L$$
$$(2)\ \text{Time}(TM) \text{ polynomiell beschränkt} \}$$

sind "praktische Probleme"

tatsächlich / effizient ~ polynomiell Zeit-beschränkt

berechenbar / entscheidbar ? welche ?

- Addition
- Multiplikation
- ⋮
- Klammerstrukturen
- ⋮

} ja !

- Graph-Suchprobleme
- ⋮
- Optimierungsprobleme
- ⋮

} ?

~ weit geteilte Vermutung:

nein !

Bsp 1:

ungerichteter Graph: $G = (V, E)$ mit (vertices, edges)

$$E \subset \{ \{v_1, v_2\} \mid v_1 \in V, v_2 \in V \}$$

$$CLIQUE = \{ (V,E), k, V' \mid$$

"V' ist Clique der Größe k in Graph (V,E)":

$$V' \subset V \text{ und } \forall \text{alle } v_1, v_2 \in V' : \{v_1, v_2\} \in E \text{ und } \| V' \| = k \}$$

polynomiell (Zeit-) beschränkt entscheidbar!

drei abgeleitete "Probleme":

$$CLIQUE_1 := \{ (V,E), k \mid \text{ex. } V' : [(V,E), k, V'] \in CLIQUE \}$$

"Clique-Größe k in (V,E) erreichbar?"

$$CLIQUE_2 : (V,E) \mapsto \text{Max} \{ k \mid \text{ex. } V' : [(V,E), k, V'] \in CLIQUE \}$$

"berechne optimale Clique-Größe in (V,E)!"

$$CLIQUE_3 : (V,E) \mapsto V' \text{ mit "1) } V' \text{ Clique in } (V,E) \text{ 2) mit optimaler Größe"}$$

"berechne eine Clique optimaler Größe in (V,E)!"

- folgende Programmtransformationen sind offensichtlich:

  Programm für $CLIQUE_3$ (optimale Clique!)

  ↘ Programm für $CLIQUE_2$ (optimale Clique-Größe!)

  abzählen!

  ↘ Programm für $CLIQUE_1$ (Clique-Größe k erreichbar?)

  $k \leq$ optimale Clique-Größe?

- Umkehrungen gelten auch!

- alle "Probleme" erscheinen "aufwändig":

  unmittelbares Überprüfen der Definition

  erfordert Suche in einem

  "exponentiell großen Suchraum"

  [$V' \in \mathcal{P}V$ mit $\|\mathcal{P}V\| = 2^{\|V\|}$]

zu "Umkehrungen gelten auch":

Programm für $CLIQUE_1$ (Clique-Größe k erreichbar?)

↷ Programm für $CLIQUE_2$ (optimale Clique-Größe!)

↳ Eingabe: (V,E)

Methode: 1) bestimme $n := \|V\|$

$[\, CLIQUE_2(V,E) \leq n \,]$

2) für $k := 1, \ldots, n$:

- 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!)

↷ 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" (mit Programm für $CLIQUE_2$) $= k_{opt}$

  dann e endgültig entfernen

  sonst e behalten

Ausgabe: "Restgraph" [bildet optimale Clique]

Beobachtung: entsprechend!

Satz 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

[tatsächliche exponentiell ?],

reicht es in diesem Beispiel (und vielen anderen), das Entscheidungsproblem zu untersuchen:

$$CLIQUE_1 := \{ (V,E), k \mid \text{ex. } V' : [(V,E), k, V'] \in CLIQUE \}$$
$$= proj_{(V,E),k}(CLIQUE)$$

Bsp 2: "Rucksack packen"

n Objekte $1, \dots, n$ — 1 Rucksack mit

mit Gewichten $g_1, \dots, g_n \in \mathbb{N}$ — Gewichtsschranke $G \in \mathbb{N}$

und Nutzen $a_1, \dots, a_n \in \mathbb{N}$ — geforderten Nutzen $A \in \mathbb{N}$

$$\text{KNAPSACK} := \{ g_1, \dots, g_n, a_1, \dots, a_n, G, A, I \mid$$

$I \subset \{1, \dots, n\}$ "Packung" mit

$$\underbrace{\sum_{i \in I} g_i \leq G}_{\text{"packbar"}} \quad \text{und} \quad \underbrace{\sum_{i \in I} a_i \geq A}_{\text{"nützlich"}} \}$$

$$\text{KNAPSACK}_1 := \{ g_1, \dots, g_n, a_1, \dots, a_n, G, A \mid \text{ex. } I \text{ mit } g_1, \dots, G, A, I \in \text{KNAPSACK} \}$$

$= \text{proj}_{g_1, \dots, g_n, a_1, \dots, a_k, G, A}(\text{KNAPSACK})$

"Nützlichkeit A erreichbar?"

$\text{KNAPSACK}_2 : g_1, \dots, g_n, a_1, \dots, a_n, G \mapsto \text{Max}\{ A \mid \dots \}$

"maximale Nützlichkeit!"

$\text{KNAPSACK}_3 : g_1, \dots, g_n, a_1, \dots, a_n, G \mapsto I$ mit "I packbar und maximal nützlich"

"maximal nützliche Packung!"

Bsp. 3: Handelsreisender (Traveling Salesman)

n Orte $1, \dots, n$
mit "Reisekosten" $c(i,j) \in \mathbb{N}$

Graph mit Knoten $\{1,..,n\}$
gerichteten Kanten (i,j),
jeweils mit $c(i,j)$ bewertet

Rundreise
mit Kostenbeschränkung $B \in \mathbb{N}$

Hamiltonkreis, d.h. Permutation
$\pi = \begin{pmatrix} 1 & \cdots & n \\ \pi(1) & \cdots & \pi(n) \end{pmatrix}$ mit

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

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

$$\text{TRAVSALE}_1 := \text{proj}_{(c(i,j))_{\substack{1 \leq i \leq n \\ 1 \leq j \leq n}}, B} (\text{TRAVSALE})$$

"Kostenbeschränkung B erreichbar"

$$\text{TRAVSALE}_2 : (c(i,j))_{\substack{1 \leq i \leq n \\ 1 \leq j \leq n}} \longmapsto \text{Min}\{ B \mid \dots \}$$

"minimale Kosten!"

$$\text{TRAVSALE}_3 : (c(i,j))_{\substack{1 \leq i \leq n \\ 1 \leq j \leq n}} \longmapsto \pi \text{ mit "}\pi \text{ kostenminimale Rundreise"}$$

"kostenminimale Rundreise!"

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 $\| Perm(1,..,n) \| = n!$

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!]

Abstraktion für "Suchraum durchlaufen":

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

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!]

Abstraktion im Kontext von Turing-Programmen:

bislang: deterministisch $\rightsquigarrow$ δ Funktion

nunmehr: nichtdeterministisch $\rightsquigarrow$ δ Relation, d.h.

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

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 Akzeptieren von Sprachen betrachten]

$L(NTM) := \{ w \mid$ Eingabewort $w \mapsto$ Ausgangssituation

$\vdash^*$ (für mindestens eine Rechnung)

Stopp-Situation $\mapsto$ Akzeptieren $\}$

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

$NP := \{ L \mid$ ex. NTM mit

(1) $L = L(NTM)$

(2) $NTIME(NTM)$ polynomiell beschränkt $\}$

Satz $CLIQUE_1$, $KNAPSACK_1$ und $TRAUSALE_1$ sind Elemente von NP.

Beweis: siehe obige Beobachtungen

Satz $P \subset NP$.

Beweis: gewiß Definitionen

(jedes deterministische Programm auch als formal "nichtdeterministisch" deutbar)

$$\mathbb{P} := \{ L \mid \text{ex. TM mit } \chi_L = \sigma(TM), \text{Time}(TM) \text{ polynomiell beschränkt} \}$$

$$\mathbb{NP} := \{ L \mid \text{ex. NTM mit } L = L(NTM), \text{NTIME}(NTM) \text{ polynomiell beschränkt} \}$$

Satz Für alle $L \in \mathbb{NP}$

ex. Polynom p, ex. (deterministische) TM

mit (1) $\chi_L = \sigma(TM)$

(2) $\text{TIME}(TM)(n) \leq 2^{p(n)}$

Beweis: sei $L \in \mathbb{NP}$

$\rightsquigarrow$ ex. NTM, ex. Polynom q mit $L = L(NTM)$

$$\text{NTIME}(NTM)(n) \leq q(n)$$

$\rightsquigarrow$ f. alle $w \in L$, ex. akzeptierende nichtdeterministische Rechnung mit

$$\#(\text{Schritte für } w) \leq q(\underbrace{|w|}_{\text{Länge von } w}) =: m$$

für jeden Schritt gilt: nichtdeterministische Auswahl

erfolgt aus höchstens

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

↖ Eigenschaft von NTM, unabhängig von w

vielen Möglichkeiten

$\rightsquigarrow$ ("zulässige") Rechnung von NTM der Länge m

$$\hat{=} \; (i_1, \ldots, i_m) \in (Q \times \Gamma \times \{R, L, N\})^m$$

mit: $i_t \in Q \times \Gamma \times \{R, L, N\}$

ist Tripel $(\underset{\downarrow}{q'}, \underset{\downarrow}{a'}, \underset{\downarrow}{d})$,

das in Schritt t, $t = 1, \ldots, m$,

bei dann gegebenem Zustand q

und gelesenem Zeichen a

gewählt werden kann, d.h.

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

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

"Konstruktion" der behaupteten TM:

diese simuliert alle möglichen "zulässigen" Rechnungen von NTM

Aufwand

Eingabe: w

Methode:

1) berechne $n := |w|$ — polynomiell

2) berechne $m := q(n)$ — polynomiell

3) simuliere nacheinander — k^m Durchläufe, jeweils polynomiell

jede "zulässige" Rechnung von NTM

4) falls eine (von NTM) akzeptierende Rechnung gefunden — polynomiell

dann akzeptiere

sonst verwerfe

$q(n)$ mit q poly

$\rightsquigarrow$ insgesamt: "poly + $k^{\sim m}$ · poly + poly"

$\leq 2^{p(n)}$ für geeignetes Polynom p

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:

$$w \in L_1 \quad \text{gdw} \quad f(w) \in L_2$$

$$\chi_{L_1} = \chi_{L_2} \circ f$$

für Entscheidungsverfahren:

wenn f total rekursiv und L_2 rekursiv,

dann L_1 rekursiv

[denn: Menge der total rekursiven Fkt abgeschlossen unter $\circ$]

für "effiziente" Entscheidungsverfahren:

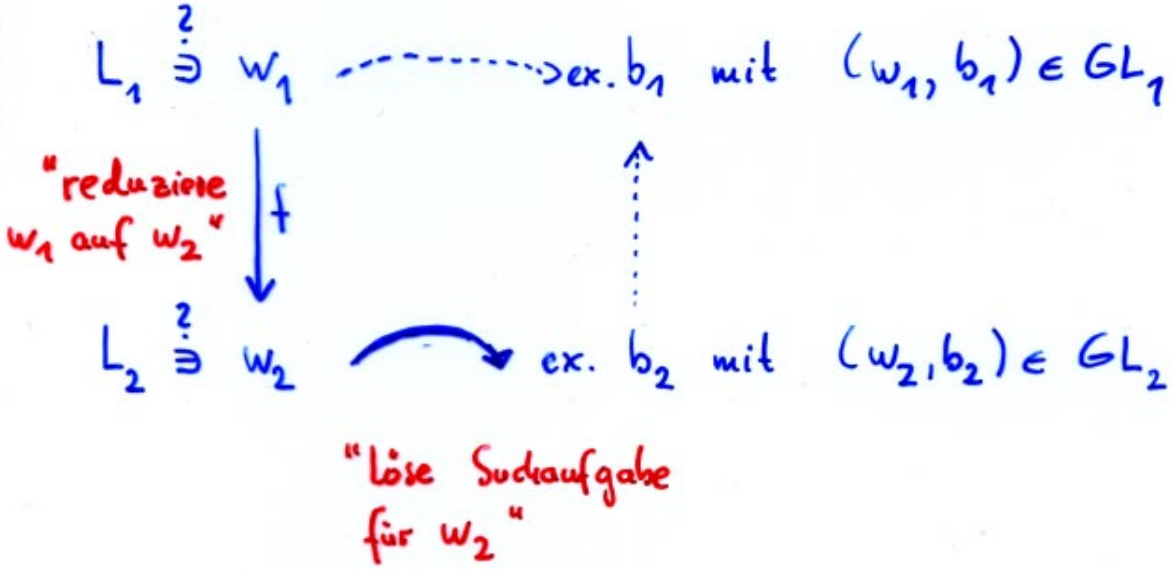
wenn f polynomiell (Zeit-) beschränkt und $L_2 \in \mathbb{P}$

dann $L_1 \in \mathbb{P}$

[denn: Menge der polynomiell (Zeit-) beschränkten Fkt. abgeschlossen unter $\circ$:

$\text{Time}(c(TM_2, TM_1))(w) = \text{Time}(TM_1)(w) + \text{Time}(TM_2)(\sigma(TM_1)(w))$]

für "Suchverfahren":



Definition $\leq_p$ definiere eine zweistellige Relation auf Sprachen wie folgt:

$L_1 \leq_p L_2$: gdw ex. polynomiell (Zeit-) beschränkte Funktion f mit

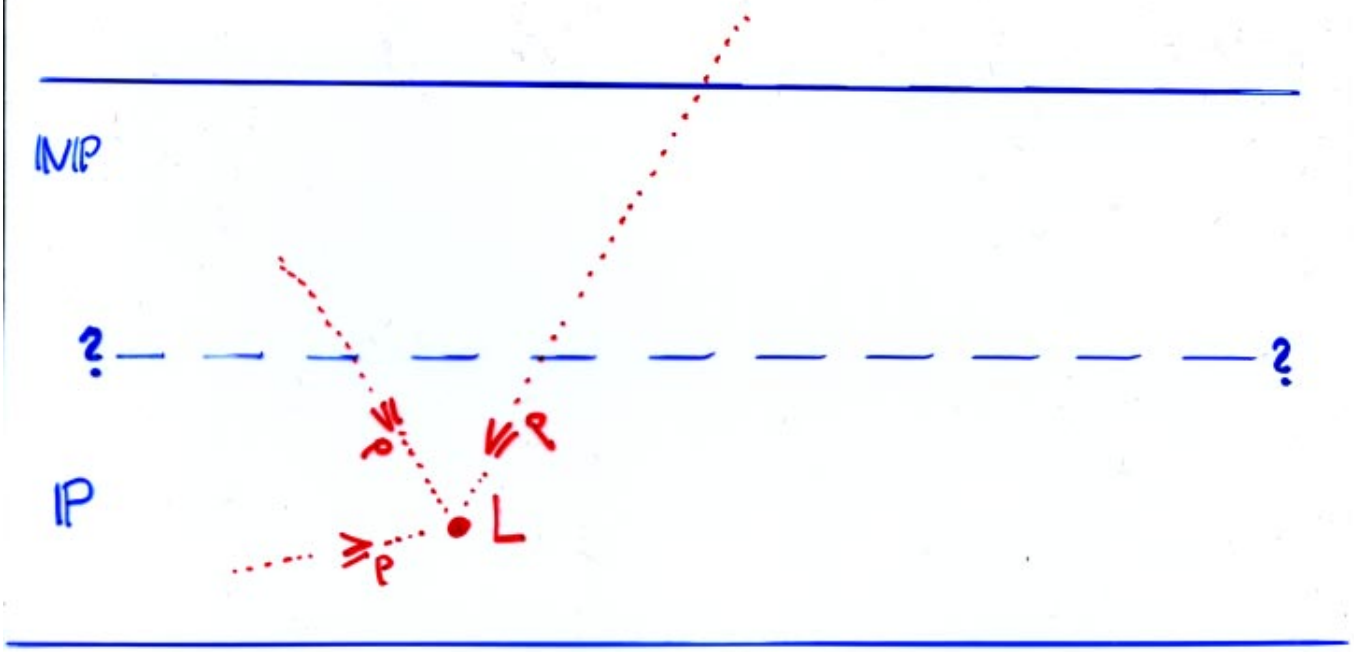
$w \in L_1$ gdw $f(w) \in L_2$

"L_1 im Wesentlichen nicht aufwändiger als L_2"

Satz: $\leq_p$ ist transitiv (und reflexiv)

Satz: $\mathbb{P}$ ist bezüglich $\leq_p$ "nach unten abgeschlossen"

Satz $\mathbb{NP}$ ist bezüglich $\leq_p$ "nach unten abgeschlossen"



$L \in \mathbb{P} \rightsquigarrow L \leq_p L_1$ f. fast alle L_1

$\mathbb{P}$ "unterste" Problemklasse bez. $\leq_p$

Definition

L heißt **NP-vollständig** : gdw

- $L \in NP$
- f. alle $L_1 \in NP$ gilt: $L_1 \leq_p L$

L heißt **NP-hart** : gdw

- f. alle $L_1 \in NP$ gilt: $L_1 \leq_p L$

Satz Sei L NP-vollständig. Dann gilt:

1. Falls $L \in P$, dann $P = NP$
2. Falls $L \notin P$, dann: f. alle NP-vollständigen L_1: $L_1 \notin P$

Beweis: (zu 1.): "$\subseteq$" $P \subseteq NP$ nach Definition

"$\supseteq$": $L_1 \in NP$ $\xrightarrow{L \text{ NP-vollständig}}$ $L_1 \leq_p L$

$\xrightarrow{L \in P}$ $L_1 \in P$

gibt es eine NP-vollständige Sprache ?

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

⋮

ja! – sogar "jede Menge"

– mit "Prototyp" SAT_1

Aussagenlogik (Schaltfunktionen)

Aussagenvariablen : $x_1, \dots, x_n, \dots$

Literale : $\dots, x_k, \overline{x}_k, \dots$ (Negation)

Klausel : Disjunktion von Literalen (von n Variablen)

Formel : Konjunktion von Klauseln (m viele)

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

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

Menge der erfüllbaren aussagenlogischen Formeln

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

Suchproblem : ... ex. $\mathcal{L}$?

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

Satz $SAT_1 \in NP$

Beweis: (1) rate nichtdeterministisch

eine erfüllende Belegung $\mathcal{L}$

(2) werte Formel Φ unter $\mathcal{L}$ aus

(3) falls Auswertung liefert 1

dann akzeptiere

sonst verwerfe

alle Schritte sind polynomiell (Zeit-) beschränkt

Satz SAT_1 ist NP-vollständig, d.h.

- $SAT_1 \in NP$: siehe oben
- f. alle $L_1 \in NP$: $L_1 \leq_p SAT_1$:
  - simuliere polynomiell beschränkte Rechnungen durch Formeln
  - reduziere "akzeptieren" auf "erfüllende Belegung finden"

Beh: f. alle $L \in NP$: $L \leq_p SAT_1$

ex. NTM, ex. Polynom p:

$w \in L$ gdw ex. nichtdet. akzeptierende Rechnung von NTM der Länge $\leq p(|w|)$

ex. polynomiell (Zeit-) beschränkte Funktion f mit

$w \in L$ gdw $f(w) \in SAT_1$

$w \longmapsto$ akzeptierende Rechnung $\longmapsto$ simulierende Formel Φ_w

Idee:

- beschreibe Arbeitsweise von NTM unter Eingabe von w

  durch Formel Φ_w der Aussagenlogik, so dass

  $w \in L$ gdw $\underbrace{\Phi_w \in SAT_1}_{\Phi_w \text{ erfüllbar}}$

- Konstruktion $w \overset{f}{\longmapsto} \Phi_w$

  ist polynomiell (Zeit-) beschränkt

sei $w \in \Sigma^*$ mit $|w| = n$ gegeben

$m := p(n)$

eine akzeptierende Rechnung muss berücksichtigen:

Zeitpunkte: $t = 1, \ldots, m$

Zellen: $z = -m, -(m-1), \ldots, -1, 0, 1, \ldots, m$

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 Kopf schaut
- im Zustand q auf diese Zelle
- im Quasizustand X^- nicht auf diese Zelle

für jede elementare Aussage: eine Aussagenvariable

insgesamt: $\underbrace{m \cdot (2m+1) \cdot \|\Gamma\| \cdot (\|Q\|+1)}_{\leq q(n) \text{ für geeignetes Polynom}}$ Aussagenvariablen

mit Hilfe der Aussagenvariablen $[z,t,a,q]$

durch geeignete Formeln beschreiben:

- Anfangssituation
- akzeptierende (Stopp-) Situation
- Situations-Übergänge
- Eindeutigkeit der Situationen

Beschreibung der Anfangssituation:

für $t=1$ steht auf Zellen $z=0,\dots,n-1$

das Wort $w = a_1 \dots a_n$; auf allen übrigen Zellen B;

Kopf schaut im Anfangszustand q_0 auf Zelle 0:

$$\alpha_{A_w} \equiv [-m,1,B,X] \wedge \dots \wedge [-1,1,B,X]$$
$$\wedge [0,1,a_1,q_0] \wedge [1,1,a_2,X] \wedge \dots \wedge [n-1,1,a_n,X]$$
$$\wedge [n,1,B,X] \wedge \dots \wedge [m,1,B,X]$$

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

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

für $t = m$ schaut der Kopf

im akzeptierenden Zustand q_F auf eine der Zellen:

$$\alpha_F \equiv \bigvee_{\substack{-m \le z \le m \\ a \in \Gamma}} [z, m, a, q_F]$$

Disjunktion über alle möglichen Worte von z und a

(polynomiell viele in n)

Beschreibung der Situations-Übergänge:

Fall 1: für $-m \le z \le m$,
$1 \le t \le m-1$,
$x \in \Gamma$,
X :

$[z,t,x,X]$

Kopf schaut nicht auf Zelle z

gilt zum Zeitpunkt $t+1$

Fall 1.1: Kopf schaut nicht auf Zelle z $[z,t+1,x,X]$

Fall 1.2: Kopf schaut auf Zelle z, im Folgezustand q' $[z,t+1,x,q']$

Fall 1.2.1: vermöge Linksschritt von Zelle $z+1$, mit q, a gemäß δ $[z+1,t,a,q]$ $\vee\ldots$

Fall 1.2.2: vermöge Rechtsschritt von Zelle $z-1$, mit q, a gemäß δ $[z-1,t,a,q]$ $\vee\ldots$

also:

$$\overline{[z,t,x,X]} \vee \Big([z,t+1,x,X] \vee \big(([z,t+1,x,q'] \wedge [z+1,t,a,q] \vee \ldots) \vee \text{Fall 1.2.1}$$
$$([z,t+1,x,q'] \wedge [z-1,t,a,q] \vee \ldots) \big) \text{ Fall 1.2.2} \Big)$$

wenn "Fall 1" — dann — "Fall 1.1" — oder

Fall 2: für $-m \leq z \leq m$,
$1 \leq t \leq m-1$,
$a \in \Gamma$
$q \in Q$

$[z,t,a,q]$

Kopf schaut auf Zelle z

gilt zum Zeitpunkt $t+1$

Fall 2.1 Kopf schaut auf Zelle z, mit q', a' gemäß δ

$[z,t+1,a',q']$
$\vee \ldots$

Fall 2.2 Kopf schaut auf Zelle $z-1$ oder $z+1$ mit a' gemäß δ

$[z,t+1,a',X]$
$\vee \ldots$

also:

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

$$\alpha_{\ddot{u}} \equiv \bigwedge \text{ „alle solchen Formeln“}$$

Beschreibung der Eindeutigkeit der Situationen:

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

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

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

$$\bigvee_{\substack{-m \le z \le m \\ x \in \Gamma \\ q \in Q}} [z,t,x,q] \qquad \text{"mindestens"}$$

"wenn Kopf auf Zelle z kommt, so nicht auf $z+1$, $z+2$: höchstens"

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

$\alpha_E \equiv \bigwedge$ "alle solchen Formeln"

(abgesehen von Ungenauigkeiten, ...)

kann man nachprüfen:

$w \in L = L(NTM)$ gdw

$\Phi_w := \alpha_{A_w} \wedge \alpha_F \wedge \alpha_{\ddot{u}} \wedge \alpha_E \in SAT_1$

- α_{A_w}: Anfangssituation für w
- α_F: akzeptierende Stopp-Situation
- $\alpha_{\ddot{u}}$: Situations-Übergänge
- α_E: Eindeutigkeit
- $\in SAT_1$: erfüllbar

ggf. noch in "Normalform" bringen

dann (immer noch):

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

bislang: für $\Phi \equiv \bigwedge_j \bigvee_i (\overline{x_{i,j}})$

beliebig lange Disjunktionen erlaubt

↷ jeweils "viel Auswahl" für erfüllendes Literal

nunmehr: Auswahl einschränken:

nur Disjunktionen mit Länge $\leq$

3 : ?

2 : ?

1 : offensichtlich polynomiell (Zeit-) beschränkt entscheidbar

[keine komplementären Literale wie z.B. $\ldots \wedge x_i \wedge \ldots \wedge \overline{x_i}$]

Satz 3-SAT_1 ist NP-vollständig

Satz $2\text{-SAT}_1 \in \mathrm{P}$

Beh: jede Klausel $\alpha = \ell_1 \vee \dots \vee \ell_k$ mit $k \geq 4$

lässt sich "ersetzen" mit

Hilfe "neuer" Aussagenvariablen $y_1, \dots, y_{k-3}$ durch

$$\begin{aligned}
\beta \equiv\ & (\ell_1 \vee \ell_2 \vee y_1) \\
& \wedge (\ell_3 \vee \bar{y}_1 \vee y_2) \\
& \wedge (\ell_4 \vee \bar{y}_2 \vee y_3) \\
& \vdots \\
& \wedge (\ell_{k-2} \vee \bar{y}_{k-4} \vee y_{k-3}) \\
& \wedge (\ell_{k-1} \vee \bar{y}_{k-3} \vee \ell_k)
\end{aligned}$$

derart, dass f. alle (Variablen-) Belegungen $\mathfrak{L}$ gilt:

$\mathfrak{L}(\alpha) = 1$ gdw ex. Erweiterung $\tilde{\mathfrak{L}}$ von $\mathfrak{L}$ mit $\tilde{\mathfrak{L}}(\beta) = 1$

"$\Rightarrow$": Sei $\mathcal{L}(\alpha) = 1$

$\rightsquigarrow$ ex i mit $\mathcal{L}(\ell_i) = 1$

α Disjunktion

definiere $\tilde{\mathcal{L}}(\gamma_j) = \begin{cases} 1 & j = 1, \dots, i-2 \\ 0 & j = i-1, \dots, k-3 \end{cases}$

"$\Leftarrow$": Sei $\tilde{\mathcal{L}}$ Erweiterung von $\mathcal{L}$ mit $\tilde{\mathcal{L}}(\beta) = 1$

zu zeigen: ex. i mit $\mathcal{L}(\ell_i) = 1$

Bew: (indirekt):

angenommen: f. alle i: $\mathcal{L}(\ell_i) = 0$

$\rightsquigarrow$ $\tilde{\mathcal{L}}(\gamma_1) = 1$ [sonst $\ell_1 \vee \ell_2 \vee \gamma_1$ nicht erfüllt]

$\rightsquigarrow$ $\tilde{\mathcal{L}}(\gamma_2) = 1$ [sonst $\ell_3 \vee \overline{\gamma}_1 \vee \gamma_2$ nicht erfüllt]

$\vdots$ (Induktion)

$\rightsquigarrow$ $\tilde{\mathcal{L}}(\gamma_{k-3}) = 1$ [sonst ...]

$\rightsquigarrow$ $\tilde{\mathcal{L}}(\ell_{k-1} \vee \overline{\gamma}_{k-3} \vee \ell_k) = 0$

$\rightsquigarrow$ $\tilde{\mathcal{L}}(\beta) = 0$ ↯

β Konjunktion

$$(x_0 \vee x_3) \wedge (x_1 \vee \bar{x}_2) \wedge (\bar{x}_1 \vee x_2) \wedge (\bar{x}_1 \vee \bar{x}_2 \vee \bar{x}_3) \wedge x_3$$

↖ erfüllt dann Klausel

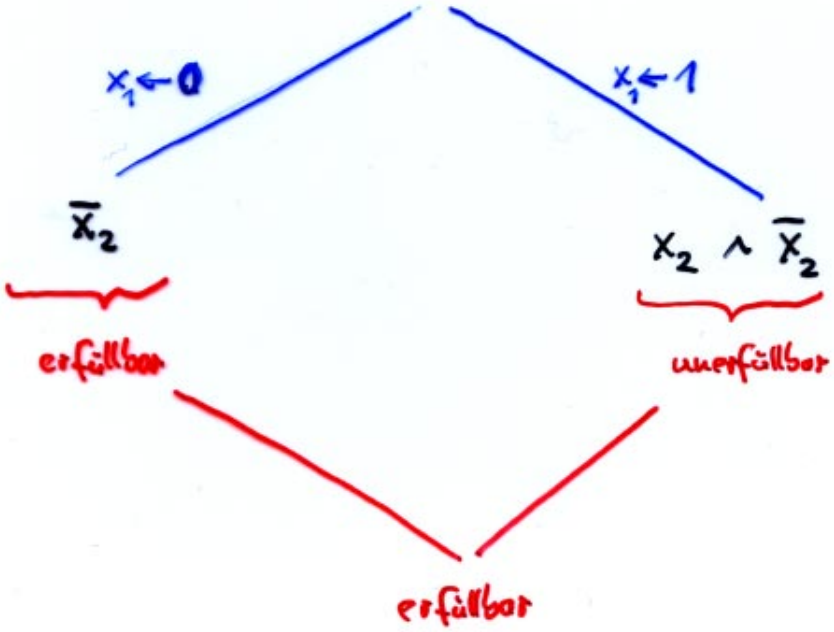
entferne Einerklausen:

↗ wird dann mit 0 belegt (kein Beitrag zum Erfüllen)

↑ muss mit 1 belegt werden

$$(x_1 \vee \bar{x}_2) \wedge (\bar{x}_1 \vee x_2) \wedge (\bar{x}_1 \vee \bar{x}_2)$$

splitte, d.h. Fallunterscheidung für z.B. x_1, mit rekursiver Anwendung:



erfüllende Belegung $\mathcal{L}$:

x_3	x_1	x_2
1	0	0

Entscheidungsverfahren für SAT_1 (Erfüllbarkeit)

Eingabe: $\alpha = \alpha_1 \wedge \ldots \wedge \alpha_k$

α_i Disjunktion von Literalen

Ausgabe: $\begin{cases} 0 \;\hat{=}\; \text{unerfüllbar} \\ 1 \;\hat{=}\; \text{erfüllbar} \end{cases}$

Methode:

- streiche mehrfach vorkommende Klauseln (bis auf ein Vorkommen)
- entferne mehrfache Vorkommen einer Variablen in einer Klausel:
  - falls x und $\bar{x}$ in α_i vorkommen
    dann streiche α_i
  - falls x, bzw. $\bar{x}$, in α_i mehrfach vorkommt
    dann streiche alle Vorkommen bis auf eines

falls alle Klauseln gestrichen, dann Ausgabe 1

- entferne Einerklauseln:
  - falls $\alpha_i \equiv x$, bzw. $\alpha_i \equiv \bar{x}$
    dann streiche alle Klauseln, die x, bzw. $\bar{x}$, enthalten
    streiche alle Literale $\bar{x}$, bzw. x, in Nicht-Einerklauseln

falls Einerklausel $\bar{x}$, bzw. x erzeugt, dann Ausgabe 0

- **falls** jede Variable entweder nur negiert
  oder nur unnegiert vorkommt

  dann Ausgabe 1

  sonst wähle x, so dass x sowohl negiert als auch unnegiert vorkommt,

  etwa (o.B.d.A) so:

$$\alpha_1 = (x_1 \vee \beta_1)$$
$$\vdots \qquad\qquad \vdots$$
$$\alpha_m = (x_1 \vee \beta_m)$$

$$\alpha_{m+1} = (\bar{x}_1 \vee \gamma_1)$$
$$\vdots \qquad\qquad \vdots$$
$$\alpha_{m+n} = (\bar{x}_1 \vee \gamma_n)$$

δ = Konjunktion der x-freien Klauseln

$[\alpha_1 \wedge \ldots \wedge \alpha_m \wedge \alpha_{m+1} \wedge \ldots \wedge \alpha_{m+n} \wedge \delta$ erfüllbar **gdw**
$(\gamma_1 \wedge \ldots \wedge \gamma_n \wedge \delta$ erfüllbar **oder** $\beta_1 \wedge \ldots \wedge \beta_m \wedge \delta$ erfüllbar$)]$

- **falls** ein γ_i, β_j mehr als eine Variable enthält
  dann wende Verfahren rekursiv an auf:

(i) $\alpha_{x_1} := \gamma_1 \wedge \ldots \wedge \gamma_n \wedge \delta$

und $\alpha_{\bar{x}_1} := \beta_1 \wedge \ldots \wedge \beta_m \wedge \delta$

(ii) berechne Maximum der Ergebnisse

- falls jedes γ_i, β_j genau eine Variable enthält

  dann wende Verfahren rekursiv an auf

$$\tilde{\alpha} :\equiv (\beta_1 \vee \gamma_1) \wedge \ldots \wedge (\beta_1 \vee \gamma_n) \wedge \ldots \wedge (\beta_m \vee \gamma_n) \wedge \delta$$

[für diesen Fall:

($\gamma_1 \wedge \ldots \wedge \gamma_n \wedge \delta$ erfüllbar oder $\beta_1 \wedge \ldots \wedge \beta_m \wedge \delta$ erfüllbar) gdw

$\tilde{\alpha}$ erfüllbar]

allgemein gelten folgende Invarianten:

- rekursive Anwendungen "erhalten Erfüllbarkeit"
- Streichungen "erhalten Erfüllbarkeit"

ferner: Ausgaben "korrekt" (bezüglich dann vorliegender Formel)

Satz

- Das Entscheidungsverfahren für SAT_1 ist "korrekt und vollständig", d.h.

  $\alpha \in SAT_1$ gdw Ausgabe = 1

  "$\leftarrow$ korrekt"

  "$\rightarrow$ vollständig"

- Der Zeitaufwand ist abschätzbar als

  $Time(\ldots)(n) = \mathcal{O}(2^n)$ mit $n = \#(\text{Variablen})$

  d.h. keine positive Lsg. für $\mathbb{P} \stackrel{?}{=} \mathbb{NP}$!

- Der Zeitaufwand für Eingaben mit Disjunktion der Länge ≤ 2 ist polynomiell beschränkt.

  d.h. $2\text{-}SAT_1 \in \mathbb{P}$!

Beweis

- "korrekt und vollständig" : Invarianten, korrekte Ausgaben

- i. Allg. exponentiell:

  grobe Abschätzung: - Streichungen, Tests, etc. : polynomiell

  - bei jeder rekursiven Anwendung eine Variable weniger

  #(Variablen) — 2 Aufrufe — jeweils eine Variable weniger

  $\rightsquigarrow$ $Time(..)(n) = 2 \cdot Time(..)(n-1) + P(n)$

  $\rightsquigarrow$ $Time(..)(n) = O(2^n)$

  $[\, 2^n = 2 \cdot 2^{n-1} \,]$

- speziell, falls alle Disjunktionen Länge ≤ 2 haben:

  - Streichungen, Tests, etc. : polynomiell

  - bei jeder rekursiven Anwendung eine Variable weniger, aber stets nur ein rekursiver Aufruf mit 2 polynomiell abschätzbar in α

  $\rightsquigarrow$ Laufzeit polynomiell beschränkt

- es gibt viele NP-vollständige Probleme (Sprachen)
- jeweils vermöge Reduktion nachweisen:

Satz Sei L NP-vollständig (z.B. 3-SAT$_1$).

Falls $\tilde{L} \in NP$ und $L \leq_p \tilde{L}$,

dann $\tilde{L}$ NP-vollständig.

Beweis:

noch zu zeigen: f. alle $L' \in NP$: $L' \leq_p \tilde{L}$

Bew: $L' \in NP \rightsquigarrow L' \leq_p L$ (L NP-vollständig)

$\rightsquigarrow L' \leq_p \tilde{L}$ ($\leq_p$ transitiv)

Satz $CLIQUE_1$ ist NP-vollständig.

zu zeigen: (1) $CLIQUE_1 \in NP$

(z.B.) (2) $3\text{-}SAT_1 \leq_p CLIQUE_1$

ad (1): vermöge nichtdeterministischem Suchprogramm

ad (2): gesucht: polynomiell (Zeit-) beschränkte Reduktionsfunktion f mit

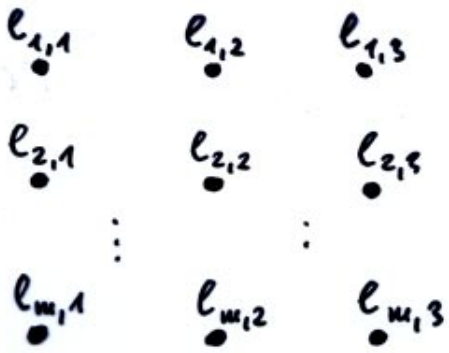
$$\underbrace{\bigwedge_j \bigvee_{i=1,2,3} x_{j,i}^{(-)} \in 3\text{-}SAT_1}_{\text{erfüllbar}} \quad \text{gdw} \quad \underbrace{f\left(\bigwedge_j \bigvee_{i=1,2,3} x_{j,i}^{(-)}\right) = ((V,E),k) \in CLIQUE_1}_{\text{enthält Clique } V' \text{ der Größe } k}$$

gefunden: für $\alpha \equiv \bigwedge_{j=1}^{m} c_j \equiv \bigwedge_{j=1}^{m} \bigvee_{i=1,2,3} l_{j,i}$

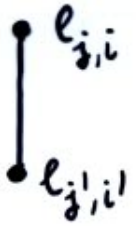
mit $l_{j,i} \in \{x_1, \bar{x}_1, \ldots, x_n, \bar{x}_n\}$

sei $f(\alpha)$ definiert durch:

Knotenmenge V:



Kantenmenge E:



:gdw $j \neq j'$ [verschiedene Klauseln]

und

$\ell_{j,i}$ und $\ell_{j',i'}$ nicht komplementär [gleichzeitig erfüllbar]

Cliquegröße k: m [#(Klauseln)]

Beh 1: $\mathcal{L}(\alpha) = 1$ $\overset{!}{\rightsquigarrow}$ (V, E) hat Clique der Größe m

↳ α Konjunktion
↳ f.alle $j = 1, \dots, m$: $\mathcal{L}(c_j) = 1$

↳ c_j Disjunktion
↳ f.alle $j = 1, \dots, m$ ex. $i_j \in \{1,2,3\}$: $\mathcal{L}(\ell_{j,i_j}) = 1$

↳ gemäß Definition
↳ $V' = \{\ell_{1,i_1}, \dots, \ell_{m,i_m}\}$ ist Clique in (V,E) der Größe m

denn:

- aus <u>verschiedenen</u> Klauseln, nach Konstruktion
- <u>nicht komplementär</u>, weil $\mathcal{L}(\ell_{j,i_j}) = 1$

Beh. 2: (V, E) hat Clique der Größe m $\overset{!}{\rightsquigarrow}$ ex. $\mathcal{L}$ mit $\mathcal{L}(\alpha) = 1$

↳ etwa $V' = \{ \ell_{j_1, i_1}, \dots, \ell_{j_m, i_m} \}$

↳ Def. E
- $\{ j_1, \dots, j_m \} = \{ 1, \dots, m \}$
- keine komplementären Literale

↳ definiere $\mathcal{L}$ derart, dass $\mathcal{L}(\ell_{j_k, i_k}) = 1$

↳ $\mathcal{L}(\alpha) = 1$

Beispiel:

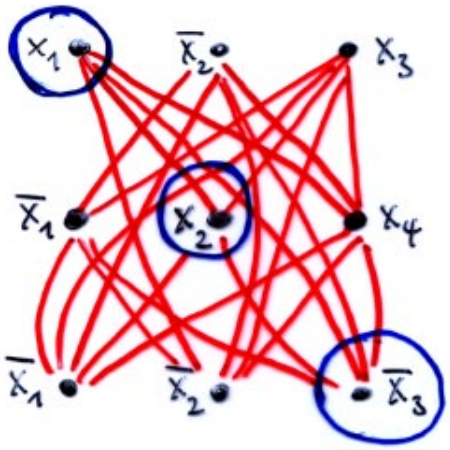
$$\alpha \equiv c_1 \wedge c_2 \wedge c_3$$

$$\equiv (x_1 \vee \bar{x}_2 \vee x_3) \wedge (\bar{x}_1 \vee x_2 \vee \bar{x}_4) \wedge (\bar{x}_1 \vee \bar{x}_2 \vee \bar{x}_3)$$

$$1 \quad 0 \quad 0 \qquad 0 \quad 1 \quad 1 \qquad 0 \quad 0 \quad 1$$

↘ für $\mathfrak{L}$ mit Variable: x_1 x_2 x_3 x_4 gilt $\mathfrak{L}(\alpha)=1$

Belegung: 1 1 0 0



Satz $KNAPSACK_1$ ist NP-vollständig.

zu zeigen: (1) $KNAPSACK_1 \in NP$

(z.B.) (2) $3\text{-}SAT_1 \leq_p KNAPSACK_1$

ad (1): vermöge nichtdeterministischem Suchprogramm

ad (2): gesucht:

polynomiell (Zeit-) beschränkte Reduktionsfunktion f

mit

$\underbrace{\bigwedge_j \bigvee_{i=1,2,3}^{(-)} x_{j,i} \in 3\text{-}SAT_1}_{\text{erfüllbar}}$ gdw $f\left(\bigwedge_j \bigvee_{i=1,2,3}^{(\neg)} x_{j,i}\right) =$

$\underbrace{\left(\begin{array}{ll} g_1,\dots,g_k, & \text{Gewichte} \\ \tilde{a}_1,\dots,\tilde{a}_k, & \text{Nutzen} \\ G, & \text{Gewichtsschranke} \\ A \;\;) & \text{geforderter N.} \end{array}\right.}_{\in KNAPSACK_1}$

ex. $I \subseteq \{1,\dots,k\}$ mit $\sum_{i\in I} g_i \leq G$ und $\sum_{i\in I} \tilde{a}_i \geq A$

gefunden: für $\alpha \equiv \bigwedge_{j=1}^{m} c_j \equiv \bigwedge_{j=1}^{m} \bigvee_{i=1,2,3} \ell_{j,i}$ mit

$$\ell_{j,i} \in \{ x_1, \bar{x}_1, \ldots, x_n, \bar{x}_n \}$$

sei $f(\alpha)$ definiert durch:

Objekte: $k = 2n + 2m$, d.h.

$1 \ldots n$	$n+1 \ldots 2n$	$2n+1 \ldots 2n+m$	$2n+m+1 \ldots 2n+2m$
$a_1 \ldots a_n$	$b_1 \ldots b_n$	$c_1 \ldots c_m$	$d_1 \ldots d_m$

Nutzen = Gewichte: $\in \mathbb{N}$ jeweils als $(m+n)$-stellige Dezimalzahl

"geeignet wählen" (s.u.)

N = G: jeweils als $(m+n)$-stellige Dezimalzahl,

und zwar:

$$\underbrace{4 \ldots 4}_{m} \; \underbrace{1 \ldots 1}_{n\text{-mal}}$$

Forderung gemäß spezieller Wahl:

$$\sum_i a_i + \sum_i b_i + \sum_j c_j + \sum_j d_j = \underbrace{4 \ldots 4\, 1 \ldots 1}_{\text{als Dezimalzahl}}$$

Zunächst am Beispiel:

$$\alpha \equiv c_1 \wedge c_2 \wedge c_3$$

$$\equiv (x_1 \vee \bar{x}_2 \vee x_3) \wedge (\bar{x}_1 \vee x_2 \vee \bar{x}_4) \wedge (\bar{x}_1 \vee \bar{x}_2 \vee \bar{x}_3)$$

$m = 3$ (Klauseln)
$n = 4$ (Variablen)

	c_1	c_2	c_3	x_1	x_2	x_3	x_4	
A =	4	4	4	1	1	1	1	
	1	0	0	1	0	0	0	a_1 beschreibt x_1
	0	1	0	0	1	0	0	a_2 beschreibt x_2
	1	0	0	0	0	1	0	a_3 beschreibt x_3
	0	0	0	0	0	0	1	a_4 beschreibt x_4
	0	1	1	1	0	0	0	b_1 beschreibt $\bar{x}_1$
	1	0	1	0	1	0	0	b_2 beschreibt $\bar{x}_2$
	0	0	1	0	0	1	0	b_3 beschreibt $\bar{x}_3$
jeweils:	0	1	0	0	0	0	1	b_4 beschreibt $\bar{x}_4$
$\sum_{\text{ausgewählt}} \in \{0,1,2,3\}$								
	1	0	0	0	0	0	0	c_1
	0	1	0	0	0	0	0	c_2
	0	0	1	0	0	0	0	c_3
	2	0	0	0	0	0	0	d_1
	0	2	0	0	0	0	0	d_2
	0	0	2	0	0	0	0	d_3

um Summe 1 zu erreichen: von jedem Paar a_i, b_i genau eines auswählen; $\hat{=}$ $\mathfrak{b}(x_i) = 1$ oder $\mathfrak{b}(\bar{x}_i) = 1$

Fall 1: $\sum_{\text{ausgewählt}} = 0$ $\rightsquigarrow c_i$ und 4 nicht erfüllt / nicht erreichbar

Fall 2: $\sum_{\text{ausgewählt}} \in \{1,2,3\}$ $\rightsquigarrow c_i$ erfüllt und 4 erreichbar

Endliche Automaten – Vorüberlegungen

① zur Erinnerung:
Klassifikation von "Problemen" (Sprachen, Funktionen):

⋮
nicht rekursiv aufzählbar z.B. $\overline{H_{\text{alleswege}}}$
rekursiv aufzählbar z.B. $H_{\text{alleswege}}$
rekursiv: ⋮
"eher nicht effizient" / exponentiell z.B. NP
⋮ ? UH ?
"eher effizient" / polynomiell: z.B. P
⋮
kubische Zeit
quadratische Zeit
lineare Zeit: ⋮
⋮

mit endlichem Gedächtnis,
zeichenweise lesen,
nicht schreiben

endliche Automaten

② zur Erinnerung:

Turing-Programme mit Übergangsfunktion

Zustand (endliches Gedächtnis) — Lesen — Folge-Zustand — Schreiben — "Adressieren"

$$\delta : Q \times \Gamma \longrightarrow Q \times \Gamma \times \{L, R, N\}$$

Spezialfall endlicher Automat:

nicht schreiben

nicht adressieren: Zeichenweise (von links nach rechts) lesen

Definition • Ein (deterministischer) endlicher Automat ist gegeben durch

$$DFA = (Q, \Gamma, \delta : Q \times \Gamma \to Q, q_0 \in Q, F \subset Q)$$

Q: endliche Zustandsmenge; Γ: Alphabet; δ: Übergangsfunktion; q_0: Startzustand; F: akzeptierende Zustände

- Fortsetzung von δ auf Γ^*:

$$\tilde{\delta} : Q \times \Gamma^* \to Q \quad \text{mit} \quad \tilde{\delta}(q, \varepsilon) := q$$

$$\tilde{\delta}(q, aw') := \tilde{\delta}(\underbrace{\delta(q,a)}_{\text{Folgezustand}}, w')$$

a: erstes Zeichen; w': Restwort

- $L(DFA) := \{w \mid w \in \Gamma^* \text{ und } \tilde{\delta}(q_0, w) \in F\}$

Variante mit externer Ausgabe: Mealy-Automat

↕ wird nicht wieder gelesen

- DFA mit
- zusätzlicher Ausgabefunktion $\gamma : Q \times \Gamma \to \Omega$

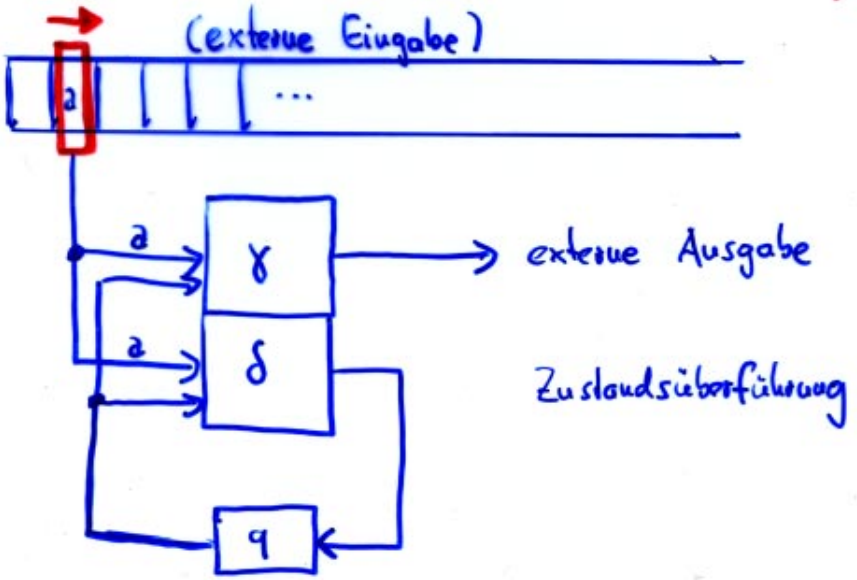
Q: Zustände, Γ: (Eingabe-) Alphabet, Ω: (Ausgabe-) Alphabet

$\tilde{\gamma} : Q \times \Gamma^* \to \Omega^*$ mit

$\tilde{\gamma}(q, \varepsilon) := \varepsilon$

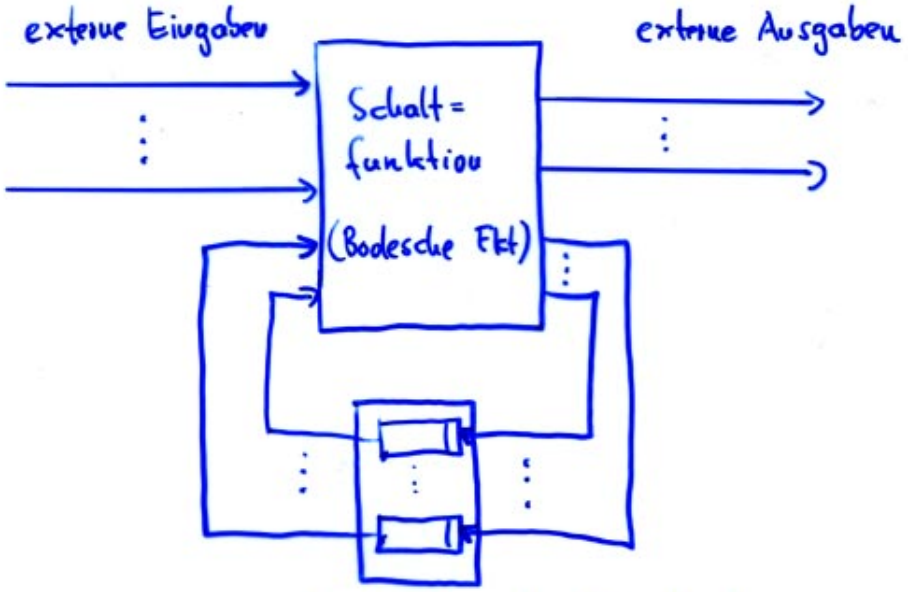
$\tilde{\gamma}(q, aw') := \gamma(q,a) \cdot \tilde{\gamma}(\delta(q,a), w')$

$\gamma(q,a)$: Ausgabe für erstes (Eingabe-) Zeichen; $\cdot$: Konkatenation; $\delta(q,a)$: Folgezustand; w': Restwort; $\tilde{\gamma}(\delta(q,a), w')$: Ausgabe für Restwort



③ zur Erinnerung:

Schaltwerke



Speicherelemente (Flip-Flops)

jeweils binäre Darstellungen von

Zuständen durch Speicherinhalte als 0-1-Folgen

Eingaben durch 0-1-Folgen

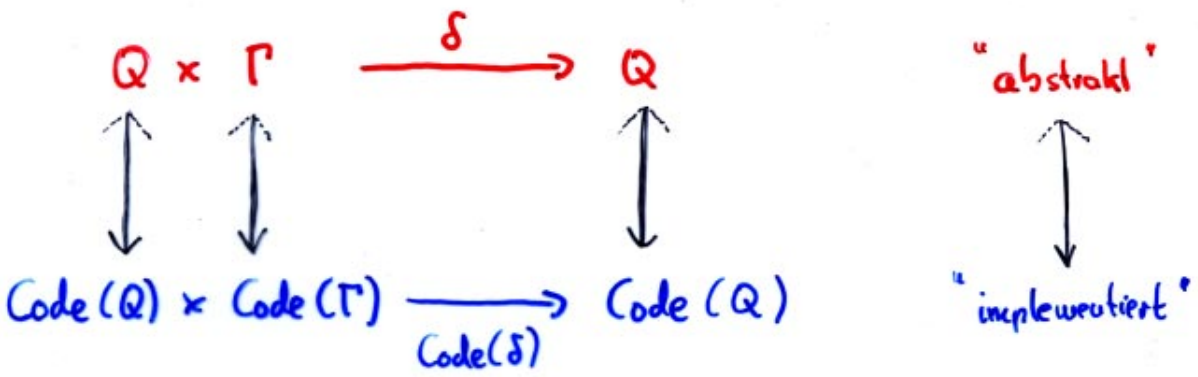
Ausgaben durch 0-1-Folgen

Zustandsüberführung / Ausgabefunktion durch Schaltfunktion

← Abstraktion (von binären Darstellungen)

→ Implementierung (mit binären Darstellungen)

z.B.



④ zur Erinnerung:
(sequentielle) Programmierung

(vereinfacht) wird die Vorgeschichte einer Rechnung

repräsentiert durch:

- momentan bearbeitete Programmstelle
  (vermöge Programmzähler entsprechend "Kontrollfluss")

  ≙ Zustand eines TM / DFA

- momentane Belegung der (Programm-) Variablen
  (gemäß "noch wirksamer" Wertzuweisungen)

  ≙ Bandinhalt eines TM / für DFA nicht möglich!

ⓘ (viele weitere Anwendungen / Deutungen)

endliche Automaten – Beispiele für Entscheidungsverfahren für Sprachen

DFA:

$\delta: Q \times \Gamma \to Q$
$q_0 \in Q$
$F \subset Q$

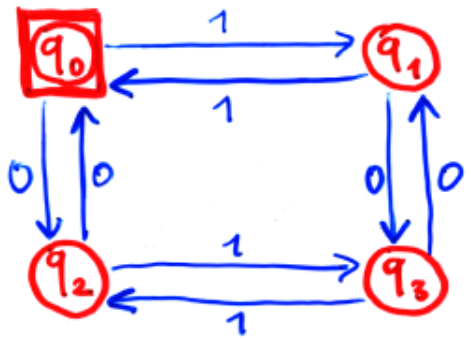
$$L(\text{DFA}) = \{ w \mid w \in \Gamma^* \text{ und } \tilde{\delta}(q_0, w) \in F \}$$

$$L_1 := \{ w \mid w \in \{0,1\}^*, \underbrace{\#(1\text{'en in } w) \text{ gerade}}_{\equiv 1}, \underbrace{\#(0\text{'en in } w) \text{ gerade}}_{\equiv 0} \}$$

vier Zustände für vier Fälle:

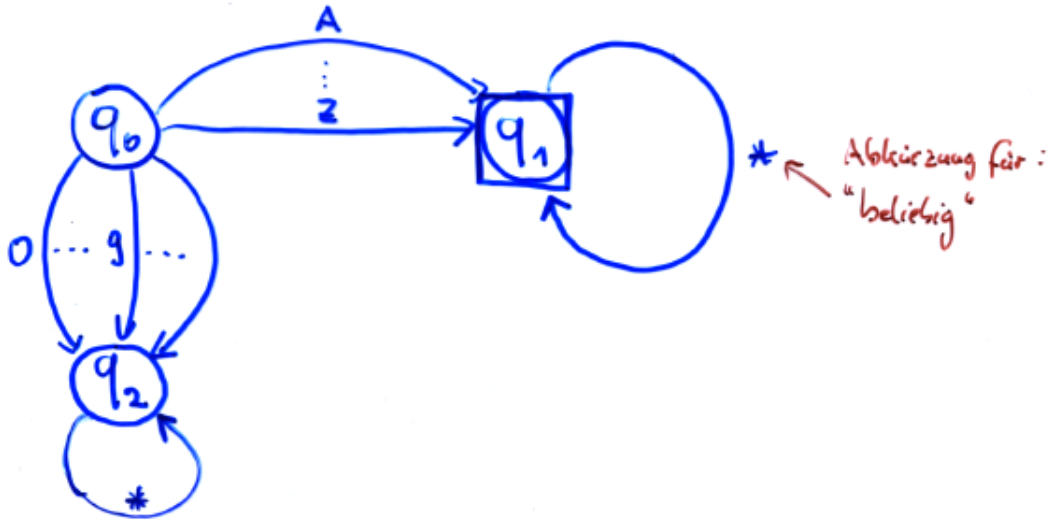
		$\equiv 0$	$\equiv 1$	Zustandsübergang 0	1
Anfangszustand → akz. Zustand	q_0	gerade	gerade	q_2	q_1
	q_1	gerade	ungerade	q_3	q_0
	q_2	ungerade	gerade	q_0	q_3
	q_3	ungerade	ungerade	q_1	q_2

graphische Darstellung:

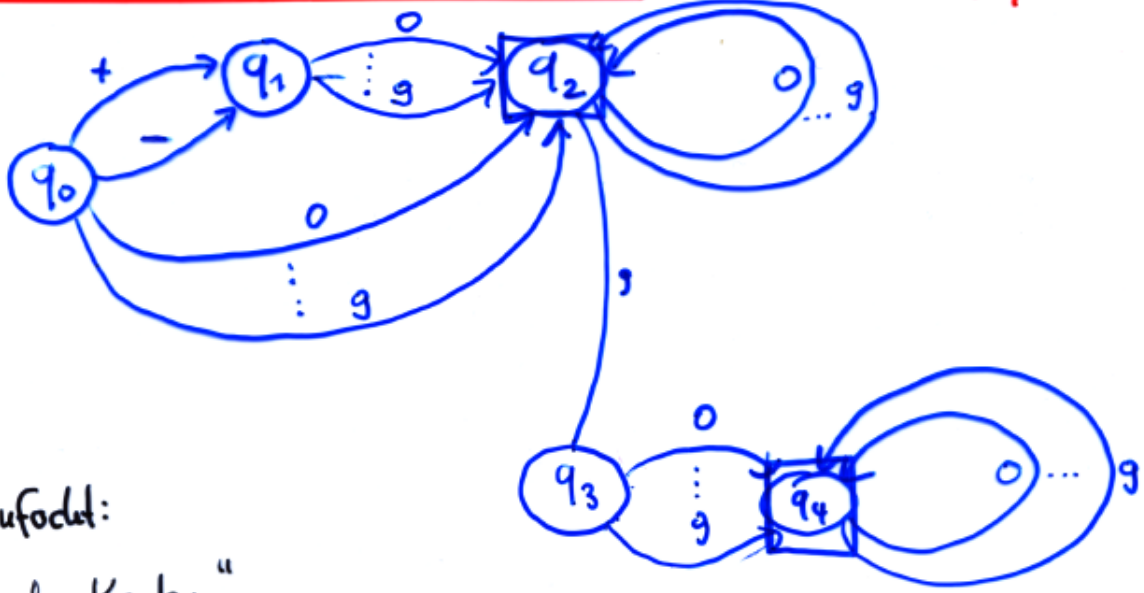


Sprache der Bezeichner (identifier), z.B.:

$$L_2 = \{ b_1 z_2 \dots z_n \mid b_1 \in \{A, B, \dots, Z\}, \text{ für } i = 2, \dots, n \in \mathbb{N}: z_i \in \{A, B, \dots, Z, 0, 1, \dots, 9, -, \_, \dots\} \}$$



Sprache der "Komma-Zahlen" (real_constant), z.B.:



vereinfacht:

"fehlende Kanten"

als Übergang in zusätzlichen Zustand q_5 ergänzen!

Sprache für "vollen Rucksack" A:

$\Gamma = \{1, \dots, A\}$, für festes $A \in \mathbb{N}$

$L_4 = \{ w_1 \dots w_n \mid n \in \mathbb{N}, w_i \in \Gamma,$
$\text{ex. } I \subset \{1,\dots,n\} : \sum_{i \in I} w_i = A \}$

$Q = \{ \{0\} \cup X \mid X \subset \Gamma \}$, $\|Q\| = 2^A$

$q = \{0, s_1, \dots, s_k\} \in Q$ soll bedeuten:

"aus der bislang gelesenen Eingabe kann man durch Auswahlen die Summen $0, s_1, \dots, s_k$ bilden"

$\delta(\overset{\in Q}{q}, \overset{\in \Gamma}{a}) := q \cup \{ s+a \mid s \in q \text{ und } s+a \le A \}$

schon vorher bildbar — zusätzlich bildbar durch Auswahl von a, sofern A nicht übertroffen

$q_0 := \{0\}$

$F := \{ q \mid q \in Q \text{ und } A \in q \}$

Gegenbeispiel:

$L = \{ 0^n 1^n \mid n \geq 1 \}$ nicht durch endlichen Automaten entscheidbar

(anschauliche) Begründung:

angenommen $DFA = (Q, \{0,1\}, \delta, q_0, F)$

entscheidet L

Q endlich

↝ ex $k \in \mathbb{N}$: $\| Q \| = k$

↝ wenn DFA einen Anfang der Form $\underbrace{0 \ldots 0}_{k+1}$ einliest

dann tritt in der entsprechenden Zustandsfolge

$$q_0 \xrightarrow{0} q_1 \xrightarrow{0} q_2 \rightarrow \ldots \xrightarrow{0} q_k \xrightarrow{0} q_{k+1}$$

ein Zustand doppelt auf, etwa

$q_i = q_j$ mit $i < j$

↝ DFA kann $\underbrace{0 \ldots 0}_{i}$ und $\underbrace{0 \ldots 0 \ldots 0}_{j}$ "nicht unterscheiden"

↝ ↯

dynamische Komplexität
Ausführungs-Komplexität

Aufwand der Ausführung für eine Eingabe

auf Semantik bezogen

statische Komplexität
Beschreibungs-Komplexität

Aufwand der Beschreibung (unabhängig von Eingabe)

auf Syntax bezogen

für Turing-Programme, z.B.:

Time (TM)
Space (TM)
⋮
γ (TM) gemäß Blum'scher Axiome

$\| Q \|$ Zustandsgröße

≙ Speicherbedarf für Programme

für μ-rekursive Funktionen, z.B.:

Laufzeit bei Simulation mit
$\mathcal{L}$: Assignment, StatementSequence / Expression
$\mathfrak{BR}$: Bounded Loop
Miu_a : Repeat/While Statement

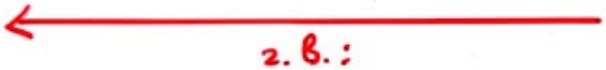
Schachtelungstiefe der Funktionale entsprechend syntaktischem Aufbau

Zusammenhänge: "wünschenswert", aber i. Allg. nur in "Glücksfällen"

dynamische Komplexität — statische Komplexität

z.B.:

"Ackermann-Funktion wächst schneller als jede primitiv-rekursive Funktion"



z.B.:

Schachtelungstiefe von PR ↝ Schachtelungstiefe von BoundedLoops

↝ pro Schachtelung: "(Aufwand eines Durchlaufs) · #(Durchläufe)"

dynamische Komplexität

statische Komplexität

bei endlichen Automaten:

als TM (mit Stoppen bei Eingabeende)

$\widetilde{\text{Time}}(\text{DFA})(a_1 \dots a_n) := n$

$\widetilde{\text{Space}}(\text{DFA})(a_1 \dots a_n) := n$

alternativ, gemäß "kein Schreiben":

$\widetilde{\widetilde{\text{Space}}}(\text{DFA})(a_1 \dots a_n) := 0$

} unabhängig von (konkretem) DFA

bleibt zu betrachten:

$\| Q \| \hat{=}$ Speicherbedarf für Programm

$\hat{=}$ "Darstellungsbreite" bei binärer Darstellung

$\hat{=}$ Schnelligkeit der Schaltfunktionen

↝ Hardware-Aufwand, beeinflusst z.B.: #(Gatter), Fan-in, Fan-out, ...

beeinflusst:

Signallaufzeiten,
Taktfrequenz
⋮

gegeben: $DFA = (Q, \Gamma, \delta: Q \times \Gamma \to Q, q_0, F)$

Aufgabe: "minimiere $\|Q\|$!"

genauer: finde DFA' mit

(1) $L(DFA) = L(DFA')$

(2) f. alle DFA'' mit $L(DFA) = L(DFA'')$:

$$\|Q''\| \geq \|Q'\|$$

warum könnte $\|Q\|$ "unnötigerweise zu groß" sein?

- Q hat "überflüssigen Zustand" q, d.h.

  q ist für keine Eingabe erreichbar:

  f. alle $w \in \Gamma^*$: $\tilde{\delta}(q_0, w) \neq q$

- Q hat "äquivalente Zustände" p und q, d.h.

  ausgehend von p bzw. q werden die gleichen Worte akzeptiert:

  f. alle $w \in \Gamma^*$: $\tilde{\delta}(p, w) \in F$ gdw $\tilde{\delta}(q, w) \in F$

Def: $p \equiv_{DFA} q$

Eigenschaften von $\equiv_{DFA}$:

$\equiv_{DFA}$ ist Äquivalenzrelation auf Q, d.h.

reflexiv, symmetrisch, transitiv

(entsprechende Eigenschaften der "logischen Äquivalenz" übertragen sich auf $\equiv_{DFA}$)

$\equiv_{DFA}$ bestimmt Zerlegung von Q in disjunkte Klassen

der Art $[q]_{\equiv_{DFA}} := \{ p \mid p \equiv_{DFA} q \}$

$$Q/_{\equiv_{DFA}} := \{ [q]_{\equiv_{DFA}} \mid q \in Q \}$$

zu DFA entsprechenden <u>Äquivalenzklassenautomat</u> bilden:

DFA' mit $Q' := Q/_{\equiv_{DFA}}$

$\Gamma' := \Gamma$

$\delta'([q]_{\equiv_{DFA}}, a) := [\delta(q,a)]_{\equiv_{DFA}}$

$q_0' := [q_0]_{\equiv_{DFA}}$

$F' := \{ [q]_{\equiv_{DFA}} \mid q \in F \}$

Satz (a) DFA' ist wohldefiniert

(b) $L(DFA) = L(DFA')$

Beweis:

zu (a): ad Q' : siehe "Diskrete Strukturen"

ad Γ': ✓

ad δ': sei $p \equiv_{DFA} q$

↷ Def. $\equiv_{DFA}$ f.alle $w \in \Gamma^*$: $\tilde{\delta}(p,w) \in F$ gdw $\tilde{\delta}(q,w) \in F$

↷ zerlege: $w = aw'$ f.alle $a \in \Gamma$, $w' \in \Gamma^*$: $\tilde{\delta}(p,aw') \in F$ gdw $\tilde{\delta}(q,aw') \in F$

↷ Def. $\tilde{\delta}$ als Erweiterung von δ f.alle $a \in \Gamma, w' \in \Gamma^*$: $\tilde{\delta}(\delta(p,a),w') \in F$ gdw $\tilde{\delta}(\delta(q,a),w') \in F$

↷ Def $\equiv_{DFA}$ f.alle $a \in \Gamma$: $\delta(p,a) \equiv_{DFA} \delta(q,a)$

ad q_0' : ✓

ad F': sei $p \equiv_{DFA} q$

↷ Def. $\equiv_{DFA}$: f.alle $w \in \Gamma^*$: $\tilde{\delta}(p,w) \in F$ gdw $\tilde{\delta}(q,w) \in F$

↷ speziell $w := \varepsilon$: $\underbrace{\tilde{\delta}(p,\varepsilon) \in F}_{= p}$ gdw $\underbrace{\tilde{\delta}(q,\varepsilon) \in F}_{= q}$

Def. $\tilde{\delta}(.,\varepsilon)$

zu (b):

	DFA	DFA'
Eingabe:	$a_1\ a_2 \dots a_n$	$a_1\ a_2 \dots a_n$
Zustandsfolge:	$q_0, q_1, q_2, \dots, q_n$	$[q_0], [q_1], [q_2], \dots, [q_n]$
Akzeptieren	$q_n \in F$	$[q_n] \in F'$

wechselseitige Simulation:

⟶ : unmittelbar gemäß Def.!

⟵ : Repräsentanten jeweils geeignet wählbar!

Aufgabe: " minimiere $\|Q\|$! "

Ansatz:
- entferne überflüssige Zustände
- identifiziere äquivalente Zustände
  (bilde Äquivalenzklassenautomat)

↷ Satz: erkennt die gleiche Sprache

- $Q/\equiv_{DFA}$ minimal ?

im Folgenden: ja !

Definition Sei $DFA = (Q, \Gamma, \delta, q_0, F)$.

$\sim_{DFA} \subseteq \Gamma^* \times \Gamma^*$ sei definiert durch:

$x \sim_{DFA} y$: gdw $\tilde{\delta}(q_0, x) = \tilde{\delta}(q_0, y)$

Eigenschaften:

- reflexiv, symmetrisch, transitiv $\rightsquigarrow \sim_{DFA}$ Äquivalenzrelation
- rechtsinvariant, d.h.

  falls $x \sim_{DFA} y$ dann f.alle $z \in \Gamma^*$: $xz \sim_{DFA} yz$

  Bew: $\tilde{\delta}(q_0, xz) = \tilde{\delta}(\tilde{\delta}(q_0, x), z)$ Def. $\tilde{\delta}$

  $= \tilde{\delta}(\tilde{\delta}(q_0, y), z)$ $x \sim_{DFA} y$

  $= \tilde{\delta}(q_0, yz)$ Def. $\tilde{\delta}$

- $\| \Gamma^* / \sim_{DFA} \| = \| \{ q \mid q \in Q, \text{ex. } x \in \Gamma^* : \tilde{\delta}(q_0, x) = q \} \|$

  vermöge Zuordnung: $[x]_{\sim_{DFA}} \leftrightarrow \tilde{\delta}(q_0, x) \in Q$ erreichbar

- $L(DFA) = \bigcup_{\tilde{\delta}(q_0, x) \in F} [x]_{\sim_{DFA}}$

Definition [Nerode-Relation]

Sei $L \subset \Gamma^*$.

$\sim_L \subset \Gamma^* \times \Gamma^*$ sei definiert durch:

$x \sim_L y$:gdw f.alle $w \in \Gamma^*$: $xw \in L$ gdw $yw \in L$

Eigenschaften:

- reflexiv, symmetrisch, transitiv $\rightsquigarrow$ $\sim_L$ Äquivalenzrelation
- rechtsinvariant, d.h.

  falls $x \sim_L y$ dann f.alle $z \in \Gamma^*$: $xz \sim_L yz$

  Bew: sei $x \sim_L y$ und $z \in \Gamma^*$

  zu zeigen: f.alle $\bar{w} \in \Gamma^*$: $xz\bar{w} \in L$ gdw $yz\bar{w} \in L$

  wähle jeweils $w := z\bar{w} \in \Gamma^*$

  dann: $xw \in L$ gdw $yw \in L$ wegen $x \sim_L y$

Satz von Nerode

Sei $L \subset \Gamma^*$.

Dann sind die folgenden Aussagen äquivalent:

(1) ex. DFA $= (Q, \Gamma, \delta, q_0, F)$

mit $L = L(\text{DFA}) = \{w \mid \tilde{\delta}(q_0, w) \in F\}$

(2) ex. rechtsinvariante Äquivalenzrelation $\sim$ auf Γ^*

mit: $\Gamma^*/\sim$ ist endlich und

$$\text{ex. } W \subset \Gamma^* : \quad L = \bigcup_{w \in W} [w]_\sim \qquad (*)$$

(3) für $\sim_L \subset \Gamma^* \times \Gamma^*$ mit: $x \sim_L y$: gdw f. alle $w \in \Gamma^*$: $xw \in L$ gdw $yw \in L$

gilt: $\Gamma^*/\sim_L$ ist endlich

Beweis:

(1) $\Rightarrow$ (2): $\sim_{\text{DFA}}$ mit: $x \sim_{\text{DFA}} y$: gdw $\tilde{\delta}(q_0, x) = \tilde{\delta}(q_0, y)$

hat die in (2) geforderten Eigenschaften

(2) ⇒ (3): wir zeigen:

" $\sim$ wird durch $\sim_L$ vergröbert ", d.h.

falls $x \sim y$, dann $x \sim_L y$, bzw.

$$[x]_\sim \subset [x]_{\sim_L}$$

$$\rightsquigarrow \| \Gamma^* / \sim_L \| \leq \underbrace{\| \Gamma^* / \sim \|}_{\text{endlich}}$$

endlich nach Voraussetzung (2)

Bew: sei $x \sim y$; betrachte $w \in \Gamma^*$:

$xw \in L$ gdw $[xw]_\sim \subset L$ (*)

gdw $[yw]_\sim \subset L$ rechtsinvariant

gdw $yw \in L$ (*)

(3) ⇒ (1): DFA wie folgt definieren:

$Q := \Gamma^* / \sim_L$ endlich nach Voraus. (3)

$q_0 := [\varepsilon]_{\sim_L}$

$F := \{ [x]_{\sim_L} \mid x \in L \}$

$\delta([x]_{\sim_L}, a) := [xa]_{\sim_L}$

Beh. 1: F ist wohldefiniert.

Bew: $x \sim_L y \overset{\text{Def.}}{\sim}$ f.alle $w \in \Gamma^*$: $xw \in L$ gdw $yw \in L$

$\sim$ speziell $w := \varepsilon$: $x \in L$ gdw $y \in L$

Beh. 2: δ ist wohldefiniert.

Bew: $x \sim_L y \overset{\text{rechtsinvariant}}{\sim} xa \sim_L ya$

Beh. 3: $L = L(\text{DFA})$

Bew: $L(\text{DFA}) \overset{\text{Def. } q_0}{=} \{ w \mid \tilde{\delta}([\varepsilon]_{\sim_L}, w) \in F \}$

$\overset{\text{Def. } \delta}{=} \{ w \mid [w]_{\sim_L} \in F \}$

$\overset{\text{Def. } F}{=} \{ w \mid w \in L \} = L$

Korollar zum Satz von Nerode

Der zu "(3) ⇒ (1)" konstruierte Automat ist minimal (bzg. # (Zustände).

Beweis: Sei DFA" mit $L(DFA") = L(DFA)$:

$$\| Q \| = \| \Gamma^* / \sim_L \| \qquad \text{Def. in "(3)} \Rightarrow \text{(1)"}$$

$$\leq \| \Gamma^* / \sim_{DFA"} \| \qquad \text{Beweis "(2)} \Rightarrow \text{(3)"}$$

$$\leq \| Q" \| \qquad \text{Beweis "(1)} \Rightarrow \text{(2)"}$$

Satz Sei $DFA = (Q, \Gamma, \delta, q_0, F)$ ohne überflüssige Zustände mit $L = L(DFA)$. Dann ist der Äquivalenzklassenautomat zu DFA,

$$\left(Q/_{\equiv_{DFA}}, \Gamma, \delta'([q]_{\equiv_{DFA}}, a) := [\delta(q,a)]_{\equiv_{DFA}}, \right.$$

$$\left. [q_0]_{\equiv_{DFA}}, \{ [q]_{\equiv_{DFA}} \mid q \in F \} \right)$$

minimal (bzg. #(Zustände), d.h. insbesondere:

$$\| Q/_{\equiv_{DFA}} \| \leq \underbrace{\| \Gamma^*/_{\sim_L} \|}_{\text{minimal gemäß Kor. Satz von Nerode}}$$

wir zeigen: ex. $f: \Gamma^*/_{\sim_L} \xrightarrow{\text{surjektiv}} Q/_{\equiv_{DFA}}$

definiere: $f([x]_{\sim_L}) := [\tilde{\delta}(q_0, x)]_{\equiv_{DFA}}$

ad surjektiv: DFA ohne überflüssige Zustände

$\rightsquigarrow$ Äquivalenzklassenautomat ohne überflüssige Zustände

$\rightsquigarrow$ f.alle $[q] \in Q/\equiv_{DFA}$ ex. $x \in \Gamma^*$ mit

$$[q]_{\equiv_{DFA}} = [\tilde{\delta}(q_0, x)]_{\equiv_{DFA}} = f([x]_{\sim_L})$$

ad wohldefiniert: sei $x \sim_L y$

Def. $\sim_L$
$\rightsquigarrow$ f.alle $w \in \Gamma^*$: $xw \in L$ gdw $yw \in L$

$L = L(DFA)$
$\rightsquigarrow$ f.alle $w \in \Gamma^*$: $\tilde{\delta}(q_0, xw) \in F$ gdw $\tilde{\delta}(q_0, yw) \in F$

Def. $\tilde{\delta}$
$\rightsquigarrow$ f.alle $w \in \Gamma^*$: $\tilde{\delta}(\tilde{\delta}(q_0, x), w) \in F$ gdw
$\tilde{\delta}(\tilde{\delta}(q_0, y), w) \in F$

Def $\equiv_{DFA}$
$\rightsquigarrow$ $\tilde{\delta}(q_0, x) \equiv_{DFA} \tilde{\delta}(q_0, y)$

Bemerkung:

- überflüssige Zustände durch Tiefensuche, ausgehend von q_0 bestimmbar:

  q überflüssig gdw q von q_0 "nicht erreichbar"

- äquivalente Zustände polynomiell (Zeit-) beschränkt bestimmbar:

  $p \equiv q$: gdw f.alle $w \in \Gamma^*$: $\tilde{\delta}(p,w) \in F$ gdw $\tilde{\delta}(q,w) \in F$

  - es reicht, nur $w \in \Gamma^*$ mit

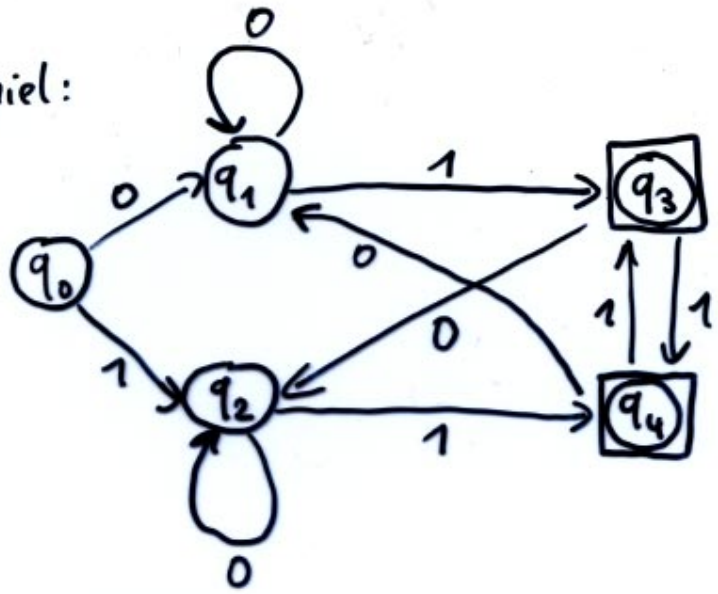
    Länge$(w) \leq \|Q\|$

    zu betrachten

  - alternativ: nicht-äquivalente Zustände in $O(\|Q\|^2 \cdot \|\Gamma\|)$ bestimmen

    ($\rightsquigarrow$ We, 96-97)

Beispiel:



Eingaben der Länge 0: bilde Zerlegung gemäß F

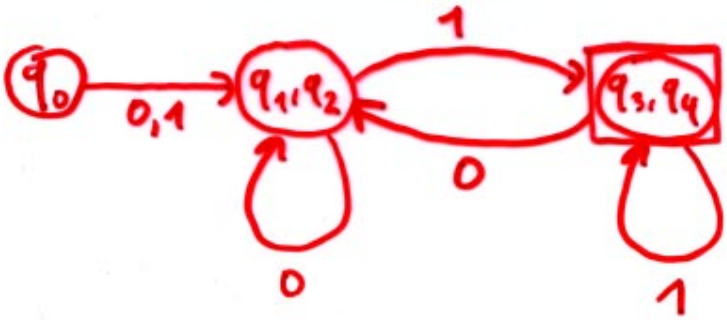
$$F = \{q_3, q_4\} \qquad Q \setminus F = \{q_0, q_1, q_2\}$$

Eingaben der Länge $i > 0$: prüfe, ob für Zustände einer Klasse auch Folgezustände aus gleicher Klasse, zerlege gegebenenfalls

F: ✓

$Q \setminus F$: $\{q_0\}, \{q_1, q_2\}$

Äquivalenzklassenautomat:



welche "Aufgaben" (Sprachen) sind durch endliche Automaten entscheidbar ?

gegeben: $L \subset \Gamma^*$

Frage : ex. DFA mit $L = L(DFA)$?

Antworten:

(1) benutze Satz von Nerode als (hinreichendes und notwendiges) Kriterium:

- bestimme für Nerode-Relation $\sim_L$ mit

  $x \sim_L y$:gdw f.alle $w \in \Gamma^*$: $xw \in L$ gdw $yw \in L$

  die Kardinalität $\| \Gamma^* / \sim_L \|$

- falls endlich, dann

  $L = L\left(\left(\Gamma^* / \sim_L , \Gamma , \delta([x]_{\sim_L}, a) := [xa]_{\sim_L} , [\varepsilon]_{\sim_L} , \{ [x]_{\sim_L} \mid x \in L \} \right) \right)$

  falls unendlich, dann

  f.alle DFA: $L \neq L(DFA)$

(2) benutze "Pumping Lemma" als notwendige Bedingung:

"grob": { L besitzt "innere Regelmäßigkeit"; DFA "durchläuft Schleifen" für lange Eingaben

Pumping Lemma

Sei $L = L(DFA)$ mit $DFA = (Q, \Gamma, \delta, q_0, F)$

Dann gilt:

ex. $n \in \mathbb{N}$:

f. alle $z \in L$ mit Länge$(z) \geq n$:

ex. $u, v, w \in \Gamma^*$ mit

- $z = uvw$
- Länge$(uv) \leq n$
- Länge$(v) \geq 1$:

f. alle $l \in \mathbb{N}$: $uv^l w \in L$

Beweis: $n := \| Q \|$ tut es! denn:

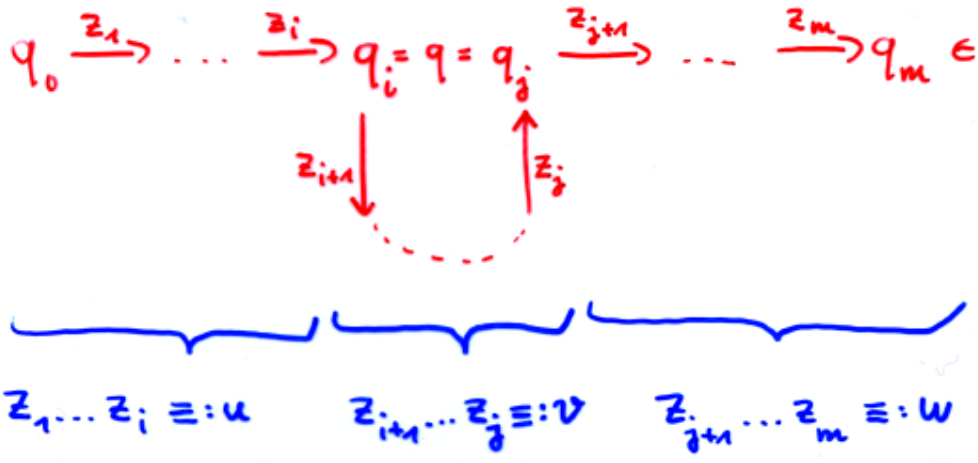
sei $z = z_1 \dots z_n \dots z_m \in L$ mit Länge$(z) = m \geq n$

betrachte Verhalten von DFA bei Eingabe von z:

$$q_0 \xrightarrow{z_1} q_1 \xrightarrow{z_2} q_2 \dots \xrightarrow{z_n} q_n \xrightarrow{z_{n+1}} \dots \xrightarrow{z_m} q_m \in F$$

wegen $\{q_0, q_1, \dots, q_n\} \subset Q$ mit $\|Q\| = n$

ex. $q \in Q$ und $0 \le i < j \le n$ mit $q_i = q = q_j$, d.h.



dann gilt: $\tilde{\delta}(q_0, uv^\ell w)$

$= \tilde{\delta}(\underbrace{\tilde{\delta}(q_0, u)}_{=q}, v^\ell w)$

$= \tilde{\delta}(q, v^\ell w)$

$= \tilde{\delta}(\underbrace{\tilde{\delta}(q, v^\ell)}_{=q}, w)$

$= \tilde{\delta}(q, w) \in F$

also: $\tilde{\delta}(q_0, uv^\ell w) \in L$ f. alle $\ell \in \mathbb{N}$

typische Anwendung des Pumping Lemmas

Vermutung: gegebene Sprache $L \subset \Gamma^*$
nicht durch endlichen Automaten entscheidbar

indirekter Beweis (versuch):

angenommen doch

Pumping Lemma ↷ ex. $n \in \mathbb{N}$ .... f. alle $l \in \mathbb{N}$: $uv^l w \in L$

konstruiere Gegenbeispiel z ↯

Beh: Die "Klammersprache"

$L = \{0^k 1^k \mid k \geq 1\} \subset \{0,1\}^*$ ist

nicht durch endlichen Automaten entscheidbar.

Bew: angenommen doch

mit "Pumping-Lemma-Zahl" n

betrachte: $z \equiv \underbrace{0 \ldots 0}_{n\text{-mal}} \underbrace{1 \ldots 1}_{n\text{-mal}}$

zerlege gemäß Pumping Lemma: $z = u\ v\ w$

Länge$(uv) \leq n$

$\rightsquigarrow$ $u \equiv 0^{\text{Länge}(u)}$

$v \equiv 0^{\text{Länge}(v)} \neq \varepsilon$

"pumping"

$\rightsquigarrow$ $u\, 0^{\ell \cdot \text{Länge}(v)}\, w \in L$, f. alle $\ell \in \mathbb{N}$

$\rightsquigarrow$ ↯ für $\ell \neq 1$ ($\ell = 0$: zu wenig 0's

$\ell \geq 2$: zu viele 0's)

Beh: Die "Palindromsprache"

$$L := \{ w \mid w = w^R \} \subset \{0,1\}^*$$

(⋮ Spiegelbild (von hinten gelesen)) ist

nicht durch endlichen Automaten entscheidbar.

Bew: angenommen doch

mit "Pumping-Lemma-Zahl" n

betrachte: $z = \underbrace{0 \ldots 0}_{n\text{-mal}} 1 \underbrace{0 \ldots 0}_{n\text{-mal}} \in L$

entsprechend "Klammersprache":

$u\, 0^{\ell \cdot \text{Länge}(v)}\, w \in L$, f. alle $\ell \in \mathbb{N}$

↷ ↯ für $\ell \neq 1$

Nichtdeterministische endliche Automaten

bislang: deterministisch, d.h. Übergangsfunktion

$$\delta : Q \times \Gamma \to Q$$

nunmehr auch: nichtdeterministisch

$NFA = (Q, \Gamma, \delta, q_0, F)$ mit Überg. relation

$\delta_{(rel)} \subset (Q \times \Gamma) \times Q$ bzw.

$\delta_{(set)} : Q \times \Gamma \longrightarrow \mathcal{P}Q$ ← möglicherweise "nichtdeterministisch" (oder "partiell")

$$\left[\begin{array}{l} \delta_{(set)}(q,a) := \{ \bar{q} \mid (q,a,\bar{q}) \in \delta_{(rel)} \} \quad \text{bzw.} \\ (q,a,\bar{q}) \in \delta_{(rel)} \quad : \text{gdw} \quad \bar{q} \in \delta_{(set)}(q,a) \end{array} \right]$$

Fortsetzung von δ: $\tilde{\delta} : Q \times \Gamma^* \to \mathcal{P}Q$

$w = \varepsilon$: $\tilde{\delta}(q, \varepsilon) := \{q\}$

$w \equiv a\bar{w} \equiv \bar{\bar{w}}b$: $\tilde{\delta}(q,w) := \bigcup_{\bar{q} \in \delta(q,a)} \tilde{\delta}(\bar{q},\bar{w}) = \bigcup_{\bar{\bar{q}} \in \tilde{\delta}(q,\bar{\bar{w}})} \delta(\bar{\bar{q}},b)$

akzeptierte Sprache: $L(NFA) := \{ w \in \Gamma^* \mid \tilde{\delta}(q_0,w) \cap F \neq \emptyset \}$

Nichtdeterminismus

(1) hilft (dem Informatik-Ingenieur) bei Spezifikation von Aufgabenlösungen

(Abstraktion von "Suchen", "Auswählen")

(2) ist automatisch entfernbar durch (semantikerhaltende) Transformations-Werkzeuge

Bsp zu (1): Sprache für "vollen Rucksack" A mit

$$L_4 := \{ w_1 \dots w_n \mid n \in \mathbb{N},\ w_i \in \underbrace{\Gamma}_{=\{1,\dots,A\}},\ \text{ex. } I \subset \{1,\dots,n\} : \sum_{i \in I} w_i = A \}$$

nichtdeterministisch entscheiden durch:

$q \in Q := \{0,\dots,A\}$ bedeutet: "aus bislang gelesener Eingabe ist durch nichtdeterministische Auswahlen die Summe q bildbar"

$q_0 := 0 \quad ; \quad F := \{A\}$

$$\delta(q,a) := \{ \underbrace{q}_{\text{"a nicht gewählt"}},\ \underbrace{q+a}_{\text{"a gewählt"}} \} \cap Q$$

Bsp zu (1): Sprache für "einfaches String Matching"

für $s = s_1 \dots s_k \in \Gamma^*$ mit

$$L_s := \{ m \mid m \in \Gamma^* \text{ und } \underbrace{\text{"}m \text{ enthält } s\text{"}}_{\text{ex. } m_1, m_2 \text{ mit } m \equiv m_1 s m_2} \}$$

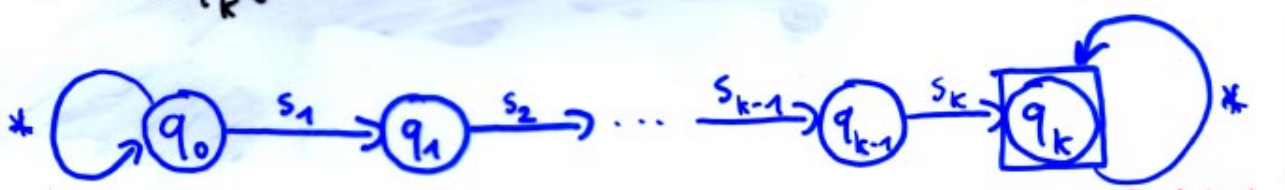
nichtdeterministisch entscheiden durch:

$Q := \{q_0, q_1, \dots, q_k\} \ni q_i$ bedeutet:

"letzten i Zeichen waren $s_1 \dots s_i$"

$$\delta := \{ (q_0, a, q_0) \mid a \in \Gamma \}$$
$$\cup \{ (q_0, s_1, q_1), (q_1, s_2, q_2), \dots, (q_{k-1}, s_k, q_k) \} \cup \{ (q_k, a, q_k) \mid a \in \Gamma \}$$

$F := \{q_k\}$



Anfangsstück m_1 überlesen

beim Lesen von s_1 den Anfang von s "erraten"

nach Lesen von $s_1 \dots s_k$ "Richtigkeit erkennen"

Endstück m_2 überlesen

zu (2) [semantikerhaltende Transformations-Werkzeuge]:

Satz Für alle nichtdeterministischen

$$NFA = (Q, \Gamma, \delta: Q \times \Gamma \to \mathcal{P}Q, q_0, F)$$

gibt es einen deterministischen DFA' mit

$$L(NFA) = L(DFA').$$

(= "semantikerhaltend")

Beweis: Konstruiere "Potenzmengen-Automaten" zu NFA:

$$Q' := \mathcal{P}Q \quad \rightsquigarrow \quad \|Q'\| = 2^{\|Q\|}$$

$$\Gamma' := \Gamma$$

$$\delta'(q', a) := \bigcup_{q \in q'} \underbrace{\delta(q, a)}_{\subset Q \text{ gemäß Def. } \delta}$$

($q' \subset Q$ gemäß Def. Q)

$$\rightsquigarrow \quad \subset Q$$

$$\rightsquigarrow \quad \in \mathcal{P}Q =: Q'$$

$$q_0' := \{q_0\}$$

$$F' := \{ q' \mid q' \cap F \neq \emptyset \}$$

Beh 1: f. alle $w \in \Gamma^*$:

$$\tilde{\delta}'(q_0', w) = \tilde{\delta}(q_0, w)$$

Bew: (mit Induktion über Länge von w)

$w = \varepsilon$:

$\tilde{\delta}'(q_0', \varepsilon) = q_0'$ Def. "det. Erweiterung"

$= \{q_0\}$ Def. q_0'

$= \tilde{\delta}(q_0, \varepsilon)$ Def "nichtdet. Erweiterung"

$w \equiv \bar{\bar{w}} b$: $\tilde{\delta}'(q_0', \bar{\bar{w}} b) = \delta'(\tilde{\delta}'(q_0', \bar{\bar{w}}), b)$

Def./Satz "det. Erweiterung"

$= \delta'(\underbrace{\tilde{\delta}(q_0, \bar{\bar{w}})}_{\in \mathcal{P}Q =: Q'}, b)$ Ind. Annahme

$= \bigcup_{\bar{\bar{q}} \in \tilde{\delta}(q_0, \bar{\bar{w}})} \delta(\bar{\bar{q}}, b)$ Def. δ'

$= \tilde{\delta}(q_0, \bar{\bar{w}} b)$ Def. "nichtdet. Erweiterung"

Beh. 1: f.alle $w \in \Gamma^*$: $\tilde{\delta}'(q_0', w) = \tilde{\delta}(q_0, w)$

Beh. 2: $L(DFA') = L(NFA)$

Bew:

$w \in L(DFA')$

gdw $\tilde{\delta}'(q_0', w) \in F'$ Def. "det. Akzeptieren"

gdw $\tilde{\delta}'(q_0', w) \cap F \neq \emptyset$ Def. F'

gdw $\tilde{\delta}(q_0, w) \cap F \neq \emptyset$ Beh. 1

gdw $w \in L(NFA)$ Def. "nichtdet. Akzept."

Korollar Für alle NFA mit n Zuständen gibt es DFA' mit 2^n Zuständen und $L(NFA) = L(DFA')$.

Bemerkung:
- Potenzmengen-Automat kann "überflüssige Zustände" enthalten; "leicht vermeidbar"!
- auch dann ggf. noch "exponentieller Zustands-Anstieg"

zu (2): semantikerhaltende Transformations-Werkzeuge

nichtdeterministische Spezifikation

Ingenieur

↓

NFA

Werkzeuge

↓

Entfernung von Nichtdeterminismus: Potenzmengen-Automat

↓

DFA'

↓

Entfernung von überflüssigen Zuständen

↓

Zustandsminimierung: Äquivalenzklassen-Automat

↓

DFA_{min}

als (weitere) Grundlage für "Werkzeuge":

(getypte) Operationen auf (getypten) Sprachen

(A) allgemeine Überlegungen

(B) speziell für reguläre Sprachen

durch endlichen Automaten entscheidbar

zu (A):

bislang (und "allgemein üblich")

manchmal "großzügige" Redeweisen, wie z.B.
"Sei L eine Sprache ..."

jetzt einmal (und für Werkzeug-Implementierungen notwendig)

"pingelig" und genau(er)

Sei Γ ein Alphabet (endliche Menge).

$L \subset \Gamma^*$ heißt Sprache über Γ.

$$\mathcal{L}_\Gamma := \{ L \mid L \subset \Gamma^* \} = \mathcal{P}\,\Gamma^*$$

Operationen auf $\mathcal{L}_\Gamma$ mit Werten aus $\mathcal{L}_\Gamma$:

Operationen des Potenzmengen-Verbands:

Durchschnitt $\quad L_1 \cap_\Gamma L_2 := \{ w \mid w \in L_1 \text{ und } w \in L_2 \}$

Vereinigung $\quad L_1 \cup_\Gamma L_2 := \{ w \mid w \in L_1 \text{ oder } w \in L_2 \}$

Komplement $\quad \complement_\Gamma L := \{ w \mid w \in \Gamma^* \text{ und } w \notin L \}$

abgeleitet, z.B. Differenz

$$L_1 \setminus_\Gamma L_2 := \{ w \mid w \in L_1 \text{ und } w \notin L_2 \}$$
$$= L_1 \cap_\Gamma \complement_\Gamma L_2$$

"Gesetze", z.B. de Morgan

$$L_1 \cap_\Gamma L_2 = \complement_\Gamma (\complement_\Gamma L_1 \cup_\Gamma \complement_\Gamma L_2)$$
$$L_1 \cup_\Gamma L_2 = \complement_\Gamma (\complement_\Gamma L_1 \cap_\Gamma \complement_\Gamma L_2)$$

"Wortverarbeitungs-Operationen":

Konkatenation (Produkt)
$$L_1 \bullet_\Gamma L_2 := \{ w \mid \text{ex. } u_1 \in L_1, \text{ex. } u_2 \in L_2 : w = u_1 u_2 \}$$

abgeleitet, z.B

Potenzen
$$L^{0_\Gamma} := \{ \varepsilon \}$$

für $n > 0$:
$$L^{n_\Gamma} := L^{(n-1)_\Gamma} \bullet_\Gamma L$$
$$= \{ w \mid \text{ex } u_1 \in L, \ldots, \text{ex } u_n \in L : w = u_1 \ldots u_n \}$$

Kleenescher Abschluss (*-Bildung)
$$L^{*_\Gamma} := \bigcup_{n \in \mathbb{N}} L^{n_\Gamma}$$

Quotient
$$L_1 /_\Gamma L_2 := \{ w \mid \text{ex. } u \in L_2 : wu \in L_1 \}$$

↑ entsteht aus Wort $wu \in L_1$ mit Suffix $u \in L_2$ durch Streichen des Suffix

⋮

Operationen auf $\mathfrak{L}_\Gamma$ mit Werten aus $\mathbb{B} = \{0,1\} \hat{=} \{\text{false}, \text{true}\}$

Leerheits-Test $\quad \text{empty}_\Gamma(L) := \begin{cases} 1 & \text{falls } L = \emptyset \\ 0 & \text{sonst} \end{cases}$

Enthaltensein-Test $\quad \text{subset}_\Gamma(L_1, L_2) := \begin{cases} 1 & \text{falls } L_1 \subset_\Gamma L_2 \\ 0 & \text{sonst} \end{cases}$

Gleichheits-Test $\quad \text{equal}_\Gamma(L_1, L_2) := \begin{cases} 1 & \text{falls } L_1 = L_2 \\ 0 & \text{sonst} \end{cases}$

Endlichkeits-Test $\quad \text{finite}_\Gamma(L) := \begin{cases} 1 & \text{falls } \|L\| \in \mathbb{N} \\ 0 & \text{sonst, d.h. } \|L\| = \aleph_0 \end{cases}$

$\vdots$

Operationen auf $\mathcal{L}_\Gamma$ mit weiteren Argumenten

Substitution für $f: \Gamma \to \mathcal{L}_\Delta$

Erweiterung auf Γ^*: $\tilde{f}: \Gamma^* \to \mathcal{L}_\Delta$ mit

$$\tilde{f}(\varepsilon) := \{\varepsilon\}$$

$$\tilde{f}(w_1 \dots w_n) := f(w_1) \bullet_\Delta f(w_2) \bullet_\Delta \dots \bullet f(w_n) \in \mathcal{L}_\Delta$$

Erweiterung auf $\mathcal{L}_\Gamma$: $\tilde{f}: \mathcal{L}_\Gamma \to \mathcal{L}_\Delta$ mit

$$\tilde{f}(L) := \bigcup_{w \in L}{}_\Delta \; \tilde{f}(w) \in \mathcal{L}_\Delta$$

speziell
Homomorphismus für $f: \Gamma \to \{\{w\} \mid w \in \Delta^*\} \subset \mathcal{L}_\Delta$

inverser Homomorphismus für $h: \Delta \to \{\{w\} \mid w \in \Gamma^*\}$

$$h^{-1}(L) := \{z \mid z \in \Delta^* \text{ und } \tilde{h}(z) \subset L\}$$

•
•
•

Alternativen für Typbindung (z.B.)

(1) ["strenge Typbindung"]:

f. alle Γ jeweils einzeln $\mathfrak{L}_\Gamma$ mit zugehörigen Operationen betrachten

Eigenschaften, z.B.: für $\Gamma \subsetneq \Sigma$

(i) $L \in \mathfrak{L}_\Gamma \rightsquigarrow L \in \mathfrak{L}_\Sigma$

(ii) $L_1, L_2 \in \mathfrak{L}_\Gamma \rightsquigarrow L_1 \cap_\Gamma L_2 = L_1 \cap_\Sigma L_2$

(iii) $L \in \mathfrak{L}_\Gamma \rightsquigarrow \wp_\Gamma L \subsetneq \wp_\Sigma L$

(iv) $L_1, L_2 \in \mathfrak{L}_\Gamma \rightsquigarrow L_1 \bullet_\Gamma L_2 = L_1 \bullet_\Sigma L_2$

(v) $L_1, L_2 \in \mathfrak{L}_\Gamma \rightsquigarrow \text{subset}_\Gamma(L_1, L_2) = \text{subset}_\Sigma(L_1, L_2)$

⋮

(2) ["implizite Typen"]:

alle $\mathcal{L}_\Gamma$, für Γ "beliebiges" Alphabet,

zusammen betrachten: $\mathcal{L} := \bigcup_{\Gamma \text{ Alphabet}} \mathcal{L}_\Gamma$

dann "gebräuchliche" Konventionen, z.B.:

- Qualifikation durch Γ "weglassen"
- alle Operationen "überladen" verwenden
- statt Komplement (Γ wichtig!) nur Differenz verwenden, insbesondere

  $\mathcal{L}_\Gamma(L)$ ersetzen durch $\Gamma^* \setminus L$

  ↑ wichtig ↑ überladen

- impliziten (aktiven, minimalen, ...) Typ definieren:

  für $L \in \mathcal{L}$ sei

  $\text{type}(L) := \bigcap_{\Gamma \text{ Alphabet}} \{\Gamma \mid L \in \mathcal{L}_\Gamma\}$

- implizite Typanpassung annehmen, z.B.

  für $L_1, L_2 \in \mathcal{L}$:

  $L_1 \cap L_2 := L_1 \cap_{\text{type}(L_1) \cup \text{type}(L_2)} L_2$

(3) [" explizite Typen "]:

alle $\mathfrak{L}_\Gamma$, für Γ "beliebiges" Alphabet,

zusammen betrachten, aber

Typangaben explizit "mitschleppen":

$$\text{typed\_}\mathfrak{L} := \bigcup_{\Gamma \text{ Alphabet}} \{ (L, \Gamma) \mid L \in \mathfrak{L}_\Gamma \}$$

alle Operationen auf typed_$\mathfrak{L}$ liefern dann auch

expliziten Ergebnistyp, z. B.:

$$\mathfrak{C}(L, \Gamma) := (\mathfrak{C}_\Gamma L, \Gamma)$$

$$(L_1, \Gamma_1) \cup (L_2, \Gamma_2) := (L_1 \cup_{\Gamma_1 \cup \Gamma_2} L_2, \Gamma_1 \cup \Gamma_2)$$

$\mathcal{L}_r$, $\mathcal{L}$ bzw. typed_$\mathcal{L}$ als Datentyp betrachten

Spezifikation (Semantik):

- Grundmenge (semantische Domäne) : $\mathcal{L}_r$, $\mathcal{L}$ bzw. typed_$\mathcal{L}$
- Operationen : siehe oben

Implementierung (Syntax, Algorithmen)

- Elemente der Grundmenge endlich darstellen, z.B.

  Code Menge der (syntaktisch erlaubten) Darstellungen

  denote : Code $\longrightarrow \mathcal{L}$ "Semantikfunktion"

- Operationen verwirklichen (implementieren) durch Algorithmen (Programme) auf Darstellungen

z.B. für Programm P mit Semantik $\sigma(P)$:

$$\sigma(P) : Code \times Code \longrightarrow Code$$

$$\begin{array}{ccccc} \downarrow \text{denote} & & \downarrow \text{denote} & & \downarrow \text{denote} \\ op & : & \mathcal{L} \times \mathcal{L} & \longrightarrow & \mathcal{L} \end{array}$$

Implementierungsaufgabe:

zu $op : \mathcal{L} \times \mathcal{L} \to \mathcal{L}$ ein Programm P bestimmen, so dass Diagramm kommutiert!

Bedeutung bestimmen:

zu P mit $\sigma(P) : Code \times Code \to Code$ diejenige Operation op bestimmen, so dass Diagramm kommutiert!

Verifikationsaufgabe:

für $op : \mathcal{L} \times \mathcal{L} \to \mathcal{L}$ und

$\sigma(P) : Code \times Code \to Code$

nachweisen, dass Diagramme kommutiert!
(bzw. widerlegen)

(getypte) Operationen auf (getypten) Sprachen

(A) allgemeine Überlegungen: siehe oben

(B) speziell für reguläre Sprachen:

- $\mathcal{L}_\Gamma^{reg} := \{ L \mid L \in \mathcal{L}_\Gamma \text{ und ex. DFA} : L = L(DFA) \}$
- statt $\mathcal{L}_\Gamma$ überall $\mathcal{L}_\Gamma^{reg}$, insbesondere

  statt $\mathcal{L}$ überall $\mathcal{L}^{reg} := \bigcup_{\Gamma \text{ Alphabet}} \mathcal{L}_\Gamma^{reg}$
- zu klären:

Abschlusseigenschaften: alle Operationen wohldefiniert?

Darstellungen: z.B.: $Code_1 := \{ DFA \mid DFA \text{ ist deterministischer endlicher Automat} \}$

$denote_1(DFA) := L(DFA)$

oder:

$Code_2 := \{ NFA \mid NFA \text{ ist nichtdeterministischer endlicher Automat} \}$

$denote_2(NFA) := L(NFA)$

(effiziente) Programme: bezüglich gewählter Darstellung Code mit denote

im Folgenden:

$Code_1 := \mathcal{DFA} := \{ DFA \mid DFA$ ist deterministischer endlicher Automat $\}$

$denote_1 := L_{DFA}$ mit

$L_{DFA}((Q, \Gamma, \delta, q_0, F)) :=$

$\{w \mid \tilde{\delta}(q_0, w) \in F\} \in \mathcal{L}_\Gamma$

"Programme" für Operationen:

Durchschnitt:

$(Q_1, \Gamma, \delta_1, q_{01}, F_1)$
$(Q_2, \Gamma, \delta_2, q_{02}, F_2)$ $\Big] \xrightarrow{P_n}$

$(Q_1 \times Q_2,$
$\Gamma,$
$\delta((q_1, q_2), a) := (\delta_1(q_1, a), \delta_2(q_2, a)),$
$(q_{01}, q_{02}),$
$F := \{(q_1, q_2) \mid q_1 \in F_1 \text{ und } q_2 \in F_2\})$

Vereinigung: analog mit

$$F := \{ (q_1, q_2) \mid q_1 \in F_1 \text{ oder } q_2 \in F_2 \}$$

Komplement:

$$(Q, \Gamma, \delta, q_0, F) \xmapsto{P_c} (Q, \Gamma, \delta, q_0, Q \setminus F)$$

Konkatenation:

$(Q_1, \Gamma, \delta_1, q_{01}, F_1)$
$(Q_2, \Gamma, \delta_2, q_{02}, F_2)$

mit o.B.d.A $Q_1 \cap Q_2 = \emptyset$
(sonst umbenennen)

$\xrightarrow{P_\bullet}$ (minimierter) Potenzmengen-Automat zu

$$\Big(Q_1 \cup Q_2,$$

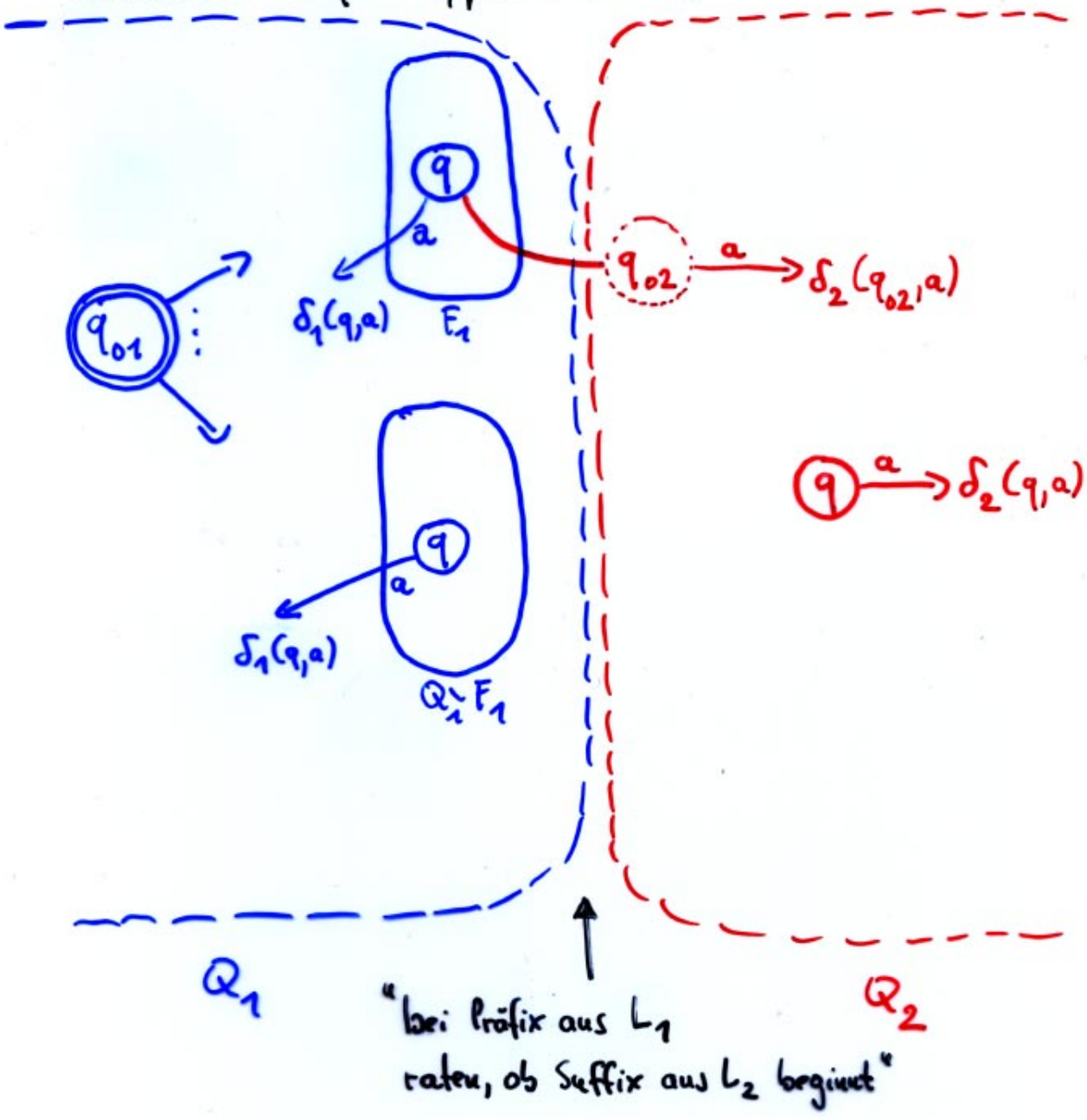
$$\Gamma,$$

$$\delta(q,a) := \begin{cases} \{\delta_1(q,a)\} & \text{falls } q \in Q_1 \setminus F_1 \\ \{\delta_1(q,a), \delta_2(q_{02},a)\} & \text{falls } q \in F_1 \\ \{\delta_2(q,a)\} & \text{falls } q \in Q_2 \end{cases}$$

$$q_{01},$$

$$F := \begin{cases} F_2 & \text{falls } \varepsilon \notin L_2 \\ F_1 \cup F_2 & \text{falls } \varepsilon \in L_2 \end{cases} \Big)$$

ausdrücklich (für "typischen Fall"):



falls $\varepsilon \in L_2$: kein Übergang nach Q_2 erforderlich

$\rightsquigarrow F_1 \subset F$

Potenzen: durch Iteration

Kleenescher Abschluss:

$(Q, \Gamma, \delta, q_0, F) \xmapsto{P_*}$

(minimierter) Potenzmengen-Automat zu

$$\Big(Q,\ \Gamma,\ \delta^*(q,a) := \begin{cases} \{\delta(q,a)\} & \text{falls } q \in Q \setminus F \\ \{\delta(q,a),\ \delta(q_0,a)\} & \text{falls } q \in F \end{cases},$$

$\delta(q,a)$: raten, ob neues Teilwort aus L beginnt

$\delta(q_0,a)$: bei Teilwort aus L

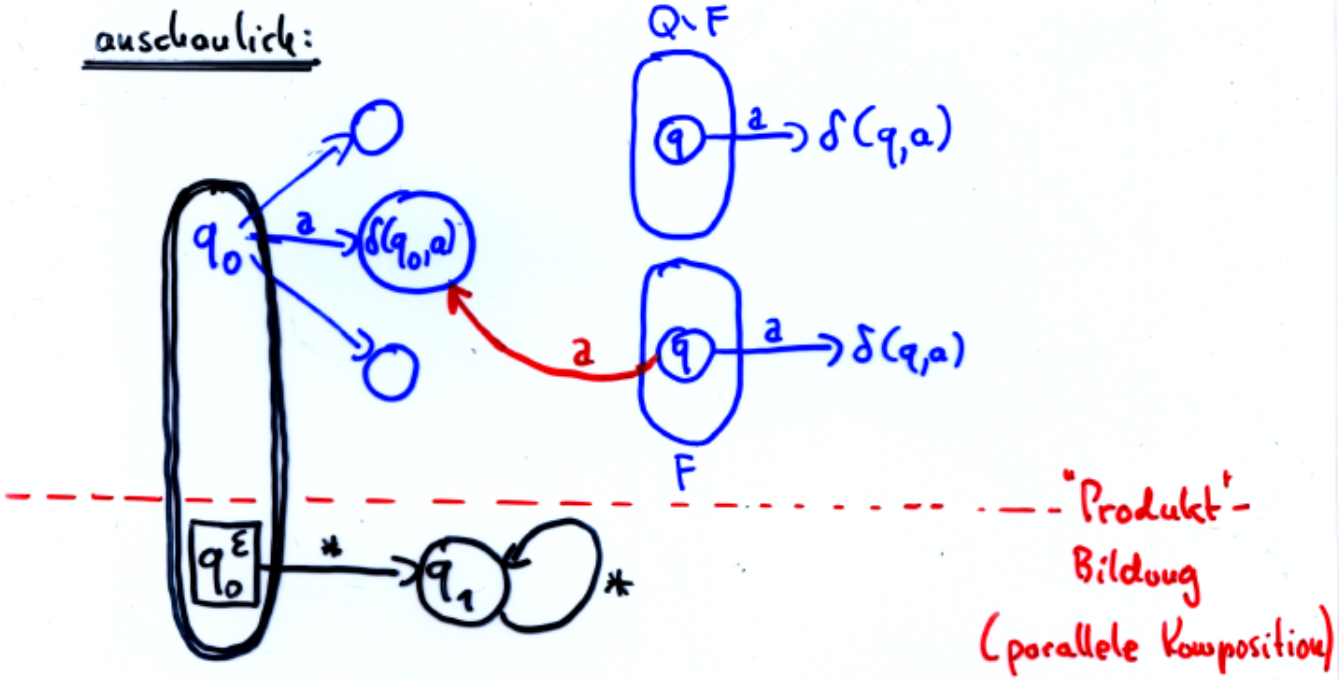
$$q_0,\ F \Big)$$

verwöge P_v verknüpft mit

$$\big(\{q_0^\varepsilon, q_1^\varepsilon\},\ \Gamma,\ \delta^\varepsilon(q,a) := q_1^\varepsilon,\ q_0^\varepsilon,\ \{q_0^\varepsilon\} \big)$$

akzeptiert $\{\varepsilon\}$

anschaulich:



Quotient :

$(Q_1, \Gamma, \delta_1, q_{01}, F_1)$

$(Q_2, \Gamma, \delta_2, q_{02}, F_2)$ $\Big\rceil \; P/$

$(Q_1, \Gamma, \delta_1, q_{01},$

$F := \{ q \mid q \in Q_1$ und ex. $u \in L_2 : \tilde{\delta}_1(q,u) \in F_1 \}$ $)$

gdw ex. Weg von q nach F_1 (im Graphen zu δ_1) mit einer "Wegmarkierung" aus L_2

algorithmisch:
- bestimme alle (schleifenfreien) Wege von q nach F_1;
- bestimme für jeden solchen Weg die "Markierung";
- prüfe jeweils, ob Markierung aus L_2 (vermöge $(Q_2, \Gamma, \delta_2, q_{02}, F_2)$);

Leerheits-Test:

$(Q, \Gamma, \delta, q_0, F) \vdash \xrightarrow{P_{empty}} \begin{cases} 0 & \text{falls ex. Weg von } q_0 \text{ nach } F \text{ (im Graphen zu } \delta) \\ 1 & \text{sonst} \end{cases}$

Enthaltenseins-Test:

Voraussetzung: $L_1 \subset L_2$ gdw $L_1 \setminus L_2 = \emptyset$

gdw $L_1 \cap \complement L_2 = \emptyset$

also: P_{subset}:

(1) konstruiere Automat für $L_1 \cap \complement L_2$

(2) wende P_{empty} an

Gleichheits-Test: P_{equal}:

(a) wende P_{subset} auf L_1, L_2 an

(b) wende P_{subset} auf L_2, L_1 an

(c) falls beide Ergebnisse 1 dann 1
sonst 0

Endlichkeits-Test:

$(Q, \Gamma, \delta, q_0, F) \xmapsto{P_{\text{finite}}}$

(1) entferne überflüssige Zustände

(2) prüfe ob:

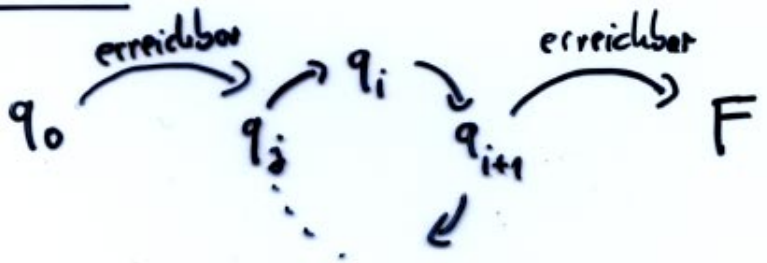
"im Graphen ex. Kreis $q_i, q_{i+1}, \dots, q_j$

mit:

vom Kreis ist F erreichbar"

gdw L ist unendlich

anschaulich:



"in etwa:" $u \quad v^l \quad w \in L$, P. alle $l \in \mathbb{N}$

mit Länge$(v) \geq 1$

Substitution: siehe Vorgeschichte, Satz 4.6.14, Seite 121

Beispiel:

$\Gamma := \{0,1\}$

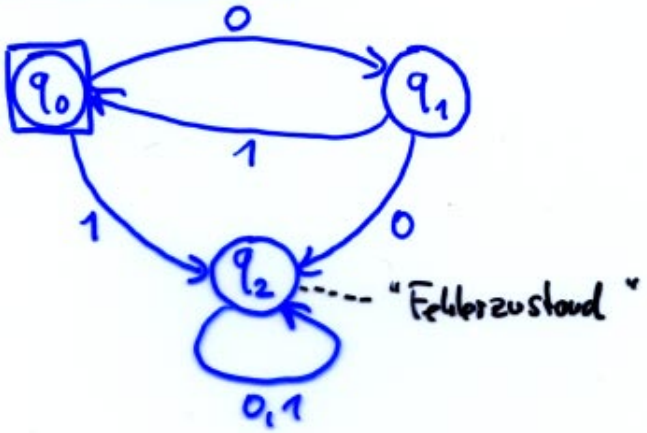
$\Delta := \{a,b,e\}$

$L := \{\varepsilon, 01, 0101, 010101, \ldots\}$

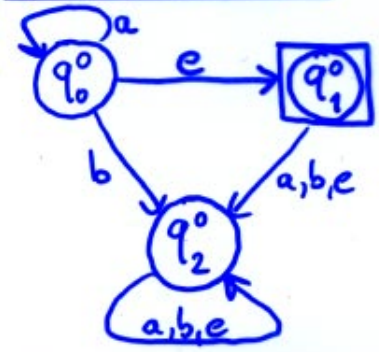
$f(0) := \{ a^m e \mid m \geq 0 \}$

$f(1) := \{ b^l e \mid l \geq 0 \}$

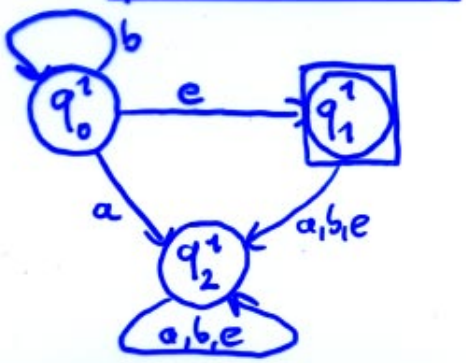
$L = L(DFA)$ für:

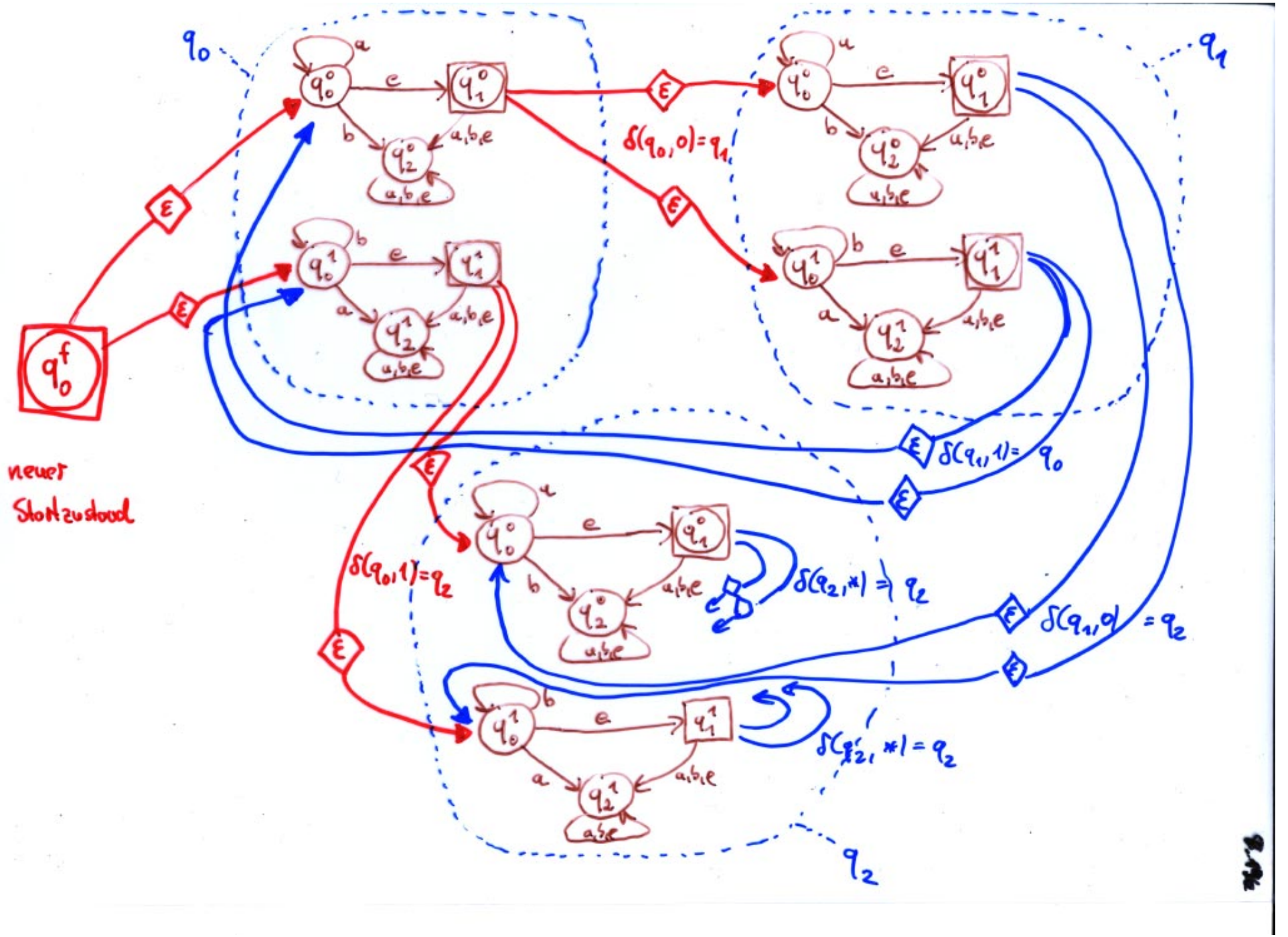


$f(0) = L(DFA^0)$:



$f(1) = L(DFA^1)$:




q_0
q_1
q_2
q_0^f
neuer Startzustand
$\delta(q_0, 0) = q_1$
$\delta(q_1, 1) = q_0$
$\delta(q_0, 1) = q_2$
$\delta(q_2, *) = q_2$
$\delta(q_1, 0) = q_2$
$\delta(q_2', *) = q_2$
a,b,e

inverser Homomorphismus: für $h : \Delta \to \{\{w\} \mid w \in \Gamma^*\}$

$(Q, \Gamma, \delta, q_0, F) \xmapsto{P_{h^{-1}}}$

$(Q,$

$\Delta,$

$\delta^{h^{-1}}(q, a) := \tilde{\delta}(q, \bigcup x [x \in h(a)])$

[dann: $\tilde{\delta}^{h^{-1}}(q, w) = \tilde{\delta}(q, \bigcup x [x \in \tilde{h}(w)])$]

$q_0,$

$F)$

der gelieferte Automat akzeptiert die Sprache

$\{z \mid z \in \Delta^* \text{ und } \underbrace{\tilde{\delta}^{h^{-1}}(q_0, z) \in F}\}$

gdw $\tilde{\delta}(q_0, \bigcup x [x \in \tilde{h}(z)]) \in F$

gdw $\{\tilde{h}(z)\} \subset L(Q, \Gamma, \delta, q_0, F)$

Satz • Die Klasse der regulären Sprachen ist abgeschlossen unter den Operationen

Durchschnitt
Vereinigung
Komplement
Konkatenation (Produkt)
Potenz
Kleenescher Abschluss
Quotient
Substitution
Homomorphismus
inverser Homomorphismus

und die Probleme

Element? (Wort)
Leerheit?
Enthaltensein?
Gleichheit?
Endlichkeit?

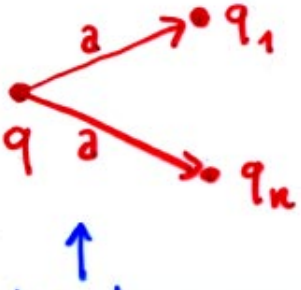
sind entscheidbar.

- Genauer: die obigen Aussagen gelten "effektiv" bezüglich der Darstellung von regulären Sprachen durch
  a) deterministische endliche Automaten,
  b) nichtdeterministische endliche Automaten.

Nichtdeterminismus mit ε-Übergängen

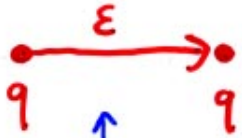
- hilfreich für Spezifikation

- bisher: nur bezüglich Verhalten
  beim "Verarbeiten (Lesen)"
  eines Eingabezeichens



↑
beim Lesen von a "raten",
welcher Folgezustand aus
$\delta(q,a) = \{q_1, \ldots, q_n\}$
"erfolgreich" sein wird

zusätzlich: auch Bezügliche
"Nichtstun":

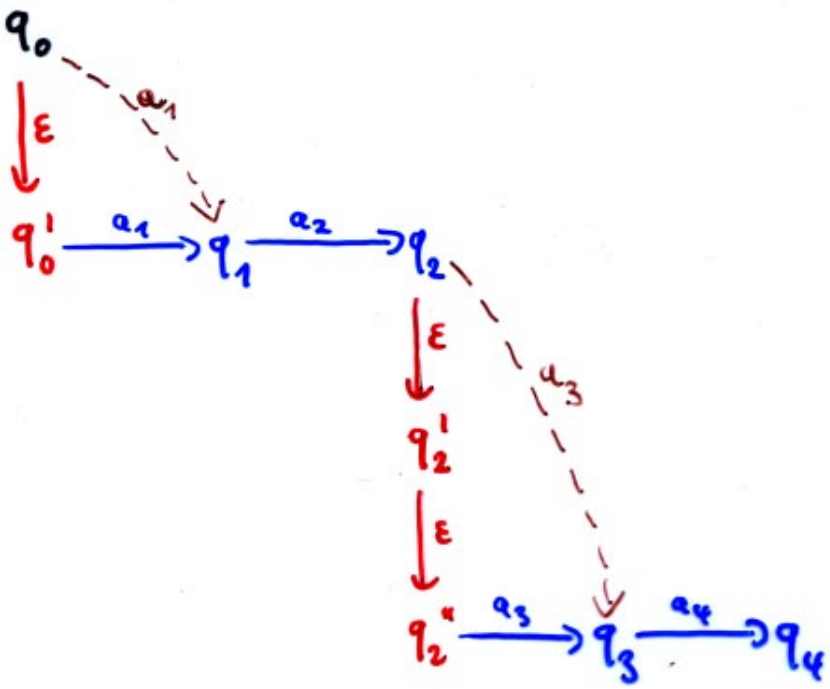


↑
"vor" (bzw. "nach") dem Lesen: "spontan"

Beispielablauf:

Zeit :	1	2	3	4	..
Eingabe :	a_1	a_2	a_3	a_4	..

Zustand :



formal: Übergangsfunktion mit Signatur

$$\delta : Q \times (\Gamma \cup \{\varepsilon\}) \to \mathcal{P}Q$$

$\delta(q, \varepsilon) \ni q'$ bedeutet:

"ohne Lesen einer Eingabe"

"spontan" den Zustand wechseln

$\tilde{\delta}$ geeignet "redefinieren"!

Satz Für alle nichtdeterministischen

$\varepsilon\text{-NFA} = (Q, \Gamma, \delta: Q \times (\Gamma \cup \{\varepsilon\}) \to \mathcal{P}Q, q_0, F)$

mit ε-Übergängen

gibt es einen nichtdeterministischen

NFA' ohne ε-Übergänge mit

$L(\varepsilon\text{-NFA}) = L(\text{NFA}')$

Beweisidee:

$Q' := Q \quad ; \quad q_0' := q_0$

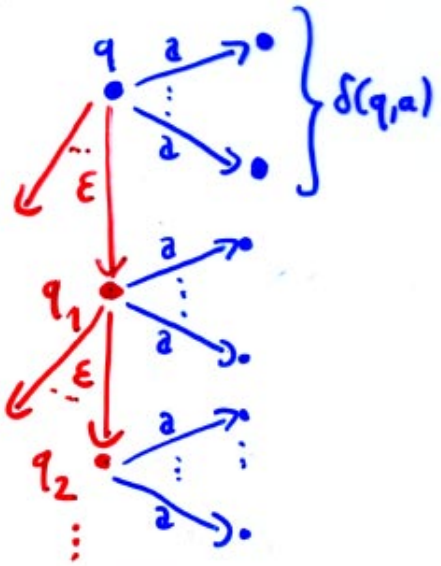
$\delta'(q,a) := \delta(q,a)$

$\cup \{ q' \mid \text{ex } q_1, \dots, q_n \in Q :$

$\delta(q,\varepsilon) \ni q_1 ,$

$\delta(q_i,\varepsilon) \ni q_{i+1} \text{ für } i = 1,\dots,n-1,$

$\delta(q_n,a) \ni q' \}$

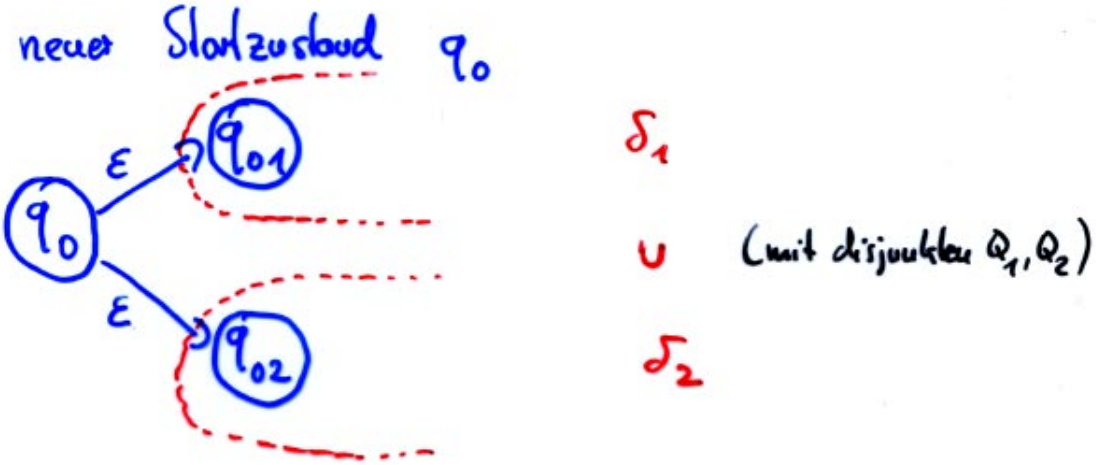


Bem: es reicht "schleifenfreie" Folgen von ε-Übergängen zu betrachten (mit Länge $\leq \|Q\|$)

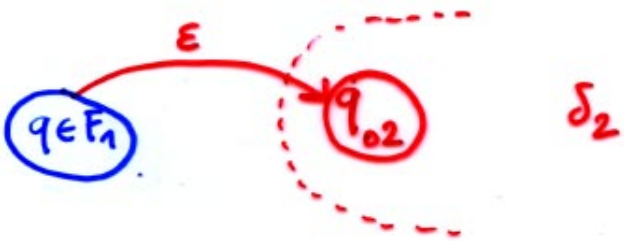
ε - Übergänge sind hilfreich für Spezifikation

Beispiele:

für Vereinigung:



für Konkatenation:



eine Sprache entscheiden versus eine Sprache erzeugen

gegeben:

(deklarative Definition einer) Sprache L über Alphabet Σ

$$L \subset \Sigma^*$$

gesucht:

(endliche) algorithmische Beschreibung Akz (Programm, Maschine, ...), die L entscheidet, d.h.

bei Eingabe: $w \in \Sigma^*$

Ausgabe: $\begin{cases} 1 \ (\hat{=} \text{ true}) & \text{falls } w \in L \\ 0 \ (\hat{=} \text{ false}) & \text{falls } w \notin L \end{cases}$

(endliche) algorithmische Beschreibung Gen (Programm, Maschine, ...), die L "erzeugt", d.h.

"Gen kann genau die Elemente von L erzeugen"

schwächere Form:

eine Sprache (nur) erkennen:

Ausgabe = 1 gdw $w \in L$

ein schon behandelter Ansatz:

Sprachklasse: Klasse der rekursiv aufzählbaren Sprachen

algorithmische Beschreibungen: (nichtdeterm.) Turing-Programme

Sprache erkennen

TM mit

$\sigma(TM)(w) = 1$ gdw $w \in L$

- starte mit w auf Band
- berechne $\sigma(TM)(w)$
- gib Ergebnis aus

Sprache "erzeugen"

TM mit $\sigma(TM)$ total

und $range(\sigma(TM)) = L$:

- starte mit leerem Band
- erzeuge (nichtdeterm.) ein $n \in \Sigma^*$
- berechne $\sigma(TM)(n)$
- gib Ergebnis aus

genau die Elemente von L so "erzeugbar"

eine Umdeutung von endlichen Automaten

Sprachklasse: Klasse der regulären Sprachen

algorithmische Beschreibungen: (nichtdeterm.) endliche Automaten

"eingeschränkte Turing-Programme"

Sprache entscheiden — Sprache "erzeugen"

für $(Q, \Sigma, \delta, q_0, F)$

Eingabe: $w = w_1 \dots w_n \in \Sigma^*$

Zustandsfolge: durch Eingabe bestimmt vermöge $\delta(q_{i-1}, w_i) = q_i$,

$q_0, q_1, \dots, q_n$

Ausgabe: $q_n \in F$

als "Wahrheitswert"

Eingabe: –

Zustandsfolge: nichtdeterministisch erraten gemäß δ,

$q_0, q_1, \dots, q_n$

mit q_n ist Element von F

Ausgabe:

ein Wort $w = w_1 \dots w_n$ mit

$\delta(q_{i-1}, w_i) = q_i$

- durchlaufe Wbg gemäß δ
- "sammle Wegmarkierung auf"

eine formale(re) Betrachtung

$\underbrace{q_0 w_1}\, w_2 \dots w_n$	q_0
$\vdash_{\delta(q_0, w_1) = q_1}$	$\vdash_{q_0 \to w_1 q_1}$
$\underbrace{q_1 w_2}\, \dots w_n$	$w_1 q_1$
$\vdash_{\delta(q_1, w_2) = q_2}$	$\vdash_{q_1 \to w_2 q_2}$
$\underbrace{q_2 w_3}\, \dots w_n$	$w_1 w_2 q_2$
$\vdots$	$\vdots$
$\vdash_{\delta(q_{n-1}, w_n) = q_n}$	$\vdash_{q_{n-1} \to w_n q_n}$
q_n	$w_1 w_2 \dots w_n q_n$
$\vdash_{q_n \in F}$ ε (accept)	$\vdash_{q_n \to \varepsilon}$ $w_1 w_2 \dots w_n$ für $q_n \in F$

als Ersetzungsregel

$\delta(q, a) = q'$	$\hat{=}$ $qa \to q'$	$q \to a q'$
$q \in F$	$\hat{=}$ $q \to \varepsilon$	$q \to \varepsilon$

eine "Neudeutung" von Turing-Programmen

für $TM = (Q, \Gamma, \delta: Q \times \Gamma \to Q \times \Gamma \times \{R, L, N\}, q_0, \ldots)$

Ausgangssituation: $\# q_0 \beta_1 \ldots \beta_\ell \#$ mit $\beta_i \in \Sigma \subset \Gamma$,

Σ Eingabealphabet

Situationen $\# \alpha q \beta \#$ mit $\alpha \in \Gamma^*$, $q \in Q$, $\beta \in \Gamma^*$

Übergänge: $\# \overbrace{\alpha_1 \ldots \alpha_{k-1} \alpha_k}^{\alpha \in \Gamma^*} q \overbrace{\beta_1 \beta_2 \ldots \beta_n}^{\beta \in \Gamma^*} \# \vdash$

"lesen"

$\delta(q, \beta_1) = (q', \beta_1', N)$: $\# \alpha_1 \ldots \alpha_{k-1} \boxed{\alpha_k q' \beta_1'} \beta_2 \ldots \beta_\ell \#$

$\delta(q, \beta_1) = (q', \beta_1', R)$: $\# \alpha_1 \ldots \alpha_{k-1} \boxed{\alpha_k \beta_1' q'} \beta_2 \ldots \beta_\ell \#$

$\delta(q, \beta_1) = (q', \beta_1', L)$: $\# \alpha_1 \ldots \alpha_{k-1} \boxed{q' \alpha_k \beta_1'} \beta_2 \ldots \beta_\ell \#$

unverändert — lokal ersetzen — unverändert

alle Ersetzungen von Form:

$$\alpha_k\, q\, \beta_1 \longrightarrow \alpha_k\, q'\, \beta_1' \qquad \text{"still"}$$

$$\alpha_k\, q\, \beta_1 \longrightarrow \alpha_k\, \beta_1'\, q' \qquad \text{"rechts"}$$

$$\alpha_k\, q\, \beta_1 \longrightarrow q'\, \alpha_k\, \beta_1' \qquad \text{"links"}$$

mit $\alpha \in \Gamma$, $\beta_1, \beta_1' \in \Gamma$, $q, q' \in Q$

<u>vereinfacht (und verallgemeinert):</u>

$$q\,\beta_1 \longrightarrow q'\,\beta_1' \qquad \text{"still"}$$

$$q\,\beta_1\beta_2 \longrightarrow \beta_1'\,q'\,\beta_2$$

$$q\,\beta_1\,\# \longrightarrow \beta_1'\,q'\,B\,\#$$

↑ blank

"rechts"

$$\alpha_k\, q\, \beta_1 \longrightarrow q'\, \alpha_k\, \beta_1'$$

$$\#\, q\, \beta_1 \longrightarrow \#\, q'\, B\, \beta_1$$

links

alles geeignet ausgearbeitet und formalisiert (↝ Salomaa):

TM deuten als Menge von <u>Ersetzungsregeln</u>

ein sehr allgemeines Modell (für [erzeugende"] Berechenbarkeit): 9.7

Definition:

- $RW = (\Sigma, P)$ mit

  Σ Alphabet (endliche Menge)

  $P \subset_{\text{endlich}} \Sigma^* \times \Sigma^*$ Menge von "Produktionen" (Ersetzungsregeln, ...)

  heißt Ersetzungssystem (rewriting system).

- statt $(u, v) \in P$ häufig: $u \to v$

- RW induziert Übergangsrelation

  $\Rightarrow_{RW} \subset \Sigma^* \times \Sigma^*$ mit

  $x \Rightarrow_{RW} y$: gdw ex. $x_1, x_2 \in \Sigma^*$,

  ex. $u \to v \in P$:

  $x = x_1 u x_2$ und

  $y = x_1 v x_2$

- $\overset{*}{\Rightarrow}_{RW}$ ist reflexiver, transitiver Abschluss von $\Rightarrow_{RW}$.

ein immer noch "sehr allgemeines" Modell mit

Unterscheidung: "Nichtterminal": Kontrolle, Zustand, ... (z.B.)

"Terminal": Eingabe, Ausgabe, ... (z.B.)

Definition:

- $G = (\underbrace{RW}_{(\Sigma, P)}, \Sigma_N, \Sigma_T, S)$ mit

  (RW: Ersetzungssystem; Σ_N: Nichtterminal-Alph.; Σ_T: Terminal-Alph.; S: Startsymbol; (Σ, P): Produktionen)

  RW Ersetzungssystem [alle linken Seiten enthalten Nichtterminal-Zeichen]

  $\Sigma = \Sigma_N \cup \Sigma_T$ und $\Sigma_N \cap \Sigma_T = \emptyset$

  $S \in \Sigma_N$

  heißt **Grammatik**.

- G **erzeugt** die Sprache

$$L(G) := \{ w \mid w \in \Sigma_T^* \text{ und } S \overset{*}{\Rightarrow}_{RW} w \} \subset \Sigma_T^*$$

- sequentiell, jeweils eine Ersetzungsstelle benutzt, Reihenfolge "nichtdeterministisch" gewählt

Beispiel 1:

deute endlichen Automaten $(Q, \Gamma, \delta, q_0, F)$ als Grammatik um:

$$\left.\begin{aligned} \Sigma_N &:= Q \\ \Sigma_T &:= \Gamma \end{aligned}\right\} \quad \Sigma = \Sigma_N \cup \Sigma_T$$

$$P := \{ q \to aq' \mid \delta(q,a) = q' \} \cup \{ q \to \varepsilon \mid q \in F \}$$

$$S := q_0$$

dann gilt:

$L((\Sigma, P), \Sigma_N, \Sigma_T, S)$ "erzeugende" Grammatik

$= L(Q, \Gamma, \delta, q_0, F)$ "entscheidender Automat"

Beispiel 2: "korrekte Klammerungen"

$\Sigma_T := \{ (\ ,\ ) \}$

$\Sigma_N := \{ S \}$

$P := \{ \quad S \to (S) \ , \quad S \to S\,S \ , \quad S \to \varepsilon \quad \}$

z.B.:

$S \Rightarrow S\,S$

$\Rightarrow S\,(S)$

$\Rightarrow S\,((S))$

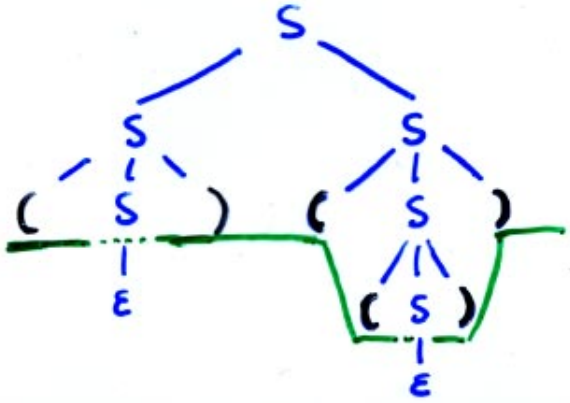
$\Rightarrow (S)((S))$

$\Rightarrow (S)(())$

$\Rightarrow ()(())$

} eine "Ableitung"

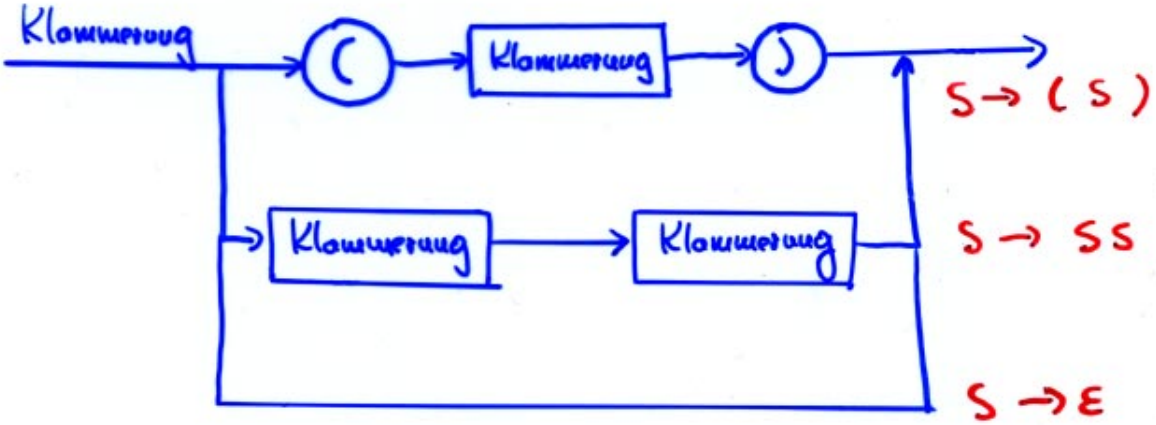
wegen besonderer Form von P als Baum darstellbar:



erzeugtes "Terminalwort": Blätter (ohne ε)

besondere Form: linke Seite : Element aus Σ_N

dann auch als Syntaxdiagramm deutbar, z.B.



Beispiel 3: (Teile) natürlicher Sprachen, z.B.

$S \longrightarrow N_p \quad V_p$

sentence — noun phrase — verb phrase

$N_p \longrightarrow A \quad N$

article — noun

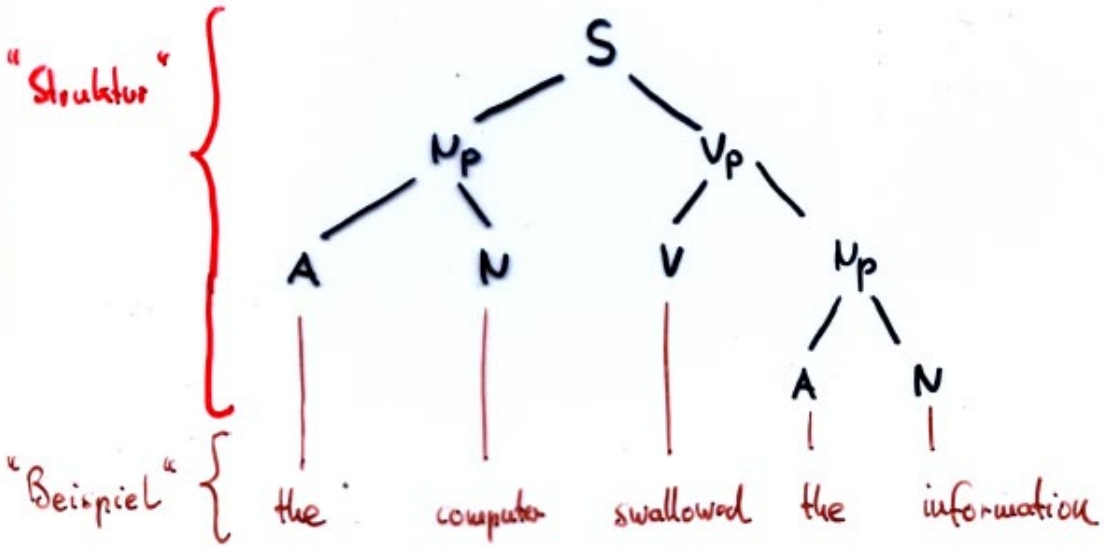
$V_p \longrightarrow V \quad N_p$

verb

$N \longrightarrow$ computer
$N \longrightarrow$ information
$\vdots$

$A \longrightarrow$ the
$\vdots$

$V \longrightarrow$ swallowed
$\vdots$



Chomsky Typ	Sprachklasse	allgemeine Form der Produktionen	Normalform	äquivalente Definition laut "Wegner"
0	rekursiv aufzählbar	$u \to w$ mit $u \in (\Sigma_N \cup \Sigma_T)^*$ [mindestens ein $x \in \Sigma_N$] $w \in (\Sigma_N \cup \Sigma_T)^*$	$A \to BC$ $AB \to CD$ $A \to \varepsilon$ $A \to a$ mit $A, B, C, D \in \Sigma_N$ $a \in \Sigma_T$	$u \to w$ mit $u \in \Sigma_N^+$ $w \in (\Sigma_N \cup \Sigma_T)^*$
2	kontextfrei	$A \to w$ mit $A \in \Sigma_N$, $w \in (\Sigma_N \cup \Sigma_T)^*$	$A \to BC$ $A \to a$ $(A \to \varepsilon)$ mit $A, B, C \in \Sigma_N$ $a \in \Sigma_T$	$A \to w$ mit $A \in \Sigma_N$ $w \in (\Sigma_N \cup \Sigma_T)^*$
3	regulär	$A \to wB$ $A \to w$ mit $A, B \in \Sigma_N$ $w \in \Sigma_T^*$	$A \to aB$ $A \to \varepsilon$ mit $A, B \in \Sigma_N$ $a \in \Sigma_T$	$A \to aB$ $A \to \varepsilon$ mit $A, B \in \Sigma_N$ $a \in \Sigma_T$

Normalformen besagen u.a. Folgendes:

- **immer möglich:**

  1) erst "Nichtterminal" ableiten ↝ Struktur bestimmen

     dann "Terminal" ableiten ↝ Beispiel einsetzen

  2) nur Zusammenhänge (Teilworte) der Länge 2 betrachten

- **für kontextfrei:**

  3) nur lokal ersetzen, d.h. nur "Zusammenhänge" (Teilworte) der Länge 1 betrachten

  4) Linksableitungen möglich:
     - immer das linkeste Nichtterminal ersetzen
     - links davon kann sich nichts mehr ändern

Chomsky Typ 1: $w_1 A w_2 \rightarrow w_1 w w_2$

mit $A \in \Sigma_N$,

$w_1, w_2 \in (\Sigma_N \cup \Sigma_T)^*$,

w_1, w_2: Kontext

$w \in (\Sigma_N \cup \Sigma_T)^+$, d.h. $w \neq \varepsilon$

($S \rightarrow \varepsilon$, S kommt auf keiner rechten Seite vor)

kontext-sensitive Grammatik (Sprache)

äquivalente Definition:

$u \rightarrow w$ mit $|u| \leq |w|$ (nichtverkürzend)

($S \rightarrow \varepsilon$, ...)

Satz Zu jeder Typ-0 Grammatik mit Produktionen der Form $u \to w$ mit $u, w \in (\Sigma_N \cup \Sigma_T)^*$ gibt es eine Typ-0 Grammatik in Normalform mit Produktionen der Form

[expandierend] $A \to BC$

[kontext-sensitiv] $AB \to CD$

[löschend] $A \to \varepsilon$

[terminal] $A \to a$

mit $A, B, C, D \in \Sigma_N$,

$a \in \Sigma_T$,

die die gleiche Sprache erzeugt.

Beweisskizze:

- benenne "verbotene" Formen
- simuliere "verbotene" Formen durch
  erlaubte Formen oder
  "weniger verbotene" Formen
- iteriere

verboten	Abhilfe
• "gemischte" Produktionen (außer terminalen)	zu jedem Terminal x ein neues Nichtterminal A_x einführen: – ersetze x durch A_x – $A_x \to x$ neue Produktion
• $\varepsilon \to w$	für alle Nichtterminal A: – $A \to Aw$ neue Produktion – $A \to wA$ neue Produktion
• $A_1 \dots A_m \to B_1 \dots B_n$ mit $n < m$ analog: $A_1 \to B_1$ $A_1 \dots A_m \to \varepsilon$ mit $m \geq 2$	rechte Seite verlängern: – hänge rechts neues Nichtterminal B an – $B \to \varepsilon$ neue Produktion
• $A_1 \dots A_m \to B_1 \dots B_n$ mit $m \leq n$, $n \geq 3$	führe neue Nichtterminal ein und bilde "Ketten":

$m = 1$: für $A \to B_1 B_2 B_3 \dots B_n$

- führe neue Nichtterminal $C_1, \dots, C_{n-2}$ ein
- ersetze durch neue Produktionen:

$$A \to B_1 C_1$$

$$C_1 \to B_2 C_2$$

$$\vdots$$

$$C_{n-2} \to B_{n-1} B_n$$

$m > 1$: für $A_1 A_2 \dots A_m \to B_1 \dots B_n$

- führe neues Nichtterminal C_2 ein
- ersetze durch

$$A_1 A_2 \to B_1 C_2$$

$$C_2 A_3 \dots A_m \to B_2 \dots B_n$$

erlaubt

beide Seiten um 1 kürzer

Citeriere!)

Satz Sei $L \subset \Sigma_T^*$. Dann sind die folgenden Aussagen äquivalent:

1. L ist rekursiv aufzählbar, d.h.

   ex. TM mit $\sigma(TM)(x) = 1$ gdw $x \in L$

   bzw.

   ex TM mit domain $(\sigma(TM)) = L$

   bzw.

   ex TM mit $\sigma(TM)$ total und range $(\sigma(TM)) = L$

   bzw. ...

2. ex. Typ-0 Grammatik $G = ((\Sigma, P), \Sigma_N, \Sigma_T, S)$ mit $L = L(G)$.

Beweisidee:

"1. $\Rightarrow$ 2." : deute Turing-Programme als Typ-0 Grammatiken

"2. $\Rightarrow$ 1." : das folgende "Erzeugungsverfahren" ist berechenbar:

- für $k = 0, 1, 2, \ldots$

  bilde alle Ableitungen der Länge k

- gib dabei gebildete Terminalworte aus ("numeriere" dabei die Ausgaben)

Satz Sei $L \subset \Sigma_T^*$. Dann sind die folgenden Aussagen äquivalent:

1. L ist regulär, d.h.

   ex. $DFA = (Q, \Sigma_T^*, \delta, q_0, F)$ mit $L(DFA) = L$

   bzw.

   ex. $\epsilon\text{-}NFA' = (Q', \Sigma_T^*, \delta', q_0', F')$ mit $L(\epsilon\text{-}NFA') = L$

2. ex. Typ-3 Grammatik $G = ((\Sigma, P), \Sigma_N, \Sigma_T, S)$ mit $L = L(G)$.

Beweisidee:

benutze folgende Entsprechungen:

endlicher Automat	Typ-3 Grammatik
• Q (als Zustände)	• $Q \hat{=} \Sigma_N$ als Nichtterminal
• $\begin{cases} \delta(q,a) = q' \\ \delta(q,a) \ni q' \end{cases}$	• $q \to aq'$
• q_0 (als Startzustand)	• q_0 als Start-Nichtterminal
• $q \in F$	• $q \to \epsilon$

Satz Zu jeder Typ-3 Grammatik mit Produktionen

der Form $A \to wB$

$A \to w$ mit $A, B \in \Sigma_N, w \in \Sigma_T^*$

gibt es eine Typ-3 Grammatik in Normalform

mit Produktionen der Form

$A \to aB$

$A \to \varepsilon$ mit $A, B \in \Sigma_N, a \in \Sigma_T$

Beweisskizze:

- benenne "verbotene" Formen
- simuliere "verbotene" Formen durch erlaubte Formen

Sei $G = ((\Sigma, P), \Sigma_N, \Sigma_T, S)$ Typ-3 Grammatik,

folgende Formen sind möglich:

$A \to aB$
$A \to \varepsilon$ } erlaubt

$A \to a_1 \dots a_k B$ mit $k \geq 2$
$A \to a_1 \dots a_k$ mit $k \geq 1$
$A \to B$ } "verboten"

zur "verbotenen" Form $A \to a_1 \dots a_k B$ mit $k \geq 2$:

- neue Nichtterminals $B_1, \dots, B_{k-1}$
- neue Produktionen $A \to a_1 B_1$

$$B_1 \to a_2 B_2$$

$$\vdots$$

$$B_{k-1} \to a_k B$$

zur "verbotenen" Form $A \to a_1 \dots a_k$ mit $k \geq 1$:

- neue Nichtterminals $A_1, \dots, A_k$
- neue Produktionen $A \to a_1 A_1$

$$A_1 \to a_2 A_2$$

$$\vdots$$

$$A_{k-1} \to a_k A_k$$

$$A_k \to \varepsilon$$

Zur "verboteneu" Form $A \to B$:

bezüglich neuer Grammatik sei

$$N(A) := \{ B \mid A \overset{*}{\Rightarrow} B, B \in \Sigma_N \} \text{ für } A \in \Sigma_N$$

entspricht ε-Übergängen bei Automaten

- entferne alle "verboteneu" $A \to B$
- ersetze bisherige Produktionen $A \to aB$ durch:

  $A \to aC$, für $C \in N(B)$

- füge hinzu:

  $\overline{S} \to aC$, für $S' \to aB$ mit $S' \in N(S)$ und $C \in N(B)$, d.h. $S \overset{*}{\Rightarrow} S' \Rightarrow aB \overset{*}{\Rightarrow} uC$

  ($\overline{S}$: neues Start-Nichtterminal)

- falls ex. $A \in N(S)$ mit $A \to \varepsilon$,

  dann füge hinzu:

  $\overline{S} \to \varepsilon$

- was ist { berechenbar / einfach berechenbar } ?

- viele (äquivalente) Abstraktionen, z.B.:

		allgemein	bzw.	einfach
• durch Programme / Maschinen bearbeitbar	⇝	TM		DFA
• durch Ersetzungssysteme erzeugbar	⇝	Typ-0		Typ-3
⋮				
• "effektiver" Abschluss "elementarer Gegebenheiten"	⇝	μ-rekursiv		reguläre Ausdrücke

reguläre Sprachen über Σ

elementare Gegebenheiten (z.B.):

$\emptyset$

$\{\varepsilon\}$ sind reguläre Sprachen

$\{a\}$ für $a \in \Sigma$

"effektiver" Abschluss (z.B.):

Die Klasse der regulären Sprachen ist abgeschlossen unter (z.B.):

Vereinigung

Konkatenation

Kleenescher Abschluss

im Folgenden:

- obige Eigenschaften "charakterisieren" die regulären Sprachen
- erlauben "endliche Darstellungen"

Definition [Syntax und Semantik regulärer Ausdrücke]

1. Syntax:

Sei Σ ein Alphabet, und seien

$\emptyset$, ε, $\cup$, $\cdot$, *, $[$, $]$ Zeichen, die nicht in Σ vorkommen.

Die Menge (Sprache) der reguläre Ausdrücke (über Σ) wird induktiv wie folgt definiert:

(i) $\emptyset$
ε
a, für $a \in \Sigma$ sind reguläre Ausdrücke.

(ii) sind α_1 und α_2 reguläre Ausdrücke, so auch

$[\alpha_1] \cup [\alpha_2]$

$[\alpha_1] \cdot [\alpha_1]$

$[\alpha_1]^*$

(iii) "sonst nichts", d.h. formaler:

Die Menge der regulären Ausdrücke Reg_Σ ist die kleinste Menge von Worten über $\Sigma \cup \{\emptyset, \varepsilon, \cup, \cdot, ^*, [,]\}$ die (i) und (ii) erfüllt.

2. Semantik:

Jedem regulären Ausdruck (über Σ) wird induktiv wie folgt eine Sprache über Σ zugeordnet:

$$\tau : Reg_\Sigma \longrightarrow \mathcal{P}\Sigma^*$$

(i) $\tau(\emptyset) := \emptyset$

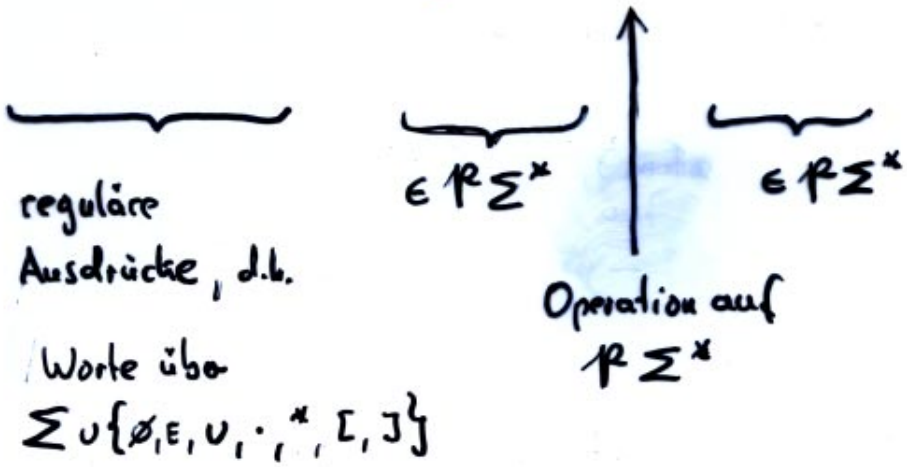
$\tau(\varepsilon) := \{\varepsilon\}$

$\tau(a) := \{a\}$, für $a \in \Sigma$

(ii) $\tau([\alpha_1] \cup [\alpha_2]) := \tau(\alpha_1) \cup \tau(\alpha_2)$

$\tau([\alpha_1] \cdot [\alpha_2]) := \tau(\alpha_1) \cdot \tau(\alpha_2)$

$\tau([\alpha_1]^*) := (\tau(\alpha_1))^*$



Bemerkung:

Die Menge der regulären Ausdrücke, Reg_Σ,

kann durch eine **kontextfreie Grammatik** erzeugt werden:

$S \to \emptyset$

$S \to \varepsilon$

$S \to a$, für $a \in \Sigma$

$S \to [S] \cup [S]$

$S \to [S] \cdot [S]$

$S \to [S]^*$

Satz Sei $L \subset \Sigma^*$. Dann sind die folgenden Aussagen äquivalent:

1. L ist regulär.
2. ex. regulärer Ausdruck α mit $\tau(\alpha) = L$

Damit gilt:
"die regulären Ausdrücke liefern Darstellungen genau der regulären Sprachen".

Beweis:

"2 ⇒ 1": siehe Vorüberlegung

"1 ⇒ 2": "simuliere" endliche Automaten durch reguläre Ausdrücke;

genauer:

beschreibe das Verhalten eines endlichen Automaten bezüglich der "Erreichbarkeiten" von Zuständen durch reguläre Ausdrücke:

sei $DFA = (Q, \Sigma, \delta, q_0, F)$ mit $L = L(DFA)$

o.B.d.A: $Q = \{1,\dots,n\}$ $\quad$ $q_0 = 1$

noch Voraussetzung:

$$L = \{ w \mid w \in \Sigma^* \text{ und } \tilde{\delta}(q_0, w) \in F \}$$

$$= \{ w_1 \dots w_m \mid 1 \xrightarrow{w_1} q_1 \xrightarrow{w_2} \dots \xrightarrow{w_i} q_i \to \dots \xrightarrow{w_m} q_m \in F \}$$

($q_1 \in Q$, $q_i \in Q$)

"Zwischenzustände" (für w)

wir definieren:

$$R^k_{i,j} := \{ w \mid w \in \Sigma^* \text{ und } \tilde{\delta}(i, w) = j$$

und alle "Zwischenzustände liegen in $\{1,\dots,k\} \subset Q$ $\}$

formaler für $w \equiv w_1 \dots w_m$:

f.alle $\ell = 1,\dots,m-1$: $\tilde{\delta}(i, w_1 \dots w_\ell) \in \{1,\dots,k\}$

dann gilt offensichtlich:

$R^n_{1,j}$ ---- alle "Zwischenzustände in $\{1,\dots,n\} = Q$

$$(*) \quad L = \bigcup_{j \in F} R^n_{1,j}$$

Ende: $j \in F$ $\quad$ Start: 1

es reicht zu zeigen:

(xx) alle $R^k_{i,j}$ durch regulären Ausdruck darstellbar

$\rightsquigarrow$ alle $R^n_{1,j}$ mit $j \in F$ "

$\overset{(x)}{\rightsquigarrow}$ L "

[gegeben die $R^n_{1,j}$ werden höchstens $\|Q\| - 1$ zusätzliche Operationszeichen benötigt]

Beweis von (xx) durch Induktion über k:

$k = 0$: also $\{1, \dots, k\} = \emptyset$,

d.h. keine Zwischenzustände!

$\rightsquigarrow$ $w \in R^0_{i,j} \Rightarrow w \in \underbrace{\Sigma}_{= \{z_1, \dots, z_s\}}$ oder $w = \varepsilon$

$\rightsquigarrow$ $R^0_{i,j} \subset \{z_1\} \cup \{z_2\} \cup \dots \cup \{z_s\} \cup \{\varepsilon\}$

$\rightsquigarrow$ $R^0_{i,j}$ durch regulären Ausdruck darstellbar

[es werden höchstens $\|\Sigma\|$ Operationszeichen benötigt]

$k > 0$: wir zeigen:

$$(***) \qquad R^k_{i,j} = R^{k-1}_{i,j} \cup R^{k-1}_{i,k} \cdot \left(R^{k-1}_{k,k}\right)^* \cdot R^{k-1}_{k,j}$$

Jud. Anm.: durch reguläre Ausdrücke darstellbar

↷ $R^k_{i,j}$ ebenso

[gegeben die R^{k-1}'s werden zusätzlich 4 Operationszeichen benötigt]

Bew. von (***):

jeder Weg von i nach j — "zusätzlich erlaubt"

mit Zwischenzuständen in $\{1, \dots, k\}$

lässt sich zerlegen in Teilwege

mit Zwischenzuständen in $\{1, \dots, k-1\}$,

für deren Endpunkte einer der folgenden Fälle vorliegt:

- $i \xrightarrow{*} j$ [k wird nicht benutzt]
- $i \xrightarrow{*} k$ [k wird erstmals benutzt]
- $k \xrightarrow{*} k$ [k wird wiederholt benutzt]
- $k \xrightarrow{*} j$ [k wird letztmals benutzt]

Bemerkungen

- #(Operationen im konstruierten regulären Ausdruck)

$$\leq \underbrace{\|\Sigma\|}_{\text{gewöß "k=0"}} \cdot \underbrace{4^{\|Q\|}}_{\text{gewöß (***)}} + \underbrace{\|Q\|-1}_{\text{gewöß (*)}}$$

#(Induktionsschritte)

- es gibt Beispiele (siehe Wegener) für Sprachen mit "kleinem DFA" und "nur großen reg. Ausdr."

bzw.

"kleinem reg. Ausd." und "nur großen DFA's"

Satz Die Klasse der regulären Sprachen ist abgeschlossen bezüglich Substitution.

(neuer) Beweis:

seien

α ein regulärer Ausdruck für L,

α_i reguläre Ausdrücke für $f(a_i)$

$\rightsquigarrow$ $\alpha\left[{}^{a_i}/_{\alpha_i}\right]$ regulärer Ausdruck für $f(L)$

hier gedeutet als Operator:

ersetze jedes Vorkommen von a_i durch α_i

eine Sprache L "erzeugen":

- durch Turing-Programm TM
  (mit $\sigma(TM)$ total):

$$L = \text{range}(\sigma(TM))$$

- durch Grammatik $G = ((\underbrace{\Sigma, P}_{RW}), \Sigma_V, \Sigma_T, S)$

$$L = L(G)$$
$$= \{ w \mid w \in \Sigma_T^* \text{ und } S \overset{*}{\Rightarrow}_{RW} w \}$$

  - G "operiert" auf einem Wort über $\Sigma = \Sigma_V \cup \Sigma_T$
  - ersetzt jeweils ein Teilwort u durch v,
    mit $u \to v \in P$
  - startet mit S

- durch Post-Systeme:
  - Post-Systeme "operiert" auf Menge von Worten
  - erzeugt jeweils ein weiteres Wort als neues Mengenelement
    (wie in Beweisen: weitere Behauptung aus schon Bewiesenen)
  - started mit Menge von "Axiomen"

Definition • Ein Post-System $PS = (\Sigma_V, \Sigma_C, P, S)$

ist gegeben durch:

Σ_V Alphabet (endliche Menge) von **Variablen**

Σ_C Alphabet (endliche Menge) von **Konstanten**

mit $\Sigma_V \cap \Sigma_C = \emptyset$

$S \underset{\text{endlich}}{\subset} \Sigma_C^*$ endliche Menge von **Axiomen**

$P \underset{\text{endlich}}{\subset} \bigcup_{k \geq 1} (\Sigma^{*k} \times \Sigma^*)$ mit $\Sigma := \Sigma_V \cup \Sigma_C$

endliche Menge von **Regeln**

- eine Regel $(\alpha_1, \ldots, \alpha_k, \alpha_{k+1})$ wird auch geschrieben als $\underbrace{\alpha_1, \ldots, \alpha_k}_{\text{Premises}} \vdash \underbrace{\alpha_{k+1}}_{\text{Conclusion}}$

- $\alpha_1, \ldots, \alpha_k \vdash \alpha_{k+1}$ hat als **konstante Fassung** $\overline{\alpha}_1, \ldots, \overline{\alpha}_k \vdash \overline{\alpha}_{k+1}$, wenn Letzteres aus Ersterem durch **uniforme Ersetzung** der Variablen (aus Σ_V) durch Worte über Σ_C entsteht

- **Semantik:**

$PS = (\Sigma_V, \Sigma_C, P, S)$

bestimmt die **Menge seiner Theoreme** über Σ_C^*

induktiv wie folgt:

(i) jedes Axiom $w \in S$ ist Theorem

(ii) sind $\bar{\alpha}_1, \ldots, \bar{\alpha}_k$ Theoreme
und ist für $\bar{\alpha} \in \Sigma_C^*$
$\bar{\alpha}_1, \ldots, \bar{\alpha}_k \vdash \bar{\alpha}$ eine konstante Fassung
einer Regel $\alpha_1, \ldots, \alpha_k \vdash \alpha$ aus P,

dann ist auch $\bar{\alpha}$ ein Theorem

(iii) "sonst nichts"

Für ein **Terminal-Alphabet** $\Sigma_T \subset \Sigma_C$ sei

$L(PS, \Sigma_T) := \{w \mid w \in \Sigma_T^*, w \text{ ist Theorem von } PS\}$

Beispiel

betrachte: $L = \{ 0^n 1^n \mid n \geq 0 \}$

zu erzeugen: ε

$0\ 1$

$0\ 0\ 1\ 1$

$0\underbrace{0 \ldots 0}_{n-1}\underbrace{1 \ldots 1}_{n-1}1$

schon in der Sprache : Variable A

nächstes Wort : $0\ A\ 1$

als Regel:

$A \vdash 0A1$

Post-System: $PS = (\underbrace{\{A\}}_{\Sigma_V}, \underbrace{\{0,1\}}_{\Sigma_C}, \underbrace{\{A \vdash 0A1\}}_{P}, \underbrace{\{\varepsilon\}}_{S})$

Beh: $L = L(PS, \underbrace{\{0,1\}}_{\Sigma_T})$

Beweis:

"$\subseteq$": (Induktion über n):

$n=0$: ε ist Axiom $\rightsquigarrow$ $\varepsilon \in L(PS, \{0,1\})$

$n+1$: • Ind. Annahme: $0^n 1^n \in L(PS, \{0,1\})$,
insbesondere Theorem

• $0^n 1^n \vdash 0\,0^n 1^n\,1$ ist konstante Fassung von

$A \vdash 0\,A\,1$ "uniform ersetzt"

$\in P$

$\rightsquigarrow 0^{n+1} 1^{n+1}$ Theorem und damit Element von $L(PS, \{0,1\})$

"$\supseteq$": (Induktion über die Länge ℓ von Ableitungen:

Ind.: mit Ableitungen der Länge ℓ ist nur $0^\ell 1^\ell$ ableitbar

Bew: $\ell = 0$: ε ist einziges Axiom

$\ell+1$: • Ind. Annahme: $0^\ell 1^\ell$ sei einziges (neues) ableitbares Wort für ℓ

• einzige Regel $A \vdash 0A1$ bzw. einzige Variable A

• uniforme Ersetzung von A durch $0^\ell 1^\ell$ liefert $0^{\ell+1} 1^{\ell+1}$

Typ-0 Grammatiken als Post-Systeme gedeutet

Sei $G = (\underbrace{(\Sigma, P)}_{RW}, \Sigma_N, \Sigma_T, S)$ Typ-0 Grammatik

erzeugte Sprache:

$$L(G) := \{ w \mid w \in \Sigma_T^* \text{ und } S \overset{*}{\Rightarrow}_{RW} w \} \subset \Sigma_T^*$$

durch Post-System "deuten":

$$\Sigma_C := \Sigma = \Sigma_N \cup \Sigma_T$$

$$\Sigma_V := \{ \alpha_{Anfang}, \alpha_{Ende} \}$$

$$P\_Post := \{ \alpha u \alpha \vdash \alpha v \alpha \mid u \to v \in P \}$$

$$S\_Post := \{ S \}$$

$$PS := (\Sigma_V, \Sigma_C, P\_Post, S\_Post)$$

Beh.: $S \overset{*}{\Rightarrow}_{RW} w$ gdw w ist Theorem von PS

Beweisidee: PS beschreibt genau, was durch $S \overset{*}{\Rightarrow}_{RW} \ldots$ definiert ist

"$\Rightarrow$" (Induktion über Länge der Ableitung von w bzg. RW)

Länge = 0: $\rightsquigarrow$ $w = S$ wegen $S \overset{*}{\underset{RW}{\Rightarrow}} S$

gemäß "reflexivem Abschluss"

$S\_Post = \{S\}$ $\rightsquigarrow$ w ist Axiom von PS

Def. "Theorem" $\rightsquigarrow$ w ist Theorem von PS

Länge > 0: sei $S \overset{*}{\underset{RW}{\Rightarrow}} \bar{w} \underset{RW}{\Rightarrow} w$

Ind. Annahme: $\bar{w}$ Theorem von PS (1)

$\bar{w} \underset{RW}{\Rightarrow} w$: ex. $u_1, u_2 \in \Sigma^* = \Sigma_C^*$,

ex. $u \to v \in P$:

$\bar{w} = u_1 u u_2$

$w = u_1 v u_2$

$\rightsquigarrow$ $\bar{w} \vdash w$ ist konstante Fassung von $\alpha u \varepsilon \vdash \alpha v \varepsilon$ aus P_Post (2)

(1) $\wedge$ (2): w ist Theorem von PS

"$\Leftarrow$": analog

eine Grammatik für $\{0^n 1^n \mid n \geq 0\}$:

$\Sigma_T := \{0, 1\}$

$\Sigma_N := \{S\}$

$\rightsquigarrow \quad \Sigma = \Sigma_T \cup \Sigma_N = \{0, 1, S\}$

$P := \{ S \to 0S1 \;,\; S \to \varepsilon \}$

"deutendes" Post-System:

$\Sigma_C := \{0, 1, S\}$

$\Sigma_V := \{\alpha, \varepsilon\}$

$P\_Post := \{ \alpha S \varepsilon \vdash \alpha 0 S 1 \varepsilon,$
$\alpha S \varepsilon \vdash \alpha \varepsilon \}$

$S\_Post := \{S\}$

Grammatik		Post-System		
$\underline{S}$		S		Axiom
$\Downarrow$	$S \to 0S1$	$S \vdash 0S1$	konkrete Fassung:	$\alpha/\varepsilon, \varepsilon/\varepsilon$
$0\underline{S}1$		$0S1$		Theorem
$\Downarrow$	$S \to 0S1$	$0S1 \vdash 00S11$	konkrete Fassung:	$\alpha/0, \varepsilon/1$
$00\underline{S}11$		$00S11$		Theorem
$\Downarrow$	$S \to \varepsilon$	$00S11 \vdash 0011$	konkrete Fassung:	$\alpha/00, \varepsilon/11$
0011		0011		Theorem

Satz Sei $L \subset \Sigma_T^*$ eine Sprache.

Dann sind folgende Aussagen äquivalent:

(1) L ist rekursiv aufzählbar

(2) ex. Post-System PS mit

$$L = L(PS, \Sigma_T)$$

Beweis:

(1) $\Rightarrow$ (2): • L rekursiv aufzählbar

$\rightsquigarrow$ ex. Grammatik G mit $L(G) = L$

• deute G als Post-System PS,

d.h. $S \overset{*}{\underset{\text{alw}}{\Rightarrow}} w$ gdw. w Theorem von PS

$\rightsquigarrow L = L(PS, \Sigma_T)$

(2) $\Rightarrow$ (1): alle Theoreme von einem PS lassen sich "algorithmisch aufzählen"

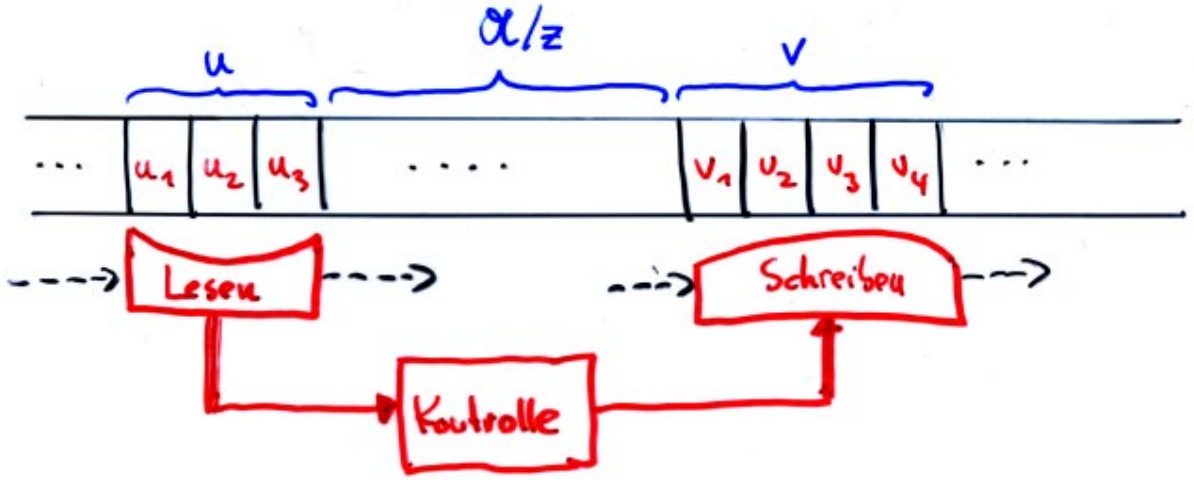
(Churchsche These anwenden)

Normalform - Ergebnisse

- im allgemeinen: beliebig viele Variablen (nur endlich)
- Deutung von Grammatiken:
  - 2 Variablen reichen
  - nur Regeln der Form $\alpha u \Sigma \vdash \alpha v \Sigma$
- man kann zeigen:
  - 1 Variable reicht
  - nur Regeln der Form $u\alpha \vdash \alpha v$ mit $\{\alpha\} = \Sigma_V$, $u, v \in \Sigma_C^*$

"Maschinen-Deutung":

- bei konstanter Fassung: α/z
- "links den Kopf u von uz (zerstörend) lesen"
- "rechts neues Ende v hinter z schreiben"



- "Lese-Kopf" so groß wie maximal langes u in Regeln
- "Schreib-Kopf" so groß wie maximal langes v in Regeln
- durch TM simulierbar:
  - Löschend lesen und im Zustand merken
  - ans rechte Ende laufen
  - schreiben
  - zurück an linkes Ende laufen

Beweisidee für Normalform:

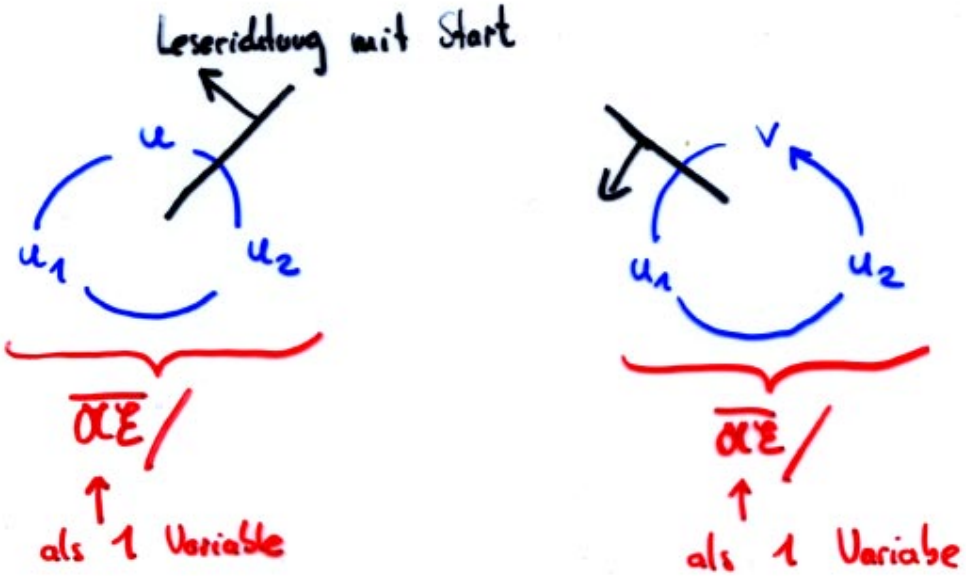
Grammatik deuten:

$u \to v \in P$

$u_1 \; u \; u_2 \quad \Longrightarrow \quad u_1 \; v \; u_2$

$\alpha / \qquad \xi / \qquad\qquad \alpha / \qquad \xi /$

anders schreiben (grobe Veranschaulichung):



$u\,\overline{\alpha\xi} \quad \vdash \quad \overline{\alpha\xi}\,v$

hat gewünschte Form

"Start" versetzen:

betrachte $\alpha u \mathcal{E} \vdash \alpha v \mathcal{E}$

mit $u, v \in \Sigma_c^*$

Kopie von Σ_c anlegen: $\overline{\Sigma}_c := \{ \bar{x} \mid x \in \Sigma_c \}$

dann simulieren mit neuen Regeln:

(1) $u \, \alpha \mathcal{E} \longrightarrow \alpha \mathcal{E} \, \bar{v}$ mit $\overline{v_1 \dots v_n} := \bar{v}_1 \dots \bar{v}_n$

(2) $x \, \alpha \mathcal{E} \longrightarrow \alpha \mathcal{E} \, \bar{x}$

(3) $\bar{x} \, \alpha \mathcal{E} \longrightarrow \alpha \mathcal{E} \, x$

typischer Ablauf:

$u_1 \, u \, u_2$

$\downarrow^* (2)$

$u \, u_2 \, \bar{u}_1$

$\downarrow (1)$

$u_2 \, \bar{u}_1 \, \bar{v}$

$\downarrow^* (2)$

$\bar{u}_1 \, \bar{v} \, \bar{u}_2$

$\downarrow^* (3)$

$\bar{v} \, \bar{u}_2 \, u_1$

$\downarrow^* (3)$

$\bar{u}_2 \, u_1 \, v$

$\downarrow^* (3)$

$u_1 \, v \, u_2$

einige Literaturhinweise:

A. I. Malcev: Algorithmen und rekursive Funktionen,
Vieweg, 1974
(Übersetzung aus dem Russischen, 1960)

Kapitel VI:
Varianten der Maschinen und Algorithmen
von Turing und Post

W. S. Brainerd / L. H. Landweber: Theory of Computation,
Wiley, 1974

Chapter 7: Formal Languages

Egon Börger: Berechenbarkeit, Komplexität, Logik
Vieweg 1992 (3. Auflage)

Kapitel A: Mathematischer Algorithmusbegriff
Teil I: Churchsche These
§ 3: Erweiterter Äquivalenzsatz

Kontextfreie Sprachen

- erzeugbar mit kontextfreien Grammatiken der Art

$$G = (\overbrace{(\Sigma, P)}^{RW}, \Sigma_N, \Sigma_T, S),$$

alle Produktionen aus P haben Form

$A \to w$ mit $A \in \Sigma_N$, $w \in (\Sigma_N \cup \Sigma_T)^* = \Sigma^*$

(behauptete) Normalform: $A \to BC$ mit $A, B, C \in \Sigma_N$

$A \to a$ $a \in \Sigma_T$

$(A \to \varepsilon)$

$$L(G) = \{ w \mid w \in \Sigma_T^* \text{ und } S \overset{*}{\Rightarrow}_{RW} w \}$$

- Beispiel "einfache Klammerungen":

$S \to 01$
$S \to 0S1$

$\rightsquigarrow L(G) = \{ 0^n 1^n \mid n \geq 1 \}$

- Beispiel "Palindrome":

$S \to 0$
$S \to 1$
$S \to 0S0$
$S \to 1S1$
$S \to \varepsilon$

$\rightsquigarrow L(G) = \{ w \mid w \in \{0,1\}^* \text{ und } w = w^R \}$

- **Beispiel "GLEICH viele Nullen und Einsen":**

$S \to 0B$ erzeugt 0; "merkt" B (noch eine 1 erzeugen!)

$S \to 1A$ erzeugt 1; "merkt" A (noch eine 0 erzeugen!)

$A \to 0$ erzeugt "gemerkte" 0

$A \to 0S$ erzeugt "gemerkte" 0 und iteriert

$A \to 1AA$ erzeugt 1; "merkt" AA (noch zwei 0 erzeugen!)

$B \to 1$

$B \to 1S$ (analog)

$B \to 0BB$

Beh:

(i) $S \overset{*}{\Rightarrow} w$ gdw w hat gleich viele Nullen und Einsen $\equiv$: GLEICH(w)

(ii) $A \overset{*}{\Rightarrow} w$ gdw w hat eine Null mehr als Einsen $\equiv$: NULL(w)

(iii) $B \overset{*}{\Rightarrow} w$ gdw w hat eine Eins mehr als Nullen $\equiv$: EINS(w)

Plausibilität: P ist mit Behauptungen verträglich

Beweis • jede Regelanwendung erzeugt genau ein Terminalzeichen

↝ erzeugtes Teilwort der Länge k durch k Regelanwendungen entstanden

• Induktion über k:

$k = 1$:

einerseits	andererseits
• aus S kein Wort der Länge 1 erzeugbar	kein Wort mit GLEICH(w)
• aus A nur 0 erzeugbar	0 einziges Wort mit NULL(w)
• aus B nur 1 erzeugbar	1 einziges Wort mit EINS(w)

$k > 1$:

"$\Rightarrow$":

zu (i): sei $S \underbrace{\Rightarrow \dots \Rightarrow}_{k\text{-mal}} w = w_1 \bar{w}$ mit $w_1 \in \{0, 1\}$

Fall 1: $w_1 = 0$

Def. P/hfr: $S \Rightarrow 0B \Rightarrow \dots \Rightarrow 0\bar{w}$

mit $B \underbrace{\Rightarrow \dots \Rightarrow}_{(k-1)\text{-mal}} \bar{w}$

Ind. Annahme: EINS($\bar{w}$)

$\rightarrow$ GLEICH($0\bar{w}$), $0\bar{w} = w$

Fall 2: $w_1 = 1$: analog

zu (ii): sei $A \underbrace{\Rightarrow \dots \Rightarrow}_{k\text{-mal}} w = w_1 \bar{w}$ mit $w_1 \in \{0,1\}$

Fall 1: $w_1 = 0$

Def. P/kfr ↘ $A \Rightarrow 0S \Rightarrow \dots \Rightarrow 0\bar{w}$

mit $S \underbrace{\Rightarrow \dots \Rightarrow}_{(k-1)\text{-mal}} \bar{w}$

Ind. Annahme ↘ GLEICH($\bar{w}$)

↘ NULL($\underbrace{0\bar{w}}_{=w}$)

Fall 2: $w_1 = 1$

Def. P/kfr ↘ $A \Rightarrow 1AA \Rightarrow \dots \Rightarrow 1\bar{w}_a\bar{w}_b$

mit $A \dots \Rightarrow \dots \bar{w}_a$

$A \dots \Rightarrow \dots \bar{w}_b$

Ind. Annahme ↘ NULL($\bar{w}_a$) und NULL($\bar{w}_b$)

↘ NULL($\underbrace{1\bar{w}_a\bar{w}_b}_{=w}$)

zu (iii): analog

„$\Leftarrow$“:

zu (i): sei GLEICH(w)

Fall 1: $w = 0\bar{w}$

Def. GLEICH/EINS ↷ EINS($\bar{w}$)

Ind. Annahme ↷ $B \overset{*}{\Rightarrow} \bar{w}$

Def. P/kfr ↷ $S \Rightarrow 0B \overset{*}{\Rightarrow} \underbrace{0\bar{w}}_{=w}$

Fall 2: $w = 1\bar{w}$: analog

zu (ii): sei NULL(w)

Fall 1: $w = 0\bar{w}$

Def NULL/GLEICH ↷ GLEICH($\bar{w}$)

Ind. Annahme ↷ $S \overset{*}{\Rightarrow} \bar{w}$

Def P/kfr ↷ $A \Rightarrow 0S \overset{*}{\Rightarrow} \underbrace{0\bar{w}}_{=w}$

Syntaxbaum

- graphische Darstellung von Ableitungen
- abstrahiert bei Wahlmöglichkeit von Reihenfolge der Ersetzungen
- Wurzel: mit Startsymbol markiert
- falls Knoten mit Nichtterminal A markiert und (dieses Vorkommen von) A wird vermöge Produktion $A \to w_1 \dots w_r$ ersetzt,

  dann r Nachfolgerknoten, mit w_1 bzw. w_2 ... bzw. w_r markiert

  [speziell für $A \to \varepsilon$: ein Nachfolgerknoten, mit ε markiert]
- falls Knoten mit Terminal oder ε markiert,

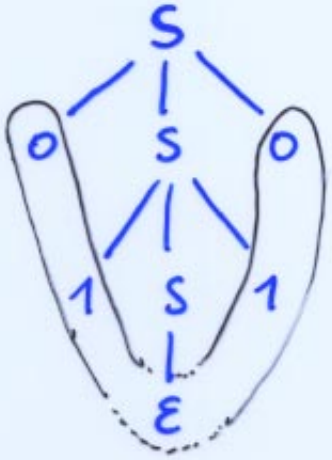
  dann Blatt

Beispiele:

- Palindrome:

$S \Rightarrow 0S0 \Rightarrow 01S10 \Rightarrow 0110$

$S \to 0S0$ $S \to 1S1$ $S \to \varepsilon$



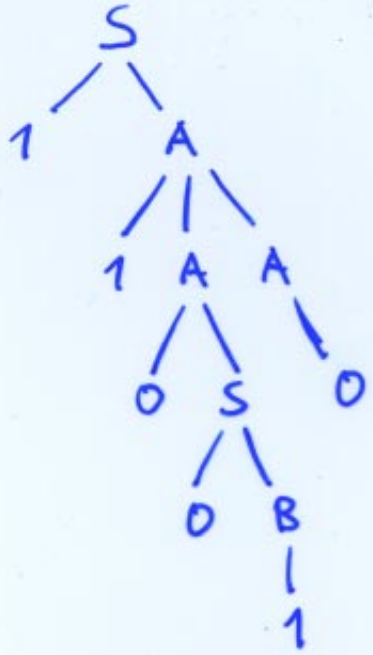
- GLEICH:

$S \Rightarrow 1A \Rightarrow 11AA \Rightarrow 11A0 \Rightarrow 110S0$

$S \to 1A$ $A \to 1AA$ $A \to 0$ $A \to 0S$

$\Rightarrow 1100B0 \Rightarrow 11001 0$

$S \to 0B$ $B \to 1$



Beobachtungen

(1) zu jeder Ableitung genau ein Syntaxbaum

(2) ein Syntaxbaum kann mehrere Ableitungen darstellen, die sich nur in der Reihenfolge der Ersetzungen unterscheiden

(3) stets "Normierungen" möglich : Linksableitung, Rechtsableitung, ...

eine Linksableitung mit gleichem Syntaxbaum:

$$S \Rightarrow \underset{S\to 1A}{1\,\underline{A}} \Rightarrow \underset{S\to 1AA}{1\,1\,\underline{A}\,A} \Rightarrow \underset{A\to 0S}{1\,1\,0\,\underline{S}\,A} \Rightarrow \underset{S\to 0B}{1\,1\,0\,0\,\underline{B}\,A} \Rightarrow \underset{B\to 1}{1\,1\,0\,0\,1\,\underline{A}} \Rightarrow \underset{A\to 0}{1\,1\,0\,0\,1\,0}$$

Definition

- kfr Grammatik G heißt **eindeutig** : gdw

  f. alle $w \in L(G)$ gibt es **genau einen** Syntaxbaum

- kfr Sprache L heißt **eindeutig** : gdw

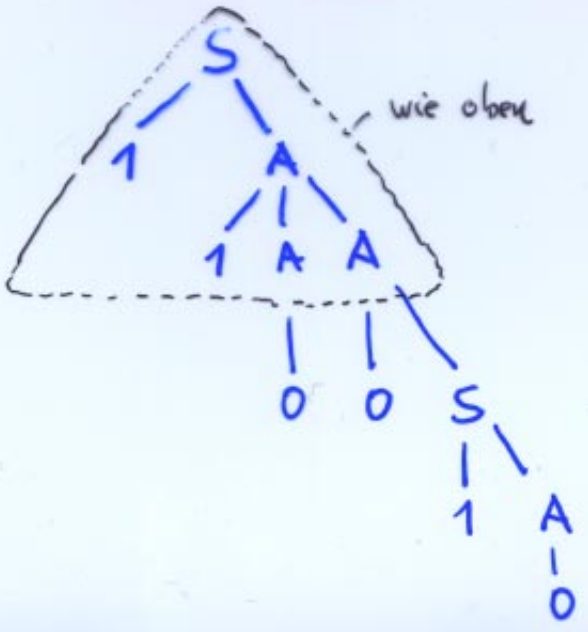
  ex. eindeutige kfr Grammatik G mit $L = L(G)$

  (sonst inhärent mehrdeutig)

Bsp:
- einfache Klammerung: } eindeutige Grammatiken
- Palindrome : } (stets nur ein Nichtterminal)

↷ eindeutige Sprachen

- GLEICH : Grammatik **nicht** eindeutig, z. B.:



kfr allgemein: $A \to w$ mit $A \in \Sigma_N$, $w \in (\Sigma_N \cup \Sigma_T)^*$

Chomsky-Normalform: $A \to BC$

$A \to a$

($A \to \varepsilon$; genauer: falls $\varepsilon \in L(G)$

dann nur $S' \to S$, $S' \to \varepsilon$ • neues Startsymbol, • kommt sonst nicht vor)

- im Folgenden: $\varepsilon \notin L(G)$
- "verbotene" Produktionen können durch "erlaubte" simuliert werden:

(i) rechts "gemischt"

(wie allgemein:)
für jedes Terminal x
ein neues Nichtterminal A_x

- ersetze x durch A_x
- $A_x \to x$ neue Produktion

(ii) $A \to B_1 \dots B_m$ mit $m \geq 3$

(wie allgemein:) bilde Ketten
mit neuen Nichtterminal $C_1, \dots, C_{m-2}$
vermöge neuer Produktionen

$A \to B_1 C_1$
$C_1 \to B_2 C_2$
$\dots$
$C_{m-2} \to B_{m-1} B_m$

(iii) $A \to \varepsilon$
(ε-Produktionen)

bestimme $EPS := \{A \mid A \Rightarrow^* \varepsilon, A \in \Sigma_N\}$

- streiche alle ε-Produktionen
- für $A \to BC$ mit

  $B \in EPS$ bzw. $C \in EPS$

  neue zusätzliche Produktionen:

  $A \to C$ bzw. $A \to B$

(iv) $A \to B$
(Ketten-Produktionen)

betrachte durch Ketten-Produktionen gebildeten Graphen:

- <u>falls</u> Kreis $A_1 \to A_2 \to \dots \to A_r \to A_1$

  [alles austauschbar]

  <u>dann</u> ersetze $A_2, \dots, A_r$ durch A_1

  ["alles zusammenziehen"]

- sortiere kreisfreien Restgraphen topologisch mit

  $A_i \to A_j \curvearrowright i < j$

- für verbliebene $A_1, \dots, A_m$

  Kettenregeln induktiv entfernen

  für $k = m, \dots, 1$ (absteigend):

falls $A_k \to A_\ell$ Kettenregel

dann $\ell > k$ (topologisch sortiert)

und von A_ℓ keine Kettenregeln mehr

(gemäß Ind. Annahme):

ersetze $A_k \to A_\ell$

durch $A_k \to \alpha$

für alle $A_\ell \to \alpha$

Zeitbedarf, für $s(G) := \#(\text{Zeichenvorkommen in Regeln})$:

$O(s(G)^2)$

Größe der normierten Grammatik: $O(s(G)^2)$

Wortproblem für kfr Grammatiken in Chomsky-Normalform

entscheide: $w \in L(G)$?

"brutale Methode":

[benutzt, dass alle "Struktur-Ableitungen" echt verlängernd sind, vermöge $A \to BC$]

für $i = 0, 1, \ldots, |w| - 1$:

- bilde alle Struktur-Ableitungen der Länge i
  [erzeugte Nichtterminalworte haben Länge $i+1$, jeweils ein zusätzliches Nichtterminal]
- bilde jeweils alle weitergehenden Terminal-Ableitungen
- falls w erscheint: ja!
  sonst: nein!

- nicht zielgerichtet, zu aufwendig!
- Cocke-Younger-Kasami-Algorithmus: Laufzeit $O(|w|^3 \cdot \|P\|)$:

CYK-Algorithmus für Wortproblem

Parameter: kfr Grammatik $G = ((\Sigma, P), \Sigma_N, \Sigma_T, S)$

in Chomsky-Normalform

Eingabe: $w = w_1 \dots w_n$

Idee:
- f. alle $1 \leq i \leq j \leq n$ bestimme

$$V_{i,j} := \{ A \mid A \overset{*}{\Rightarrow} \underbrace{w_i \dots w_j}_{\text{Teilwort der Eingabe}} \}$$

- $w \in L(G)$ gdw $S \in \underbrace{V_{1,n}}_{\text{gesamte Eingabe}}$

Methode: für $\ell = 0, 1, \dots n-1$:

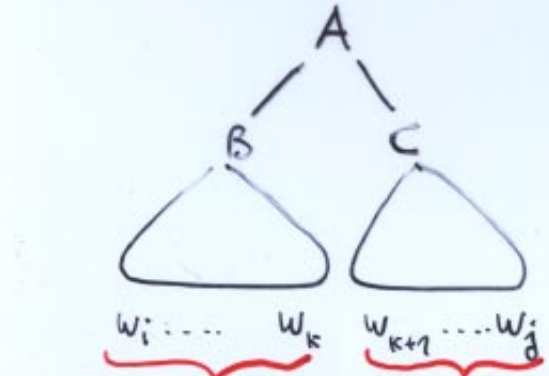
für i, j mit $\ell = j - i$:

falls $\ell = 0$: $[\rightsquigarrow i = j]$ $\quad V_{i,i} := \{ A \mid A \to w_i \in P \}$

↑ Chomsky-Normalform

falls $\ell > 0$: $[\rightsquigarrow i < j]$

Chomsky-NF $\rightsquigarrow$ eine Ableitung $A \overset{*}{\Rightarrow} w_i\, w_{i+1} \dots w_j$ hat Form



für ein $k \in \{i, \dots, j-1\}$

$\rightsquigarrow B \in V_{i,k}$ $\quad \rightsquigarrow C \in V_{k+1,j}$

$\rightsquigarrow V_{i,j} := \{ A \mid \text{ex. } B, C \in \Sigma_N ,$ ① ①

$\text{ex. } k \in \{i, \dots, j-1\}:$ ②

(i) $A \to BC \in P$ ③

(ii) $B \in V_{i,k}$

(iii) $C \in V_{k+1,j} \}$ ④

schon bekannt!

① $\leq |w|^2$ viele

② $\leq |w|$ viele

③ $\leq \|P\|$ viele

④ "table lookup"

$\rightsquigarrow$ Laufzeit $\mathcal{O}(|w|^3 \cdot \|P\|)$

Pumping Lemma für kontextfreie Sprachen

f. alle kfr. Sprachen $L \subset \Sigma_T^*$

ex. $n \in \mathbb{N}$:

f. alle Worte $z \in L$ mit $|z| \geq n$

ex. $u, v, w, x, y \in \Sigma_T^*$:

(i) $z = u\,v\,w\,x\,y$

(ii) $|v\,x| \geq 1$ [d.h. $v \neq \varepsilon$ oder $x \neq \varepsilon$]

(iii) $|v\,w\,x| \leq n$ ["zusammenhängende Pumpstelle"]

(iv) f. alle $i \geq 0$: $u\,v^i\,w\,x^i\,y \in L$

Ogden's Lemma

f. alle kfr. Sprachen $L \subset \Sigma_T^*$

ex. $n \in \mathbb{N}$:

f. alle Worte $z \in L$ mit $|z| \geq n$

f. alle "Markierungen" von mindestens n Zeichen in z

ex. $u, v, w, x, y \in \Sigma_T^*$:

(i) $z = u\,v\,w\,x\,y$

(ii) $v\,x$ enthält mindestens ein markiertes Zeichen

(iii) $v\,w\,x$ enthält höchstens n markierte Zeichen

(iv) f. alle $i \geq 0$: $u\,v^i\,w\,x^i\,y \in L$

Beweis Pumping Lemma

- sei $G = ((\Sigma, P), \Sigma_N, \Sigma_T, S)$ kfr. Grammatik in Chomsky-Normalform mit Produktionen der Form

  $A \to BC$ [Struktur]

  $A \to a$ [Beispiel]

  (o.B.d.A. keine ε-Produktion)

  und

  $L = L(G)$

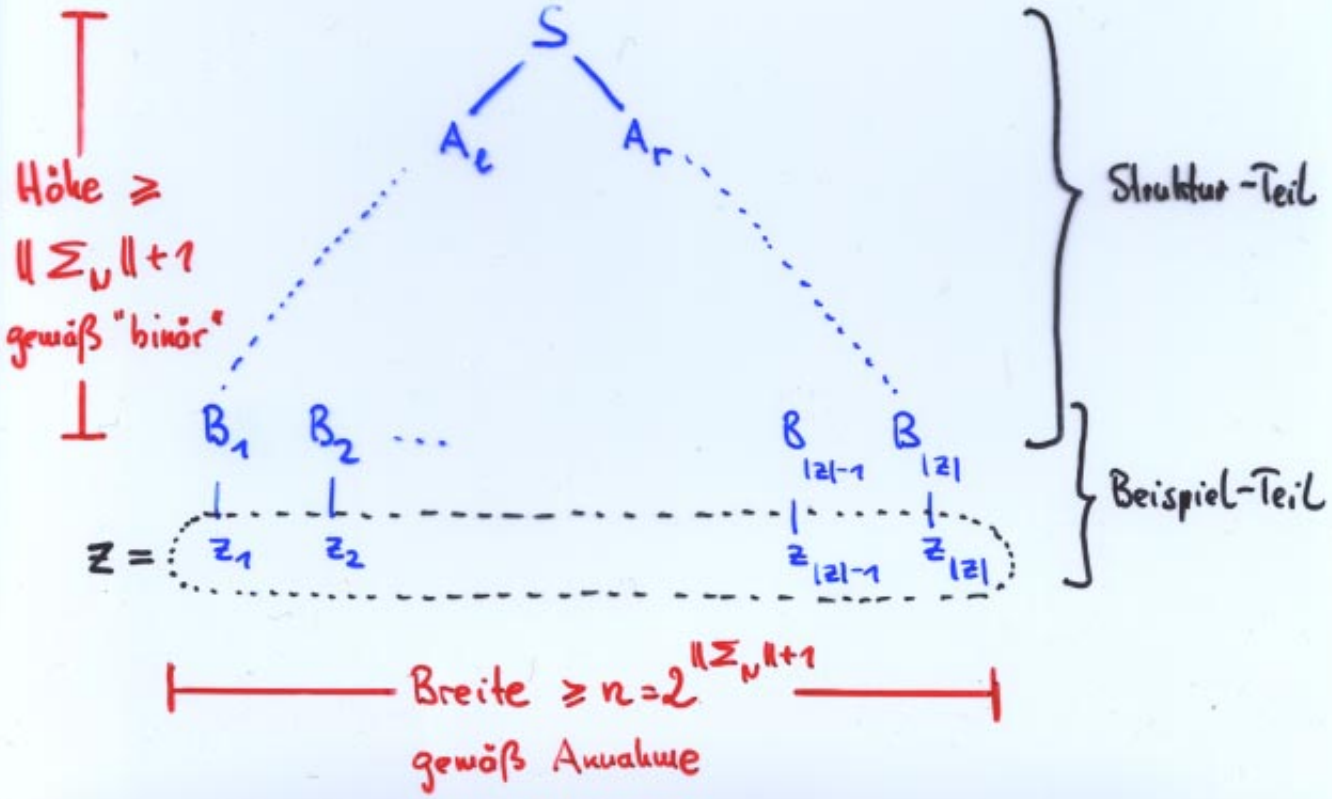
- definiere $n := 2^{\|\Sigma_N\|+1}$

- sei $z \in L$ mit $|z| \geq n$,

  betrachte Syntaxbaum für $S \overset{*}{\Rightarrow} z$:

  Struktur-Teil: binär

  Beispiel-Teil: unär



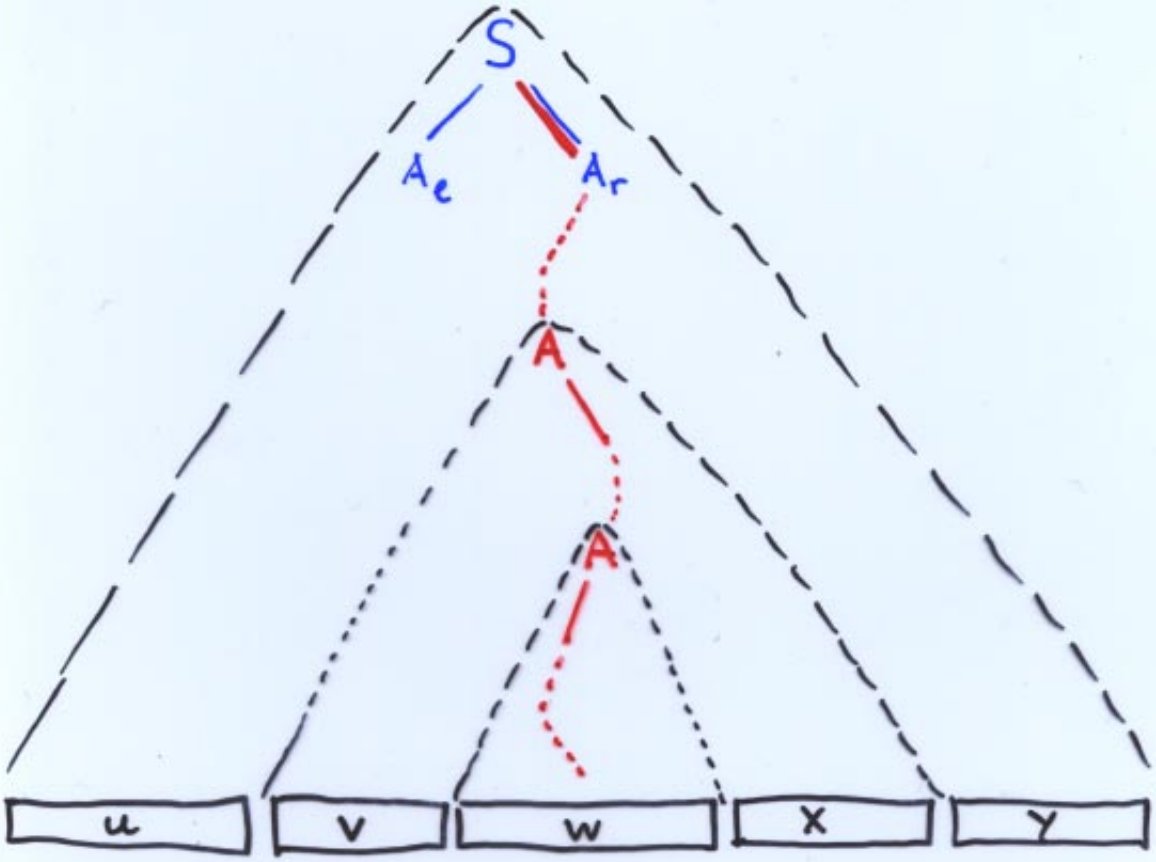
↝ im Struktur-Teil liegt ein Pfad

$$S =: A_0 \to A_1 \to \dots \to A_t \quad [= B_k \text{ für ein } k]$$

von S zu einem "Struktur-Blatt" A_t

der Länge $t \geq \| \Sigma_N \| + 1$

↝ Schubfach-Prinzip ex $i < j$ mit $A_i = A_j =: A$, z.B.



↝ $S \overset{*}{\Rightarrow} u\,A\,y$ mit $A \overset{*}{\Rightarrow} v\,A\,x$

$\overset{*}{\Rightarrow} u\,v\,A\,x\,y$ mit $A \overset{*}{\Rightarrow} w$

} ↝ $A \overset{*}{\Rightarrow} v^i\,w\,x^i$ f.alle $i \geq 0$

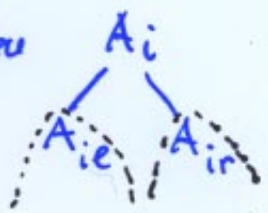
↝ $S \overset{*}{\Rightarrow} u\,v^i\,w\,x^i\,y$ f.alle $i \geq 0$

Beweis Ogden's Lemma

- analog wie Beweis Pumping Lemma
- spezielle Wahl des Pfades im Struktur-Teil:

  $S =: A_0 \to A_1 \to \dots \to A_t$

  derart, dass für A_i mit Nachfolgerknoten

  ein Pfad-Nachfolger A_{i+1} so gewählt wird, dass

  dessen Teilbaum mindestens so viele markierte Zeichen enthält wie der andere Teilbaum;

  "Verzweigungsknoten": solche A_i, so dass beide Teilbäume markierte Zeichen enthalten

  ↝ Pfad enthält mindestens $\|\Sigma_V\|+1$ Verzweigungsknoten

  [wegen mindestens $2^{\|\Sigma_V\|+1}$ markierte Blätter]

- dann analog weiter

Anwendung als "notwendige Bedingung" für kontextfrei:

Beh: $L = \{ a^i b^i c^i \mid i \geq 1 \}$
ist **nicht** kontextfrei.

Beweis: (indirekt)

angenommen L kfr

↝ für n aus Pumping Lemma
betrachte $a^n b^n c^n$

↝ (Pumping Lemma) ex. Zerlegung $a^n b^n c^n = uvwxy$ mit

- $uv^2wx^2y \in L$
- $|vx| \geq 1$ ↝ $vx \neq v^2x^2$
- $|vwx| \leq n$ ↝ in vwx kommen höchstens 2 der 3 Zeichen a, b, c vor

↝ in uv^2wx^2y kommt
eines der 3 Zeichen weniger als ein anderes vor

↝ ↯ (Def. L)

Chomsky-Hierarchie:

$$\mathcal{L}_i := \{ L \mid \text{ex. Typ-}i\text{ Grammatik } G \text{ mit } L = L(G) \}$$

für $i = 0$: rekursiv aufzählbar
$i = 1$: kontext-sensitiv
$i = 2$: kontextfrei
$i = 3$: regulär

Satz Die Chomsky-Hierarchie ist echt, d.h.

$$\mathcal{L}_3 \subsetneq \mathcal{L}_2 \subsetneq \mathcal{L}_1 \subsetneq \mathcal{L}_0$$

regulär — kfr — kontext-sensitiv — rek.aufz.

Beweisskizze

(a) "$\subseteq$" folgt aus Definitionen

(b) vermöge "Gegenbeispielen":

- $\{0^i 1^i \mid i \geq 1\}$ kfr, aber nicht regulär
- $\{a^i b^i c^i \mid i \geq 1\}$ kontext-sensitiv, aber nicht kfr

  Beweis A: konstruiere Typ-1 Grammatik
  Beweis B: "Komplexitäts-Argument"
- "Diagonalsprache" D rek.aufz., aber nicht kontext-sensitiv

Beweisidee: kontext-sensitiv in Normalform

↷ Ableitungen "verlängernd"

↷ Wortproblem entscheidbar

aber: D unentscheidbar
(nicht rekursiv)

Abschlusseigenschaften der Klasse der kfr Sprachen

"effektiv" abgeschlossen unter:

- Vereinigung $\cup$
- Konkatenation $\cdot$
- Kleenescher Abschluss *
- Substitution $f : \Sigma_T \to \mathcal{P}\Delta^*$

nicht abgeschlossen unter:

- Durchschnitt
- Komplement

"abgeschlossen"

Konstruiere (berechne) zu gegebenen kfr Grammatik eine kfr Grammatik für Ergebnis:

Vereinigung: $L(G_1) \cup L(G_2) = L(G)$ mit

$$\Sigma_N := \Sigma_{N_1} \cup \Sigma_{N_2} \cup \{S\}$$

(Σ_{N_1}: ggf. umbenennen; S: neues Startsymbol)

$$P := P_1 \cup P_2 \cup \{ S \to S_1, \; S \to S_2 \}$$

($S \to S_1$: erster Schritt "alternativ")

Konkatenation: $L(G_1) \cdot L(G_2) = L(G)$ mit

$$\Sigma_N := \Sigma_{N_1} \cup \Sigma_{N_2} \cup \{S\}$$

$$P := P_1 \cup P_2 \cup \{ S \to S_1 S_2 \}$$

bilde Konkatenation der Startsymbole

Kleenescher Abschluss: $(L(G_1))^* = L(G)$ mit

$$\Sigma_N := \Sigma_{N_1} \cup \{S\}$$

$$P := P_1 \cup \{ S \to \varepsilon, \; \underbrace{S \to SS}_{\text{Iteration}}, \; S \to S_1 \}$$

ggf. "leer"

Substitution $f(L(G_1))$ mit

$f(a) = L(G_a)$ für $a \in \Sigma_T$ mit

$$\Sigma_N := \Sigma_{N_1} \cup \dot{\bigcup_{a \in \Sigma_T}} \Sigma_{N_a}$$

ggf. umbenennen

$$P := P_1[{}^{a}/_{S_a}] \cup \bigcup_{a \in \Sigma_T} P_a$$

liefert $\{ S_{a_1} S_{a_2} \dots S_{a_k} \mid a_1 a_2 \dots a_k \in L(G_1) \}$

d.h. $S_1 \overset{*}{\Rightarrow}_{P_1} \underbrace{a_1 \dots a_k}_{\in \Sigma_T^*}$

gdw $S_1 \overset{*}{\Rightarrow}_{P_1[{}^{a}/_{S_a}]} S_{a_1} \dots S_{a_k}$

ferner $S_a \overset{*}{\Rightarrow}_{P_a} \underbrace{w}_{\in \Delta^*}$ gdw $w \in f(a)$

also: Ableitungen mit $P_1[{}^{a}/_{S_a}]$ vermöge der P_a wie gewünscht fortsetzbar

nicht abgeschlossen:

Durchschnitt : Gegenbeispiel:

- $\{a^n b^n \mid n \geq 1\}$ kfr
  $\{c\}^*$ kfr
  $\rightsquigarrow \{a^n b^n c^m \mid n \geq 1, m \geq 0\}$
  $= \{a^n b^n \mid n \geq 1\} \cdot \{c\}^*$ kfr

- analog: $\{a^m b^n c^n \mid n \geq 1, m \geq 0\}$ kfr

- aber: $\{a^n b^n c^m \mid n \geq 1, m \geq 0\}$
  $\cap \{a^m b^n c^n \mid n \geq 1, m \geq 0\}$
  $= \{a^n b^n c^n \mid n \geq 1\}$ nicht kfr
  (siehe Pumping Lemma)

- **Komplement**: (indirekt)

  angenommen: abgeschlossen unter Komplement
  siehe oben: abgeschlossen unter Vereinigung
  abgeschlossen unter Durchschnitt ↯

de Morgan: $L_1 \cap L_2 = C(CL_1 \cup CL_2)$

Entscheidungsprobleme für kontextfreie Sprachen

entscheidbar:

- **Wortproblem:** $w \in L(G)$

  ↷ CYK-Entscheidungsverfahren

- **Leerheit:** $L(G) = \emptyset$

  betrachte G kfr Grammatik in Normalform

  mit Pumping Lemma Konstante $n = 2^{|\Sigma_N|+1}$

  $L(G) \neq \emptyset$

  ↷ ex. $w \in L(G)$ mit: $|w| < n$ oder $|w| \geq n$

  dann enthält $L(G)$ auch kürzeres Wort

  Pumping Lemma: verkürzend

  ex. $w \in L(G)$ mit $|w| < n$

  entscheidbar durch systematisches Erzeugen aller Ableitungen für Worte der Länge $< n$

- **Unendlichkeit:** $L(G)$ unendlich

  ex. $w \in L(G)$ mit $|w| \geq n$

  Pumping Lemma { verlängernd, verkürzend }

  ex $w \in L(G)$ mit $n+n \geq |w| \geq n$

nicht entscheidbar (Beweise : siehe Wegener)

- Totalität : $L(G) = \Sigma_T^*$
- Äquivalenz : $L(G_1) = L(G_2)$
- Enthaltensein : $L(G_1) \subset L(G_2)$
- Nichtleerer-Durchschnitt: $L(G_1) \cap L(G_2) \neq \emptyset$
- Regularität : $L(G)$ regulär
- Kontextfreier-Durchschnitt: $L(G_1) \cap L(G_2)$ kfr
- Kontextfreies-Komplement: $\complement L(G)$ kfr
- Mehrdeutigkeit : G mehrdeutig
- Inhärente-Mehrdeutigkeit: $L(G)$ inhärent mehrdeutig

erzeugen	vs	erkennen
mit Grammatiken	⟷ effektiv berechenbar	mit "Maschinen"

Typ-0 Grammatik (rek.aufz.)

Turing-Programme TM

mit endlichem Gedächtnis (Zustände) und unbeschränktem Zusatzspeicher (Band):
Eingabewort unbeschränkt lesen/bearbeiten

Typ-1 Grammatik (kontext-sensitiv)

TM wie oben, aber mit linear beschränktem Zusatzspeicher (Band):
Eingabewort wiederholt lesen/bearbeiten

Typ-2 Grammatik (kontextfrei)

Kellerautomat (Stackautomat, Pushdown-Automaton) PDA
mit endlichem Gedächtnis (Zustände) und beschränkt nutzbarem Kellerspeicher:
Eingabewort von links nach rechts (nichtdeterministisch) lesen

Typ-3 Grammatik (regulär)

endlicher Automat DFA
mit endlichem Gedächtnis (Zustände) ohne Zusatzspeicher: Eingabewort einmal lesen

reguläre Grammatik: Produktionen: $A \to aB$, $A \to \varepsilon$

Ableitungen:

$S \Rightarrow a_1 A_1$
$\Rightarrow a_1 a_2 A_2$
$\vdots$
$\Rightarrow a_1 a_2 \dots a_n A_n$
$\Rightarrow a_1 a_2 \dots a_n$

Terminalwort — jeweils ein Nichtterminal (endliches Gedächtnis)

kontextfreie Grammatik: Produktionen: $A \to w$

Linksableitungen:

$S \Rightarrow w_1 W_1$
$\Rightarrow w_1 w_2 W_2$
$\vdots$
$\Rightarrow w_1 w_2 \dots w_n$

mit $w_i \in \Sigma_T^*$, $W_i \in (\Sigma_T \cup \Sigma_N)^*$

"gemischt"

Terminalwort — jeweils ggf. viele Nichtterminals

kann man in den W_i's Nichtterminals und Terminals trennen?

Definition Eine kontextfreie Grammatik ist in

Greibach-Normalform gdw.

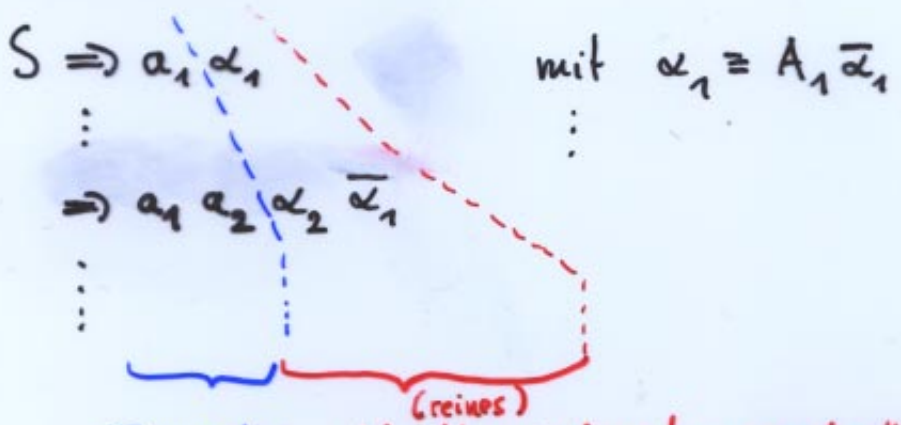
alle Produktionen haben die Form

$A \to a\,\alpha$ mit $A \in \Sigma_N$

$a \in \Sigma_T$

$\alpha \in \Sigma_N^*$

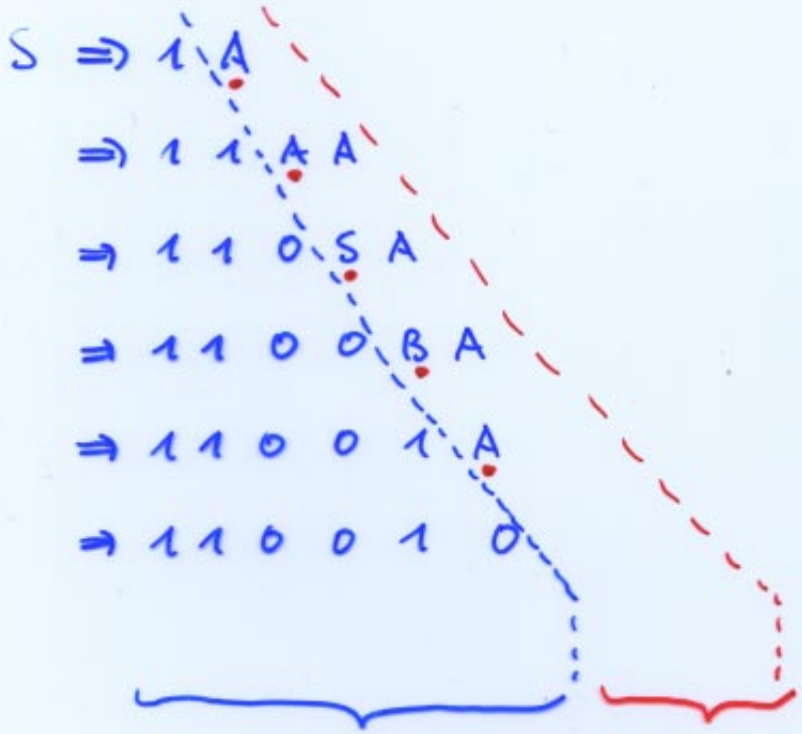
Linksableitungen:



Beispiel: für "GLEICH viele Nullen und Einsen" (siehe oben)

$S \to 0B$	$A \to 0$	$B \to 1$
$S \to 1A$	$A \to 0S$	$B \to 1S$
	$A \to 1AA$	$B \to 0BB$

eine Linksableitung:



Satz Sei $L \subset \Sigma_T^*$ eine kontextfreie Sprache mit $\varepsilon \notin L$. Dann gibt es eine kontextfreie Grammatik $\bar{G}$ in Greibach-Normalform mit $L = L(\bar{G})$.

Beweis. durch effektives Verfahren zur Lösung von $L(G) = L(\bar{G})$

- L kann durch eine kontextfreie Grammatik in Chomsky-Normalform erzeugt werden:

  $G = ((\Sigma, P), \Sigma_N, \Sigma_T, S)$ mit Produktionen der Form

  $A_i \to A_j A_k$

  $A_i \to a_j$

- sei $\Sigma_N = \{A_1, \dots, A_m\}$

  $\Sigma_T = \{a_1, \dots, a_n\}$

- Idee für effektives Transformations-Verfahren:
  - 5 Nebenbedingungen für Ergebnis, die auch als Vorbedingung erfüllt sind und als Invariante erhalten bleiben
  - 1 zusätzliche "Zwischenbedingung"

- **Kontrollstruktur:** Wiederholung "in Richtung Ziel"
- **Transformations-Methoden:**

Methode 1: betrachte $\boxed{A \to \alpha_1 C \alpha_2}$ mit:

$$\left.\begin{array}{l} C \to \beta_1 \\ \vdots \\ C \to \beta_r \end{array}\right\} \text{alle Produktionen mit linker Seite } C$$

↘ ersetze durch $A \to \alpha_1 \beta_1 \alpha_2$ … $A \to \alpha_1 \beta_r \alpha_2$

Methode 2: betrachte $\boxed{\begin{array}{l} A \to A\alpha_1 \\ \vdots \\ A \to A\alpha_r \end{array}}$ } alle Produktionen mit links: A, rechts: A am Anfang

mit: $\left.\begin{array}{l} A \to \beta_1 \\ \vdots \\ A \to \beta_s \end{array}\right\}$ alle sonstigen Produktionen mit links: A [rechts: nicht A am Anfang]

↘
- nimm neues Nichtterminal B
- ersetze durch

$A \to \beta_1 B$ … $A \to \beta_s B$

$B \to \alpha_1$ … $B \to \alpha_r$

$B \to \alpha_1 B$ … $B \to \alpha_r B$

Beh. 1: Jede Anwendung von Methode 1 oder Methode 2 lässt die erzeugte (Terminal-) Sprache unverändert.

Beweis:

zu Methode 1:

vorher	nachher
$A \Rightarrow \alpha_1 C \alpha_2$	$A \Rightarrow \alpha_1 \beta_r \alpha_2$
↑ Nichtterminal muss ersetzt werden	↑ für eines der β_r
$\Rightarrow \alpha_1 \beta_r \alpha_2$	
↑ für eines der β_r	

zu Methode 2:

vorher	nachher
betrachte alle Linksableitungen $A \overset{*}{\Rightarrow} w$ so dass für w erstmals gilt: w beginnt nicht mit A, und enthält kein B	
$A \Rightarrow A\alpha_{i(1)}$	$A \Rightarrow \beta_j B$
$\Rightarrow A\alpha_{i(2)}\alpha_{i(1)}$	$\Rightarrow \beta_j \alpha_{i(1)} B$
⋮	⋮
$\Rightarrow \beta_j \alpha_{i(t)} \cdots \alpha_{i(2)} \alpha_{i(1)}$	$\Rightarrow \beta_j \alpha_{i(1)} \cdots \alpha_{i(1)}$

$\overline{\Sigma}_N := \{A_1, \dots, A_m, B_1, \dots, B_m\}$

5 Nachbedingungen, auch Vorbedingung und Invariante:

(E1) rechte Seite beginnt mit Nichtterminal X
↝ rechte Seite besteht nur aus Nichtterminals

$Y \to X\,w \quad \rightsquigarrow \quad w \in \overline{\Sigma}_N^*$

(E2) rechte Seite beginnt mit Nichtterminal X
↝ dieses Nichtterminal stammt aus Σ_N

$Y \to X\,w \quad \rightsquigarrow \quad X \in \Sigma_N$

(E3) rechte Seite enthält Terminal a_j
↝ rechte Seite beginnt mit a_j

$Y \to \alpha_1\, a_j\, \alpha_2 \quad \rightsquigarrow \quad \alpha_1 = \varepsilon$

(E4) rechte Seite beginnt mit Nichtterminal aus Σ_N
↝ rechte Seite beginnt mit zwei Nichtterminals aus Σ_N

$Y \to A_j \alpha \quad \rightsquigarrow \quad \alpha = A_k \bar{\alpha}$

(E5) linke Seite aus $\{B_1, \dots, B_m\}$
↝ rechte Seite besteht nur aus Nichtterminals

$B \to \alpha \quad \rightsquigarrow \quad \alpha \in \overline{\Sigma}_N^*$

1 zusätzliche "Zwischenbedingung":

(E6) $A_i \to A_j \alpha \quad \rightsquigarrow \quad j > i$

Transformations-Verfahren

[Vorbedingung : (E 1)-(E 5) gemäß Chomsky-NF]

(I.)

betrachte alle $A_1 \to A_1\alpha$ Produktionen : ersetze mit Methode 2

betrachte alle $A_2 \to A_1\alpha$ Produktionen : ersetze mit Methode 1

betrachte alle $A_2 \to A_2\alpha$ Produktionen : ersetze mit Methode 2

$\vdots$

betrachte alle $A_m \to A_1\alpha$ Produktionen : ersetze mit Methode 1

betrachte alle $A_m \to A_2\alpha$ Produktionen : ersetze mit Methode 1

$\vdots$ $\qquad\qquad$ $\vdots$

betrachte alle $A_m \to A_{m-1}\alpha$ Produktionen : ersetze mit Methode 1

betrachte alle $A_m \to A_m\alpha$ Produktionen : ersetze mit Methode 2

[(E 1)-(E 5) : invariant

(E 6) : durch Induktion :

- in $A_k \to A_j\alpha$ mit $j < k$
  wird mit Methode 1 $[\alpha_1 := \varepsilon ; C := A_j ; \alpha_2 := \alpha]$
  A_j ersetzt, das schon (E 6) erfüllt
- in $A_k \to A_k\alpha$
  wird mit Methode 2 $[A := A_k]$
  rechts A_k ersetzt, so dass ausschließend
  (E 6) erfüllt]

[(E1)-(E6) ; alle Produktionen von Form

$A \to a\alpha$: erlaubt

$A \to \alpha$ mit $\alpha \in \overline{\Sigma}_N^*$: verboten]

<u>für</u> $k = m, \dots, 1$:

<u>$k=m$:</u> [wegen (E2) und (E6) keine verbotene Produktion der Form $A_m \to \alpha$, $\alpha \in \overline{\Sigma}_N^*$]

<u>$k<m$:</u> betrachte verbotene Produktion

$A_k \to \alpha$ mit $\alpha = C C_1 \dots C_l \in \overline{\Sigma}_N^*$

$C = A_i$ mit $i > k$ (E6)

↝ ersetze mit Methode 1

[Ind. Annahme für A_i : neue rechte Seiten erlaubt]

<u>für</u> alle Produktionen mit linker Seite aus $\{B_1, \dots, B_m\}$

[verbotene Form $B_i \to A_j \alpha$ gemäß (E5), (E2)]

↝ ersetze mit Methode 1

[neue rechte Seiten erlaubt, da A_j "erlaubt"]

Beispiel:

$\Sigma_N := \{ A_1, A_2, A_3 \}$ (mit $A_1 =: S$) $\qquad \Sigma_T := \{a, b\}$

P:

R1: $A_1 \to A_2 A_3$

R2: $A_2 \to A_3 A_1$

R3: $A_2 \to b$

R4: $A_3 \to A_1 A_2$ — verletzt (E6) ↝ ersetzen

R5: $A_3 \to a$

R4 mit M1: R6: $A_3 \to \overline{A_2 A_3}\, A_2$ — verletzt (E6) ↝ ersetzen

R6 mit M1: R7: $A_3 \to \overline{A_3 A_1}\, A_3 A_2$ — verletzt (E6) ↝ ersetzen

R8: $A_3 \to \overline{b}\, A_3 A_2$

R7 mit M2: R9: $A_3 \to a B_3$ — mit R5

R10: $A_3 \to b A_3 A_2 B_3$ — mit R8

R11: $B_3 \to A_1 A_3 A_2$

R12: $B_3 \to A_1 A_3 A_2 B_3$

Beispiel:

$\Sigma_N := \{ A_1, A_2, A_3 \}$ (=: S) $\qquad \Sigma_T := \{a, b\}$

P:

R1: $A_1 \to A_2 A_3$

R2: $A_2 \to A_3 A_1$

R3: $A_2 \to b$

~~R4: $A_3 \to A_1 A_2$~~ verletzt (E6) ↝ ersetzen

R5: $A_3 \to a$

~~R4 mit M1: R6: $A_3 \to A_2 A_3 A_2$~~ verletzt (E6) ↝ ersetzen

~~R6 mit M1: R7: $A_3 \to A_3 A_1 A_3 A_2$~~ verletzt (E6) ↝ ersetzen

R8: $A_3 \to b A_3 A_2$

R7 mit M2: R9: $A_3 \to a B_3$ mit R5

R10: $A_3 \to b A_3 A_2 B_3$ mit R8

R11: $B_3 \to A_1 A_3 A_2$

R12: $B_3 \to A_1 A_3 A_2 B_3$

k = 3: alle A_3 Produktionen erlaubt

k = 2: R2 verboten ↝ ersetzen

R2 mit M1:

R13 $A_2 \to a B_3 A_1$ mit R9

R14 $A_2 \to b A_3 A_2 A_1$ mit R8

R15 $A_2 \to a A_1$ mit R5

R16 $A_2 \to b A_3 A_2 B_3 A_1$ mit R10

k=1: R1 verboten ↷ ersetzen

R1 mit M1:

R17: $A_1 \to b\, A_3$ mit R3

R18: $A_1 \to a\, B_3\, A_1\, A_3$ mit R14

R19: $A_1 \to b\, A_3\, A_2\, A_1\, A_3$ mit R16

R20: $A_1 \to a\, A_1\, A_3$ mit R15

R21: $A_1 \to b\, A_3\, A_2\, B_3\, A_1\, A_3$ mit R16

B-Regeln:

R11 verboten ↷ ersetzen

R12 verboten ↷ ersetzen

R11 mit M1:

R22 $B_3 \to b\, A_3\, A_3\, A_2$ mit R17

R23 $B_3 \to a\, B_3\, A_1\, A_3\, A_3\, A_2$ mit R18

R24 $B_3 \to b\, A_3\, A_2\, A_1\, A_3\, A_3\, A_2$ mit R19

R25 $B_3 \to a\, A_1\, A_3\, A_3\, A_2$ mit R20

R26 $B_3 \to b\, A_3\, A_2\, B_3\, A_1\, A_3\, A_3\, A_2$ mit R21

R12 mit M1:

R27 $B_3 \to b\, A_3\, A_3\, A_2\, B_3$ mit R17

R28 $B_3 \to a\, B_3\, A_1\, A_3\, A_3\, A_2\, B_3$ mit R18

R29 $B_3 \to b\, A_3\, A_2\, A_1\, A_3\, A_3\, A_2\, B_3$ mit R19

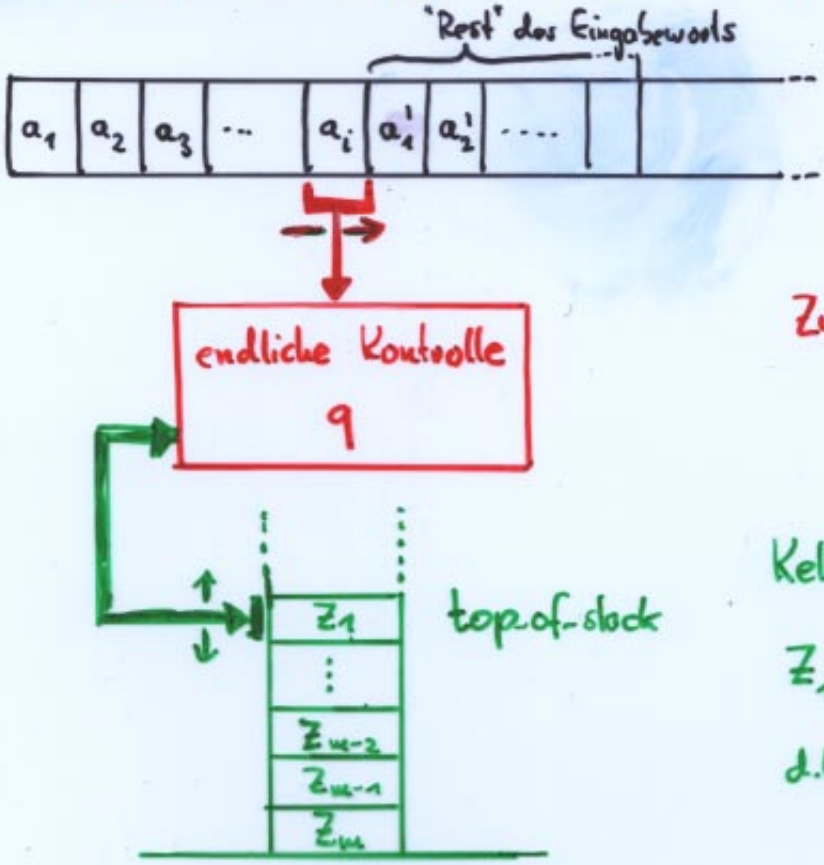
R30 $B_3 \to a\, A_1\, A_3\, A_3\, A_2\, B_3$ mit R20

R31 $B_3 \to b\, A_3\, A_2\, B_3\, A_1\, A_3\, A_3\, A_2\, B_3$ mit R21

Nichtdeterministische Kellerautomaten

(Stackautomaten, Pushdown Automata)

$$NPDA = (Q, \Sigma, \Gamma, q_0 \in Q, Z_0 \in \Gamma, \delta, F \subset Q)$$



Eingabewort mit
$a_i \in \Sigma$,
$a_i' \in \Sigma$

Zustand $q \in Q$

Kellerspeicher mit
$Z_1 \dots Z_m \in \Gamma^*$,
d.h. $Z_i \in \Gamma$

nichtdeterministische Überführungsfunktion

$$\delta : Q \times (\Sigma \cup \{\varepsilon\}) \times \Gamma \longrightarrow \mathcal{P}(Q \times \Gamma^*)$$

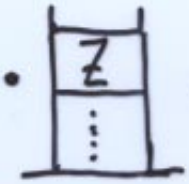
(momentaner) Zustand	gelesenes Eingabezeichen bzw. "Spontan"	oberstes Kellerzeichen	(Folge-) Zustand	abgelegte Kellerzeichen

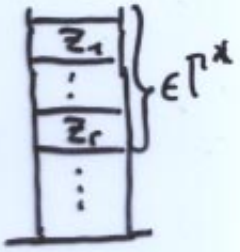
ein Übergang:

- betrachtet Zustand $q \in Q$
- liest (einmalig, "zerstörend")
  - nächstes Eingabezeichen $a \in \Sigma$
  - bzw. "spontan" ε
- liest (zerstörend)
  - oberstes Kellerzeichen $Z \in \Gamma$

für ein

$(q', Z_1 \dots Z_r) \in \delta(q, \begin{cases} a \\ \varepsilon \end{cases}, Z)$:

- Folgezustand $q' \in Q$
- 

ersetzen durch


Situation (Konfiguration)

(q, w, α) mit $q \in Q$, $w \in \Sigma^*$, $\alpha \in \Gamma^*$

- q: Zustand
- w: "Rest" des Eingabeworts
- α: Kellerinhalt

Startsituation: (q_0, w, Z_0) — Startzustand, Eingabewort, "Start-Kellerzeichen"

Überführungsrelation für Situationen:

$(q, a_1 \dots a_k, Z_1 \dots Z_m)$

$\vdash$ vermöge $\delta(q, a_1, Z_1) \ni (q', Z_1' \dots Z_r')$

$(q', a_2 \dots a_k, Z_1' \dots Z_r' Z_2 \dots Z_m)$

↑ a_1 "verbraucht" gelesen

bzw:

$(q, a_1 \dots a_k, Z_1 \dots Z_m)$

$\vdash$ vermöge $\delta(q, \varepsilon, Z_1) \ni (q', Z_1' \dots Z_r')$

$(q', a_1 \dots a_k, Z_1' \dots Z_r' Z_2 \dots Z_m)$

↑ "spontan"

erkannte (akzeptierte) Sprache:

$$L_K(\text{NPDA}) := \{ w \mid w \in \Sigma^* \text{ und } (q_0, w, Z_0) \vdash^* (q, \varepsilon, \varepsilon) \text{ mit } q \in Q \text{ "beliebig"} \}$$

"Erkennen durch leeren Keller"

alternativ:

$$L_Z(\text{NPDA}) := \{ w \mid w \in \Sigma^* \text{ und } (q_0, w, Z_0) \vdash^* (q, \varepsilon, \gamma) \text{ mit } q \in F \text{ und } \gamma \in \Gamma^* \text{ "beliebig"} \}$$

"Erkennen durch Zustände"

deterministische Kellerautomaten

$$PDA = (Q, \Sigma, \Gamma, q_0, Z_0 \in \Gamma, \delta, F \subseteq Q)$$

mit

$$\| \delta(q,a,Z) \| + \| \delta(q,\varepsilon,Z) \| \leq 1$$

f. alle $q \in Q$, $a \in \Sigma$, $Z \in \Gamma$

d.h. in <u>jeder</u> Situation, falls überhaupt etwas getan wird:

- <u>entweder</u> nächstes Eingabezeichen verarbeiten
- <u>oder</u> 'spontan',

und jeweils <u>eindeutig</u> bestimmt:

- Folgezustand
- neue Kellerzeichen

Beispiel:

$$L = \{ w \# w^R \mid w \in \{0,1\}^* \}$$

"umgangssprachlich":

- solange nicht # gelesen: Eingabezeichen $i \in \{0,1\}$ auf Keller durch $I \in \{0,1\}$ merken
- # gelesen : "Keller auf Lesen umstellen"
- solange Eingabe nichtleer : Eingabezeichen $i \in \{0,1\}$ mit oberstem Kellerzeichen $I \in \{0,1\}$ vergleichen

- arbeitet "deterministisch" mit Hilfe des Trennzeichens #

- ohne Trennzeichen: nichtdeterministisch die "Trennstelle (Spiegelstelle) erraten

Satz Die Arbeitsweisen

"Erkennen durch Zustände" und

"Erkennen durch leeren Keller"

sind gleichmächtig:

Zu jedem $NPDA_1$ kann ein $NPDA_2$ effektiv bestimmt werden mit

$$L_z(NPDA_1) = L_K(NPDA_2),$$

und umgekehrt.

Beweis: durch geeignete Simulationen:

I. "durch Zustände" ⇝ "durch leeren Keller":

- verhindere "leeren Keller ohne Akzeptieren":

  ⇝ $NPDA_2$ legt extra Kellerzeichen auf Kellerboden

- beim Akzeptieren "Keller leeren"

  ⇝ $NPDA_2$ erhält extra Zustand zum "Keller aufräumen"

formal:

$$Q_2 := Q_1 \cup \{ q_0^2 , q_E \}$$

q_0^2: "Kellerboden vorbereiten"; q_E: Keller aufräumen

$$\Gamma_2 := \Gamma_1 \cup \{ Z_0^2 \}$$

Z_0^2: extra Kellerboden

δ_2:

(1) [Kellerboden vorbereiten]

$$\delta_2 (q_0^2 , x , Z_0^2) := \begin{cases} \{ (q_0^1 , Z_0^1 Z_0^2) \} & \text{falls } x = \varepsilon \\ \emptyset & \text{falls } x \in \Sigma \end{cases}$$

(2),(3) [simuliere]

$$\delta_2 (q , x , Z) := \begin{cases} \delta_1 (q , x , Z) & \text{falls } q \in Q_1 \setminus E_1 \\ \delta_1 (q , x , Z) \cup \{ (q_E , \varepsilon) \} & \text{falls } q \in E_1 \end{cases}$$

für $q \in Q_1$, $Z \in \Gamma_1$, $x \in \Sigma \cup \{\varepsilon\}$

(4) [Keller aufräumen] ... iteriere!

$$\delta_2 (q_E , \varepsilon , Z) = \{ (q_E , \varepsilon) \} \quad \text{f. alle } Z \in \Gamma_2$$

Kellerzeichen Z "löschen"!

II. "durch leeren Keller" ↝ "durch Zustände":

- bei "leerem Keller" noch Übergang in akzeptierenden Zustand

↝ $NPDA_2$ legt extra Kellerzeichen auf Kellerboden:

"leerer $NPDA_1$-Keller" ≙ "nur extra $NPDA_2$-Kellerzeichen"

formal:

$$Q_2 := Q_1 \cup \{q_0^2, q_F\}$$

(q_0^2: Kellerboden vorbereiten; q_F: akzeptieren)

$$F_2 := \{q_F\}$$

$$\Gamma_2 := \Gamma_1 \cup \{Z_0^2\}$$

(Z_0^2: extra Kellerboden)

δ_2:

(1) $$\delta_2(q_0^2, x, Z_0^2) := \begin{cases} \{(q_0^1, Z_0^1 Z_0^2)\} & \text{falls } x = \varepsilon \\ \varnothing & \text{falls } x \in \Sigma \end{cases}$$

(2) $\delta_2(q, x, Z) = \delta_1(q, x, Z)$ für $q \in Q_1$, $x \in \Sigma \cup \{\varepsilon\}$, $Z \in \Gamma_1$

(3) $\delta_2(q, \varepsilon, Z_0^2) = \{(q_F, \varepsilon)\}$ für $q \in Q_1$

Satz Sei $L \subset \Sigma_T^*$.

Dann sind die folgenden Aussagen äquivalent:

(1) L ist kontextfrei,

d.h. ex. kfr Grammatik ($\underbrace{\text{in Greibach-Normalform}}_{\text{(o.B.d.A)}}$)

$G = ((\Sigma, P), \Sigma_N, \Sigma_T, S)$ $\quad A \to a B_1 \dots B_n$ mit $n \geq 0$

mit $L = L(G)$

(2) L wird durch einen nichtdeterministischen Kellerautomaten erkannt,

d.h. ex. $NPDA = (Q, \Sigma_T, \Gamma, q_0, Z_0, \delta, F)$

mit $L = L_{(K)}(NPDA)$

(o.B.d.A) $\hat{=}$ "Erkennen mit leerem Keller"

Die Transformationen $G \mapsto NPDA$ und

$NPDA \mapsto G$

sind (effektiv) berechenbar.

I. "G in Greibach-NF ⟼ NPDA":

G als "Automat" deuten:

$S \overset{*}{\Rightarrow} a_1 \dots a_k \, X_1 \dots X_\ell$

G: ersetzt
NPDA: als "top-of-stack" ersetzt

$\hookrightarrow a_{k+1} \bar{\alpha}$

G: "zwischenzeitlich" erzeugt
NPDA: auf Keller abgelegt

	schon "verarbeit":	gerade betrachtet:
G:	erzeugt	erzeugt
NPDA:	gelesen	gelesen

formal:

$Q = \{q_0\}$ [nur "formal benötigt": neben Kellerspeicher kein "wesentliches" endliches Gedächtnis]

$\Gamma := \Sigma_N$

$Z_0 := S$ ⇝ Startsituation: $(q_0, w \in \Sigma_T^*, S)$ ↑ Initialisierung des Kellers

$\delta(q_0, a, A) := \{(q_0, \alpha) \mid A \to a\alpha \in P\}$

F: "bedeutslos": Erkennen mit leerem Keller

Beh: $S \overset{*}{\Rightarrow} a_1 \dots a_k X_1 \dots X_\ell$ gdw

$$(q_0, a_1 \dots a_k \bar{w}, S) \overset{*}{\vdash} (q_0, \bar{w}, X_1 \dots X_\ell)$$

nach Lesen von $a_1 \dots a_k$
(mit "Restwort" $\bar{w} \in \Sigma_T^*$)
befindet sich $X_1 \dots X_\ell$ im Keller
↑
top

Beweis (Induktion über die Länge der Ableitung bzw. Ausführung)

Länge = 0:

G	NPDA
$S \overset{*}{\Rightarrow} S$	$(q_0, w, S) \overset{*}{\vdash} (q_0, w, S)$

beide Beziehungen gelten gemäß Def "*": reflexive, transitive Hülle

Länge > 0:

$$S \overset{*}{\Rightarrow} a_1 \dots a_k \underbrace{X_1 \dots X_r}_{\text{im letzten Schritt erzeugt}} \dots X_\ell$$

↷ ex. $r \in \{0, \dots, \ell\}$, ex. $X \to a_k X_1 \dots X_r \in P$

$$S \overset{*}{\Rightarrow} a_1 \dots a_{k-1} X X_{r+1} \dots X_\ell$$

Def. δ / Ind. Ann.

↷ ex. $r \in \{0, \dots, \ell\}$, ex. $(q_0, X_1 \dots X_r) \in \delta(q_0, a_k, X)$

$$(q_0, a_1 \dots a_k \bar{w}, S) \overset{*}{\vdash} (q_0, a_k \bar{w}, X X_{r+1} \dots X_\ell)$$

$\rightsquigarrow \quad (q_0, a_1 \dots a_k \bar{w}, S) \vdash^* (q_0, \bar{w}, \underbrace{X_1 \dots X_r}\, X_{r+1} \dots X_\ell)$

$a_1 \dots a_k$: im letzten Schritt "gelesen"

$X_1 \dots X_r$: im letzten Schritt abgelegt

II. "NPDA $\mapsto$ kfr G"

↑

o.B.d.A: "Erkennen mit leerem Keller", d.h.

$w \in \Sigma_T^*$ wird erkannt (akzeptiert) gdw

$(q_0, w, Z_0) \vdash^* (p, \varepsilon, \varepsilon)$

allgemeinere Beziehung:

$X \in \Gamma$: ein Kellerzeichen

$(q, \bar{w}, X) \vdash^* (p, \varepsilon, \varepsilon)$

q: beliebiger Zustand: $q \in Q$

$\bar{w}$: ein Teilwort: $\bar{w} \in \Sigma_T^*$

$(p, \varepsilon, \varepsilon)$: die Verarbeitung von $\bar{w}$ führt zum "Verarbeiten" von X

grob benannt: $[q, X, p]$ "beschreibt" Verarbeitung von $\bar{w}$

formal:

$$\Sigma_N := \{ [q, X, p] \mid p, q \in Q \text{ und } X \in \Gamma \} \cup S$$

P enthält genau:

(1) $S \to [q_0, Z_0, p]$ für $p \in Q$

(2) $[q, X, q_{m+1}] \to a\,[q_1, Y_1, q_2] \ldots [q_m, Y_m, q_{m+1}]$

f. alle $q_2, \ldots, q_{m+1}$, falls $(q_1, Y_1 \ldots Y_m) \in \delta(q, a, X)$

speziell für $m = 0$, d.h. $Y_1 \ldots Y_m = \varepsilon$: $[q, X, q_1] \to a$,
falls $(q_1, \varepsilon) \in \delta(q, a, X)$

Beh1: $[q, X, p] \overset{*}{\Rightarrow} \bar{w}$ gdw

$(q, \bar{w}, X) \overset{*}{\vdash} (p, \varepsilon, \varepsilon)$

Beh 2: Beh. 1 $\sim$ $L = L(G)$

Beweis Beh. 2:

$w \in L$

$L = L(\text{NPDA})$ ↝ ex. $p \in Q$: $(q_0, w, Z_0) \overset{*}{\vdash} (p, \varepsilon, \varepsilon)$

Beh. 1 ↝ ex. $p \in Q$: $[q_0, Z_0, p] \overset{*}{\Rightarrow} w$

Def. P, (1) ↝ ex $p \in Q$: $S \Rightarrow [q_0, Z_0, p] \overset{*}{\Rightarrow} w$

Def. L(G) ↝ $w \in L(G)$

Beweis Beh. 1:

'$\Rightarrow$' Induktion über Länge der Ableitung,

(die mindestens 1 beträgt, da keine reflexive Beziehung)

Länge = 1: $[q, X, p] \Rightarrow \bar{w}$

Def. '$\Rightarrow$' ⇝ ex. $\underbrace{[q, X, p]}_{\in \Sigma_N} \to \underbrace{\bar{w}}_{\in \Sigma_T^*} \in P$

Def. P: (2) ⇝ $|\bar{w}| \leq 1$ und $(p, \varepsilon) \in \delta(q, \bar{w}, X)$

Def. '$\vdash$' ⇝ $\underbrace{(q, \bar{w}, X)}_{\text{Situation}} \vdash (p, \varepsilon, \varepsilon)$

- Zustand
- Restwort
- Kellerinhalt
- X vom Keller entfernt
- falls $\bar{w} \in \Sigma_T$: $\bar{w}$ gelesen
- falls $\bar{w} = \varepsilon$: 'spontan'

Länge > 1:

1. Schritt der Ableitung

$$[q, X, p] \Rightarrow a\,[q_1, Y_1, q_2]\,[q_2, Y_2, q_3] \dots [q_m, Y_m, \underbrace{q_{m+1}}_{:= p}]$$

$$\overset{*}{\Rightarrow} \bar{w}$$

restliche Schritte

mit:

1. Schritt: $(q_1, Y_1 \dots Y_m) \in \delta(q, a, X)$ (*)

gemäß Def. P: (2)

restliche Schritte:

- $\bar{w} = a\, w_1 \dots w_m$ für $w_j \in \Sigma_T^*$
- $\underbrace{[q_j, Y_j, q_{j+1}] \overset{*}{\Rightarrow} w_j}_{\text{Ableitung kürzerer Länge}}$ für $j = 1, \dots, m$

Ind. Annahme

$$(q_j, w_j, Y_j) \overset{*}{\vdash} (q_{j+1}, \varepsilon, \varepsilon) \quad \text{für } j = 1, \dots, m \quad (**)$$

Def. "Übergänge"

$$(q_j, w_j, Y_j\, Y_{j+1} \dots Y_m) \overset{*}{\vdash} (q_{j+1}, \varepsilon, Y_{j+1} \dots Y_m) \quad (***)$$

für $j = 1, \dots, m-1$

Zusammengefasst

$$(q, \underbrace{a w_1 w_2 \dots w_m}_{\bar{w}}, X)$$

$$\vdash (q_1, w_1 w_2 \dots w_m, Y_1 Y_2 \dots Y_m) \quad \text{gemäß } (*)$$

$$\vdash (q_2, w_2 \dots w_m, Y_2 \dots Y_m) \quad \text{gemäß } (**) \text{ für } j = 1$$

$$\vdots$$

$$\vdash (q_m, w_m, Y_m) \quad \text{gemäß } (**) \text{ für } j = m-1$$

$$\vdash (\underbrace{q_{m+1}}_{p}, \varepsilon, \varepsilon) \quad \text{gemäß } (**) \text{ für } j = m$$

"⇐" Induktion über die Länge der Ausführung
(die mindestens 1 beträgt, da keine reflexive Beziehung)

Länge = 1:

$$(q, \bar{w}, X) \vdash (p, \varepsilon, \varepsilon)$$

Def "⊢"
$\bar{w} \in \Sigma_T \cup \{\varepsilon\}$ und $(p, \varepsilon) \in \delta(q, \bar{w}, X)$
↑ "lesend" ↑ "spontan"
↑ oberstes Kellerzeichen "zerstörend" lesen

Def P: (2)
$$[q, X, p] \to \bar{w} \in P$$

Def "⇒"
$$[q, X, p] \Rightarrow \bar{w}$$

Länge > 1: sei

$(q, \bar{w}, X) \vdash (q_1, w', Y_1 \dots Y_m)$ erster Schritt (a)

$\vdash^* (p, \varepsilon, \varepsilon)$ weitere Schritte

mit $\bar{w} = a w'$, $a \in \Sigma_T \cup \{\varepsilon\}$ gemäß erstem Schritt

$= a w_1 \dots w_m$: $w_j \in \Sigma_T^*$ derart, dass

erstmals Kellerinhalt $Y_{j+1} \dots Y_m$,

nachdem $w_1 \dots w_j$ gelesen,

wobei Zustand q_{j+1} erreicht;

$a w_1 \dots w_{j-1}$ schon gelesen $q_{m+1} := p$

$\rightsquigarrow (q_j, w_j \dots w_m, Y_j \dots Y_m) \vdash^* (q_{j+1}, w_{j+1} \dots w_m, \underbrace{Y_{j+1} \dots Y_m}_{\text{bislang nicht angerührt}})$

$\rightsquigarrow (q_j, w_j, Y_j) \vdash^* (q_{j+1}, \varepsilon, \varepsilon)$

Ind. Annahme

$\rightsquigarrow [q_j, Y_j, q_{j+1}] \overset{*}{\Rightarrow} w_j$ (b)

$\rightsquigarrow [q, X, \underbrace{p}_{= q_{m+1}}] \Rightarrow a [q_1, Y_1, q_2] \dots [q_m, Y_m, q_{m+1}]$ wegen (a) mit Def P:(2)

$\overset{*}{\Rightarrow} a w_1 \dots w_m$ gemäß (b)

inhärente Mehrdeutigkeit einer kfr Sprache L

- f. alle kfr G mit $L = L(G)$

  ex. $w \in L$: w besitzt zwei verschiedene G-Syntaxbäume

- unter Zweck "Syntax trägt Semantik"

  sehr misslich, deshalb i. Allg. vermeiden!

- Standardbeispiel für inhärente Mehrdeutigkeit:

$$L = \{ a^i b^j c^k \mid i=j \text{ oder } j=k \}$$

$$= \underbrace{\{ a^j b^j c^k \mid j \geq 0, k \geq 0 \}}_{=: L' \text{ kfr}} \cup \underbrace{\{ a^i b^j c^j \mid i \geq 0, j \geq 0 \}}_{=: L'' \text{ kfr}}$$

$$\text{kfr}$$

- beachte: L

$$\supseteq L' \cap L''$$

$$= \{ a^j b^j c^j \mid j \geq 0 \} \quad \underline{\text{nicht}} \text{ kfr}$$

- Vermutung: (geeignetes) $w \in L' \cap L''$ mehrdeutig:
  - ein Syntaxbaum "überprüft korrekte a-b Spiegelung"
  - anderer Syntaxbaum "überprüft korrekte b-c Spiegelung"

Beh: Sei G eine kfr. Grammatik mit

$$L := \{ a^i b^j c^k \mid i=j \text{ oder } j=k \} = L(G).$$

Dann ist G mehrdeutig.

Beweis: indirekt:

angenommen, G wäre eindeutig

o.B.d.A: G in Chomsky-Normalform

[Andernfalls: umformen; Eindeutigkeit bleibt erhalten]

- sei n die Konstante aus Ogden's Lemma
- betrachte $w := a^{n+n!} b^{n+n!} c^{n+n!} \in L$:

mit Ogden's Lemma konstruierbar aus:

(i) $z' := a^n b^n c^{n+n!} \in L$

(ii) $z'' := a^{n+n!} b^n c^n \in L$

mit verschiedenen Syntaxbäumen!

Genauer:

zu (i): markiere gemäß Ogden in z' alle a's

Ogden ↝ ex Zerlegung $z' = uvwxy$ mit insbesondere

$$uv^i wx^i y \in L \quad \text{f. alle } i \geq 0$$

und $\begin{cases} v \text{ oder } x & \text{enthält markiertes Zeichen} \\ v \text{ und } x & \text{homogen, d.h. } v, x \in \{a\}^* \cup \{b\}^* \cup \{c\}^* \end{cases}$

Fall 1: $x \in \{a\}^*$

v liegt vor x ↝ $v \in \{a\}^*$

Ogden $i := 2$ ↝ für $|vx| =: p \geq 1$: uv^2wx^2y

$$= a^{n+p} b^n c^{n+n!} \in L$$

↝ ↯ (Def. L)

Fall 2: $x \in \{c\}^*$, insbesondere x "nicht markiert"

v oder x "markiert" ↝ $v \in \{a\}^+$

Ogden $i := 2$ ↝ für $|v| =: p \geq 1$: uv^2wx^2y

$$= a^{n+p} b^n c^{n+n!+1} \in L$$

↝ ↯ (Def. L)

Fall 3: $x \in \{b\}^*$, insbesondere x "nicht markiert" und $x = b^j$ mit $0 \leq j \leq n$

s.o. ↝ $v \in \{a\}^+$

s.o. ↝ für $|v| =: p \geq 1$: $a^{n+p} b^{n+j} c^{n+n!} \in L$

Def. L ↝ $p = j$

also: für z' nur Fall 3 möglich; dafür gibt es

f. alle $i \geq 0$ Ableitungen der Form:

$$S \overset{*}{\Rightarrow} u\,A\,y \overset{x}{\Rightarrow} u\,v^i\,A\,x^i\,y \overset{*}{\Rightarrow} u\,v^i\,w\,x^i\,y \in L$$

$v = a^p$, $x = b^p$

wähle i derart, dass $w = a^{n+n!}\,b^{n+n!}\,c^{n+n!}$ entsteht:

$$\text{d.h.} \quad n + n! = n + (i-1)\cdot p$$

$$\rightsquigarrow \quad n! = (i-1)\cdot p$$

$$\rightsquigarrow \quad \underbrace{\frac{n!}{p}}_{\text{ganzzahlig!}} = i-1$$

zusammengefasst:

für $a^{n+n!}\,b^{n+n!}\,c^{n+n!}$ ex. Syntaxbaum T' mit

Teilbaum für $A \overset{x}{\Rightarrow} \underbrace{v^i\,w\,x^i}_{\in \{a\}^+\{b\}^+}$ $\rightsquigarrow$ kein c!

analog zu (ii):

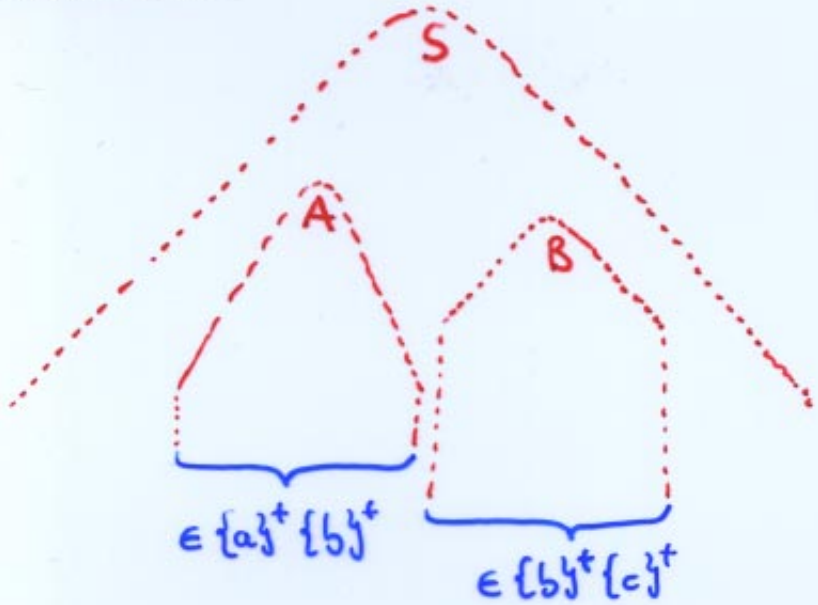
für $a^{n+n!}\,b^{n+n!}\,c^{n+n!}$ ex. Syntaxbaum T'' mit

Teilbaum für $B \overset{x}{\Rightarrow} \underbrace{\bar{v}^i\,\bar{w}\,\bar{x}^i}_{\in \{b\}^+\{c\}^+}$ $\rightsquigarrow$ kein a!

gemäß indirekter Annahme: $T' \cong T'' =: T$

isomorph: gleich bis um Umbenennungen

ausschaulich:



↷ Wurzeln A bzw. B der Teilbäume nicht auf einem Pfad

↷ zu Syntaxbaum T gibt es Ableitung der Form:

$S \overset{*}{\Rightarrow} t_1 A t_2 B t_3$ mit $t_1, t_2, t_3 \in \Sigma_T^*$ und:

f. alle k f. alle k

jeweils um p erhöht jeweils um q erhöht

↷ $S \overset{*}{\Rightarrow} t_1 a^{pk} w \underbrace{b^{pk} t_2 b^{qk}}_{\text{jeweils um } p+q \text{ erhöht}} \bar{w} c^{qk} t_3 \in L$ f. alle k

schließlich: wähle k mit: $\#(a's) \neq \#(b's) \neq \#(c's)$ ↯ (Def. L)

Berechenbarkeit schlechthin

- Ersetzungssystem : - Expansion, Kontext, Löschung
  - Terminal
- Post-System : Axiome, Regeln, Theoreme
- Turingmaschine : - endliche Zustandsmenge
  - unbegrenztes, wiederverwendbares Band
  - rechts, links, lesen, schreiben
- partiell rekursive Definition : - Null, Nachfolger, Projektion
  - Substitution, prim. Rek., μ-Op.
- MODULA : - Assignment,
  - Sequenz, For_Loop, While_Loop (Rekursion)

  (Bsp. für prozedurale Programmiersprachen)

  ↑ unbegrenztes Zählen

  ↑ Laufzeit-Stack
- Gödelnumerierung : - universelle Maschine
  - effektive Programmtransformation

  (implizite Programmdef. - Rekursionssatz)

⋮

jenseits des Berechenbaren:

- Halteproblem
- alle nichttrivialen Probleme (Satz von Rice)

Abschlußeigenschaften für rek. aufz. Sprachen:

$\circ, \cup, \cap, *$ (nicht: $\setminus$)

↑ SA rek. aufz., nicht rekursiv

dynamische Komplexität:

- beliebig
- Schachtelungstiefe der For_Loops
- $\mathcal{P} \subsetneq \mathcal{NP} \subsetneq \dots$

"endliches Gedächtnis"

- reguläre Grammatik : $A \to aB$, $A \to \lambda$

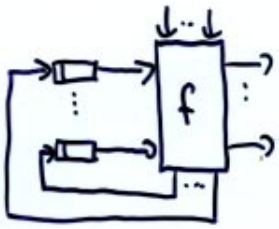
  kann deterministisch gemacht werden

- endlicher Automat

- reguläres Post-System : $uB \to wB$

- reguläre Ausdrücke : - $\varnothing$, $a \in \Sigma$
  - $\circ$, $\cup$, $*$

- Schaltwerke :



f Boolesche Fkt.

- Programme <u>ohne</u> (rekursive) Prozeduren,
  <u>ohne</u> dynamische Variablen
  <u>ohne</u> unendliche Typen

  - Werte der Variablen
  - Stelle im Programm

  } $\hat{=}$ endlichem Gedächtnis

jenseits des "endlichen Gedächtnisses":

Klammerstrukturen (pumping lemma, $\{a^n b^n \mid n \in \mathbb{N}\}$)

Abschlußeigenschaften für reguläre Sprachen:

$\circ, \cup, \cap, *, \setminus$

entscheidbare Probleme:

"alles"

dynamische Komplexität:

$\mathcal{O}(n)$, Eingabe lesen

Kellermechanismus

- kontextfreie Grammatik
- Kellerautomat

jenseits des Kellermechanismus:

Kontexte (pumping lemma, $\{a^n b^n c^n \mid n \in \mathbb{N}\}$)

Abschlußeigenschaften für kontextfreie Sprachen:

$\circ, \cup, *$ (nicht: $\setminus$, $\cap$)

entscheidbare Probleme:

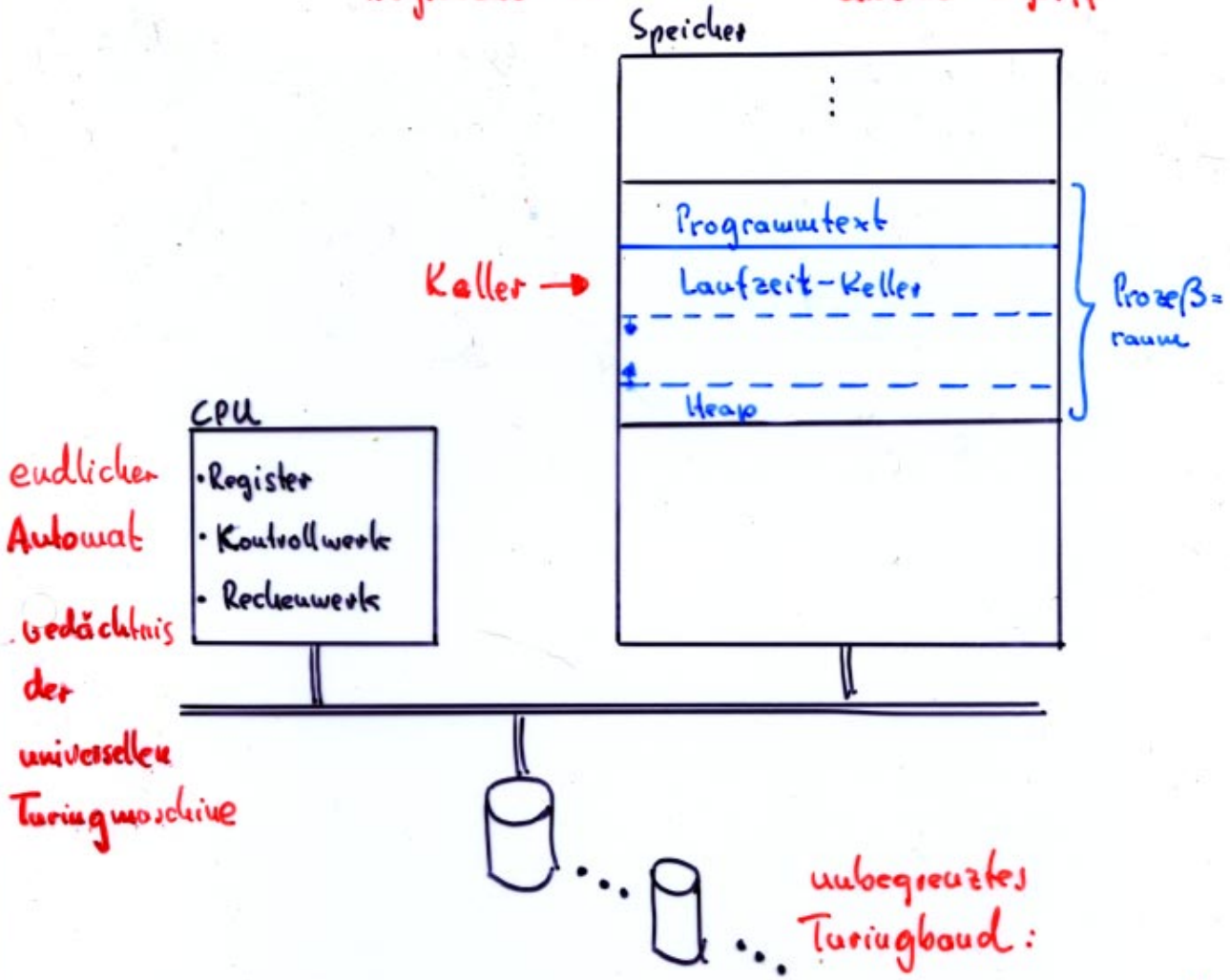
$L = \emptyset$, $x \in L$, L unendlich

(nicht: $L_1 \subset L_2$, $L_1 = L_2$)

dynamische Komplexität:

$\leq \mathcal{O}(n^3)$, Matrixmultiplikation

begrenztes Band: direkter Zugriff



endlicher Automat

Gedächtnis der universellen Turingmaschine

unbegrenztes Turingband:

(sequentieller Zugriff)

Semantik

```
PROCEDURE  Beispiel ... ;
VAR  x : INTEGER;

  FOR  x := 0  TO  10  DO ....  END;

  WHILE  x ≥ 0  DO ....  END;

  x := (x + x) * x;

END  Beispiel;
```

primitiv rekursiv

partiell rekursiv

Syntax

① kfw

```
PROCEDURE Beispiel ... ;
```

regulär

```
VAR x : INTEGER;
```

④

```
FOR x := 0 TO 10 DO .... END;
```

② ② ② ② kfw

regulär regulär

```
WHILE x ≥ 0 DO .... END;
```

③ ④ ③ ③ kfw

```
x := (x + x) * x ;
```

④ kontext-sensitiv

kfw

③ kfw

```
END Beispiel;
```

Aufgabe (für Informatik) ----> ad hoc,
im Einzelfall,
⋮

↓ "systematisch"

Aufgabenklasse

Werkzeuge für (u.a.)
- Abschlusseigenschaften
- Entscheidungs- bzw. Erkennungsprobleme

Informatik ist selbstbezüglich:
Die Erstellung eines Werkzeugs
ist selbst wieder Aufgabe!

↷ Ziel (Traum!):
"Automatisierung der Informatik"
[nicht voll verwirklichbar!!]

(Automatisierungs-) Ziel bleibt meistens "Traum"

Hindernisse: – komplexitätsmäßig: mindestens Zeit-exponentiell

– grundsätzlich:

{ nicht rek. aufz.: nicht einmal "erkennbar"
nicht rekursiv: nicht "entscheidbar"

Theorie liefert:

- "Standardbeispiele", z.B.
  - Erfüllbarkeitsproblem der Aussagenlogik
  - { Komplement des TM-Halteproblems
    TM-Halteproblem
- "Reduktionsmethode":
  "wäre <neues Problem> automatisierbar, so auch "Standardbeispiele"
  ↷ <neues Problem> auch nicht automatisierbar!
- "negativer" Glücksfall: Satz von Rice "als Meta-Reduktion"
- Universalität (bezüglich "Berechenbarkeit schlechthin) impliziert:
  - Selbstanwendung ↷ Diagonalisierung
  - unvermeidbare Partialität

trotzdem: wichtige Aufgabenklassen automatisierbar!

Werkzeuge

- arbeiten auf syntaktischen Konstrukten : stets (i. Allg.) endlich
- genügen gewünschter semantischer Eigenschaft : erlauben Bezug zu unendlichen Objekten

↘ Aufgaben bez. "Semantik" auf endlichen syntaktischen Konstrukten bearbeiten:

Korrektheit: falls "syntaktisches Ergebnis", dann wie "semantisch gewünscht"

Vollständigkeit: alles "semantisch Gewünschte" auch "syntaktisch als Ergebnis"

i. Allg: "Traum" (siehe Nichtberechenbarkeit)

↘
- klassisch: unter Korrektheit nur "Annäherung an Vollständigkeit"
- probabilistisch: Korrektheit und Vollständigkeit mit (vorbestimmbarer) "Fehlerwahrscheinlichkeit"

(grobe) Klassifikation von Aufgabenklassen

⋮

nicht rekursiv aufzählbar — Komplement Halteproblem

rekursiv aufzählbar — Wortproblem "rek. aufz."

rekursiv

⋮

"eher nicht effizient"
- mindestens Zeit-exponentiell — SAT_1 ? ($P \stackrel{?}{\neq} NP$)

⋮

"eher effizient"
- höchstens Zeit-polynomiell

⋮

kubische Zeit — Wortproblem "kfr."

⋮

quadratische Zeit

⋮

lineare Zeit — Wortproblem "regulär"

⋮

↷ reichhaltige Theorie vorhanden!
(aber auch: offene Probleme !!)

mächtiges Muster für Berechnungsmodelle

- "Ersetzungen" auf "Worten über Alphabet Σ"
  - einfachster Fall: Folgen, d.h. Σ^*
  - allgemeiner: "endliche Räume"
- "lokal": außerhalb der Ersetzungsstelle(n) unverändert
- vielfältige "Kontrollstrukturen" / "Adressierungen" / "Datenstrukturen" ... möglich
- "algebraischer Abschluss" von "Grundkonstrukten"

Nichtdeterminismus als Spezifikationsmittel:

- gedankliche Trennung von
  - "einer (erfolgreichen) Suche"
  - Suchraum systematisch erzeugen
- praktisch immer entfernbar, i. Allg. unter Erhöhung der "Komplexität"!

Grundbegriffe der theoretischen Informatik

- wichtiges, aber nicht alles!

- "Begriffe" sind "invariant unter Bezeichnungen"!

  - nicht: "Theorie der Informatik"

  - Vorlesungsauswahl "eher traditionell":
    "sequentielle Maschinen" und Sprachen und deren Komplexität

  - nicht behandelt:
    - Parallelität
    - Semantik (von Programmiersprachen)
    - Logik, automatisches Beweisen
    - ⋮