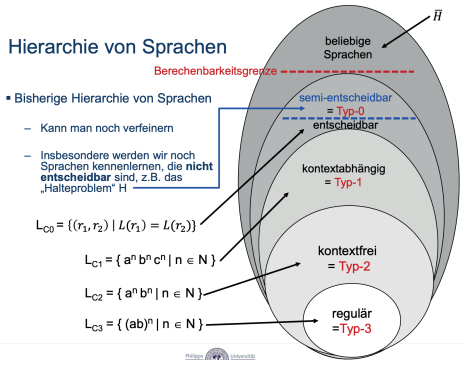
Hierarchie von Sprachen



- Bisherige Hierarchie von Sprachen
  - Kann man noch verfeinern
  - Insbesondere werden wir noch Sprachen kennenlernen, die **nicht entscheidbar** sind, z.B. das „Halteproblem" H

	Schnitt	Vereinigung	Komplement	Produkt	Stern
Typ 3	ja	ja	ja	ja	ja
Det. kf.	nein	nein	ja	nein	nein
Typ 2	nein	ja	nein	ja	ja
Typ 1	ja	ja	ja	ja	ja
Typ 0	ja	ja	nein	ja	ja

	Wort	Leerheit	Äquivalenz	Schnitt
Typ 3	ja [DFA]	ja	ja	ja
Det. kf.	ja	ja	ja	nein
Typ 2	ja [PDA,CYK]	ja	nein	nein
Typ 1	ja	nein	nein	nein
Typ 0	nein	nein	nein	nein

Grammatik	Beschreibung durch
Typ 3 (regulär)	Reguläre Grammatik DFA NFA, ε-NFA Regulärer Ausdruck
deterministisch kontextfrei	deterministischer Kellerautomat
Typ 2 (kontextfrei)	Kontextfreie Grammatik Kellerautomat (PDA)
Typ 1 (kontextsensitiv)	Kontextsensitive Grammatik LBA
Typ 0 (allgemein)	Typ 0-Grammatik Turingmaschine (TM)

Für eine partielle Funktion f :: $\mathbb{N}^k \to \mathbb{N}$ sind äquivalent:

- f ist Turing-berechenbar
- f mit TP-Programm berechenbar
- f ist RM-berechenbar
- f ist Goto-berechenbar
- f ist While-berechenbar

Nerode

u \ v	ε	0	1	10
ε		ε	1	01
0			0	0
1				1
10				

Sei L eine Sprache. Zwei Worte u, v ∈ Σ* heißen L-trennbar, falls es ein w ∈ Σ* gibt mit uw ∈ L aber vw ∉ L, oder umgekehrt.

Nerode: L sei eine Sprache. Gibt es n Worte, die paarweise L-trennbar sind, so hat jeder Automat, der L erkennt, mindestens n Zustände.

- $L_{diff} = \{a^m b^r \mid m \neq r\}$
  - $\{a^i \mid i \geq 0\}$ ist unendliche trennbare Menge
  - trenne a^i von allen a^j $(i \neq j)$ mittels b^i
- $L_{ambm} = \{a^m b^m \mid m \geq 0\}$
  - $\{a^i \mid i \geq 0\}$ ist unendliche trennbare Menge
  - trenne a^i von allen a^j $(i \neq j)$ mittels b^i
- $L_{fact} = \{a^{m!} \mid m \geq 3\}$
  - $\{a^{m!} \mid n \geq 0\}$ ist unendliche trennbare Menge
  - falls n < m, trenne $a^{n!}$ von $a^{m!}$ mittels $a^{n!-n}$:
    - $a^n a^{n!-n} = a^{(n+1)!} \in L_{fact}$, aber $a^m a^{n!-n} < a^{(m+1)!}$, daher $a^m a^{n!-n} \notin L_{fact}$.

Wortinduktion

Ein Induktionsbeweis

- Behauptung: $\forall w \in \Sigma^*: w \circ \varepsilon = w$
- Wähle: $P(w) := w \circ \varepsilon = w.$
  - Induktionsanfang: P(ε)
    $P(\varepsilon),\ \varepsilon \circ \varepsilon = \varepsilon, \Leftrightarrow true$
  - Induktionsschritt:
    $\forall a \in \Sigma: \forall v \in \Sigma^*: P(v) \Rightarrow P(a.v)$:

    Für beliebige $a \in \Sigma, v \in \Sigma^*$
    sei $P(v)$, d.h. $v \circ \varepsilon = v$ // Induktionshypothese
    Es folgt $(a.v) \circ \varepsilon = a.(v \circ \varepsilon)$ // Fall 2 in Def. von ∘
    $= a.v$ // Induktionshypothese
    Also ist $P(a.v)$ bewiesen
    Also ist $\forall a \in \Sigma: \forall v \in \Sigma^*: P(v) \Rightarrow P(a.v)$ bewiesen
  - Induktionsschluss:
    $\forall w \in \Sigma^*: P(w).$

Wir nehmen an, dass $P(v)$ gilt. Bisher aber nur für $v = \varepsilon$ gezeigt.

Da diese Implikation gilt, können wir nun beliebige Worte $v \in \Sigma^*$ ausgehend von ∅ erzeugen

Konkatenieren (= Aneinanderhängen)
- $u \circ v = uv$
- Formale Definition
  1. $\varepsilon \circ v = v$
  2. $(a.u) \circ v = a.(u \circ v)$

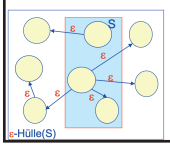
Länge (Anzahl der Zeichen)
- |w| = Anzahl der Zeichen in w
- Formale Definition:
  1. $|\varepsilon| = 0$
  2. $|a.v| = 1+|v|$

// Fall 1 in Def von ∘

$\delta^* : Q \times \Sigma^* \to Q$ definiert durch
- $\delta^*(q, \varepsilon) := q$
- $\delta^*(q, a.u) := \delta^*(\delta(q,a),u)$

epsilon-Hülle

ε(S) = kleinste Menge von Zuständen, die
- S umfasst
- gegen ε-Transitionen abgeschlossen ist



1. $S \subseteq \varepsilon(S)$
2. $q \in \varepsilon(S) \Rightarrow \delta(q, \varepsilon) \subseteq \varepsilon(S)$

Pumping-Lemma

Beispiel

- $L_{anbn} = \{a^n b^n \mid n \geq 0\}$ ist nicht durch endl. Automaten erkennbar.
  - Angenommen, L_{anbn} wäre regulär. Dann gäbe es ein k wie im Pumping Lemma. Jedes k-große ($|w| \geq k$) Wort $w \in L_{anbn}$ hätte im k-vorderen Bereich ($|xy| \leq k$) ein nichtleeres Teilwort y, das sich „aufpumpen" lässt.
  - Mit dem k von oben betrachten wir jetzt das Wort $a^k b^k$
    1. es ist in L_{anbn}
    2. es ist k-gross (|w|=2*k > k)
  - Es müsste im k-vorderen Bereich ein Teilwort haben, das sich aufpumpen lässt
    - der k-vordere Bereich besteht aber nur aus a's
    - Wenn wir hier einen nichtleeren Teil y aufpumpen, bekommen wir ein Wort mit mehr a's als b's
    - das wäre nicht mehr in L_{anbn}. Widerspruch !

x	y	z
aaaaaaaaaabbbbbbbbbb ∈ L		
k	k	

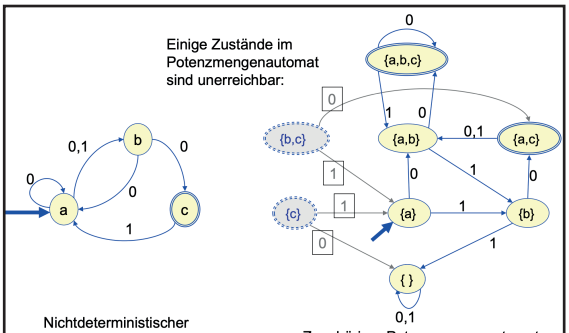
aber aaaaaaaaaaaabbbbbbbbbb ∉L

zu viele a-s

Es gibt daher keinen endlichen Automaten A mit L_{anbn}=L(A)

NFA -> DFA

Einige Zustände im Potenzmengenautomat sind unerreichbar:



Nichtdeterministischer Automat

Zugehöriger Potenzmengenautomat

rekursive Funktionen

primitiv-rekursiv

- Basisfunktionen
  - $f(x) = k$ Konstante $k \in \mathbb{N}$
  - $\pi_i^n(x_1, ..., x_n) = x_i$ i-te Projektion
  - $succ(x) = x + 1$ Nachfolger

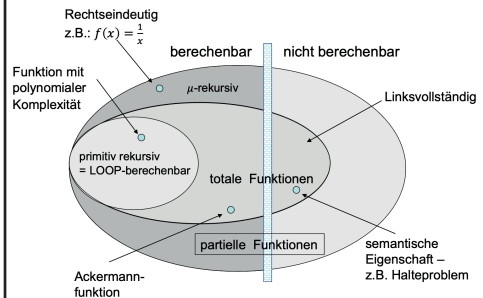
mu-rekursiv

- Basisfunktionen
  - f(x) = k Konstante $k \in \mathbb{N}$
  - $\pi_i^n(x_1,...,x_n) = x_i$ i-te Projektion
  - succ(x) = x+1 Nachfolger
- Abschluss unter
  - Komposition (Einsetzen)
  - primitives Rekursionsschema
  - μ-Operator

mu-Operator

- Sei f: $\mathbb{N}^{k+1} \to \mathbb{N}$ eine berechenbare Funktion. Definiere

$$\mu(f)(x_1,...,x_k) := \begin{cases} \min\{ n \mid f(n,x_1,...,x_k) = 0 \} & \text{falls } \forall\, i \leq n: f(i,x_1,...,x_k) \neq \perp \\ \perp & \text{sonst} \end{cases}$$



- Satz: Eine Funktion ist genau dann p.r. wenn sie LOOP-berechenbar ist.
- Jede p.r.-Funktion ist auch total.

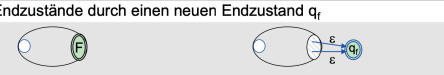
if-then-else (primitiv rekursiv)

- Seien B : $\mathbb{N}^k \to \mathbb{N}$ sowie f, g : $\mathbb{N}^k \to \mathbb{N}$ p.r. , dann auch

$ite(x_1,...,x_k) := B(x_1,...,x_k)\ ?\ f(x_1,...,x_k)\ :\ g(x_1,...,x_k)$

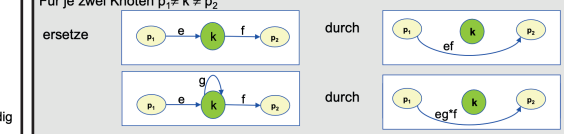
DFA -> regex

1. Ersetze Endzustände durch einen neuen Endzustand q_f



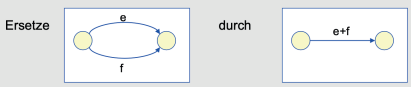
2. Elimination innerer Knoten (k ∉ {q_0,q_f})

Für je zwei Knoten $p_1 \neq k \neq p_2$

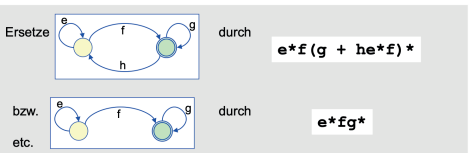


danach entferne k.

3. Elimination paralleler Kanten



4. Zum Schluss hat der Automat zwei Zustände. Ersetze ihn durch reg. Ausdruck



Grammatiken

Arithmetik

Expr → Expr + Term
| Term

Term → Term * Factor
| Factor

Factor → (Expr)
| id | num

Auswahl anderer Regeln erzeugt i.A. andere Worte w ∈ L(G)

Bei einer Links-Ableitung wird immer das linkeste Nonterminal ersetzt

Eine Links-Ableitung unter Verwendung anderer Regeln:

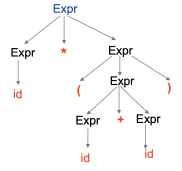
Expr ⇒ Expr + Term
⇒ Term + Term
⇒ Factor + Term
⇒ num + Term
⇒ num + Factor
⇒ num + (Expr)
⇒ num + (Term)
⇒ num + (Term * Factor)
⇒ num + (Factor * Factor)
⇒ num + (id * Factor)
⇒ num + (id * num)

Satz: Jede kontextfreie Grammatik G lässt sich äquivalent umformen in eine Grammatik G', deren Regeln von der Form sind:

A → a
A → BC

Falls ε ∈ L(G) gibt es noch die Produktion S → ε und S kommt in keiner Produktion rechts vor.

1. Entferne nicht terminierende und nicht erreichbare Variablen)* (*) optionaler Schritt. Kann nach 2. ggf. wiederholt werden
2. Mache G ε-frei und entferne alle 1-Variablen-Regeln.
3. In jeder Regel A → α mit |α| > 1 ersetze Terminalsymbol a durch neue Variable X_a und füge die Regel $X_a \to a$ hinzu.
4. Ersetze jede Regel der Form
   $A \to B_1 B_2 ... B_n$ mit Hilfe neuer Variablen C_1 . ... . C_{n-2} durch
   $A \to B_1 C_1, \quad C_1 \to B_2 C_2 \quad ... \quad C_{n-2} \to B_{n-1} B_n$

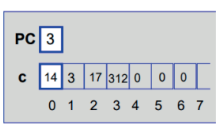


repräsentiert Ableitung
Expr ⇒* id * (id + id)

Besitzt L(G) ein Wort w, zu dem es mehrere Herleitungsbäume gibt, so heißt G mehrdeutig.

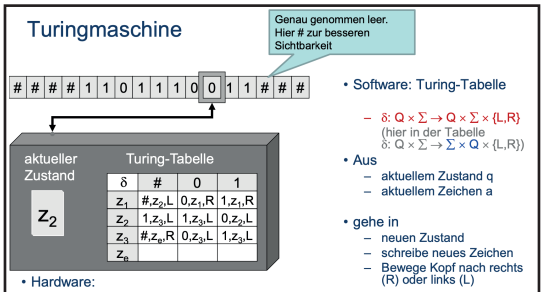
Registermaschinen (RM)

- Hardware:
  - Unbegrenztes Array von Speicherzellen c[1], c[2], …
    - Jede kann eine nat. Zahl speichern
  - Akkumulator c[0]
  - Programmzähler PC



- **Befehle :** **Semantik:**
  - Load n — c[0] := n // direkt
  - Load [n] — c[0] := c[n] // aus Speicher
  - Store [n] — c[n] := c[0]
  - Add [n] — c[0] := c[0] + c[n]
  - Sub⁺ [n] — c[0] := max(0, c[0] - c[n])
  - Goto i — PC := i
  - JZero i — PC := i, falls c[0]=0 // jump if zero

Turingmaschine



- Hardware:
  - Alphabet Σ
  - Unbegrenztes Band für Zeichen aus Σ
  - Beweglicher Lese-Schreibkopf, {L,R}
  - Zustände Q
- Software: Turing-Tabelle
  - $\delta: Q \times \Sigma \to Q \times \Sigma \times \{L,R\}$ (hier in der Tabelle $\delta: Q \times \Sigma \to \Sigma \times Q \times \{L,R\}$)
- Aus
  - aktuellem Zustand q
  - aktuellem Zeichen a
- gehe in
  - neuen Zustand
  - schreibe neues Zeichen
  - Bewege Kopf nach rechts (R) oder links (L)
- fertig,
  - falls Haltezustand erreicht oder kein Eintrag in Tabelle

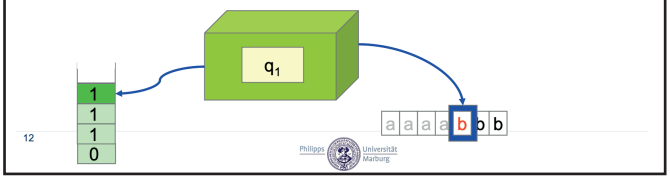
Stack-Maschine (PDA)

$Q = \{ q_0, q_1, q_2 \}$
$\Sigma = \{a, b\}$
$\Gamma = \{ 0,1 \}$
$s_0 = 0$

M

$\delta(q_0, a, 0)$	$= \{ (q_1, 0), (q_1, 10) \}$
$\delta(q_1, a, 0)$	$= \{ (q_1, 10) \}$
$\delta(q_1, a, 1)$	$= \{ (q_1, 11) \}$
$\delta(q_1, b, 1)$	$= \{ (q_2, \varepsilon) \}$
$\delta(q_2, b, 1)$	$= \{ (q_2, \varepsilon) \}$
$\delta(q_2, \varepsilon, 0)$	$= \{ (q_2, \varepsilon) \}$
$\delta(q_i, u, k)$	$= \{\}$ sonst

Stackmaschine beginnt immer mit s_0 auf dem ansonsten leeren Stack.
Im Beispiel kann das Wort aaaabbb den Stack entleeren :



12

Philipps Universität Marburg

GOTO, WHILE, LOOP

Ähnlich wie RM, nur

- Variablennamen statt Speicherzellen
- Einfache arithmetische Ausdrücke
  - *Expr*
- Vergleichsausdrücke
  - *Bexpr*
- Anweisungen
  - Zuweisung
  - Bedingter Sprung
- Datentyp: Nat (ℕ)

Goto

Stmt :: <n> : id := Expr
| <n> : if Bexpr Goto <m>

Expr :: id | 0 | succ(Expr)

Bexpr :: Expr ≤ Expr

Semantik: offensichtlich
<n>, <m> : Zeilennummern

- Äquivalente Algorithmenkonzepte
  - Goto-Sprache
  - While-Sprache
  - Kleene:
    - Ein While genügt
  - TP-Sprache, Registermaschinen
  - Stets gleiche Funktionenklasse
  - Church'sche These
    - *partiell berechenbaren Funktionen*
- Loop-Sprache
  - echt kleinere Klasse
  - alle Funktionen total
  - umfasst alle Algorithmen polynomialer Komplexität
  - Dennoch (siehe später) Nicht jede totale Funktion ist Loop-berechenbar

LOOP

Stmt :: id := Expr
| Stmt ; Stmt
| if Bexpr then Stmt else Stmt
| **loop x do Stmt**

Expr :: id | 0 | succ(Expr)

Bexpr :: Expr ≤ Expr

- Minimalistische Sprache
  - Prototypisch
- Keine Zeilennummern
  - Keine Sprünge
- Semikolon „;" bedeutet Hintereinander-ausführung
- Klammern optional
  - begin … end
  - { … }
- Stattdessen: Einrückung

While

Stmt :: id := Expr
| Stmt ; Stmt
| if Bexpr then Stmt else Stmt
| while Bexpr do Stmt

Expr :: id | 0 | succ(Expr)

Bexpr :: Expr <= Expr

Entscheidbarkeit

- Eine Teilmenge $U \subseteq \mathbb{N}$ (analog $U \subseteq \Sigma^*$) heißt entscheidbar, wenn ihre charakteristische Funktion

$$\chi_U(n) := \begin{cases} 1 & \text{falls } n \in U \\ 0 & \text{sonst} \end{cases}$$

berechenbar ist

- U heißt semientscheidbar, wenn

$$\partial_U(n) := \begin{cases} 1 & \text{falls } n \in U \\ \perp & \text{sonst} \end{cases}$$

berechenbar ist

- Satz: U ist genau dann entscheidbar, wenn U und ℕ-U semientscheidbar.
  - Beweis : klar // Foliensatz 02, Folie 27

semi-entscheidbar

- Eine Menge U heißt aufzählbar, falls es eine surjektive totale berechenbare Funktion $f: \mathbb{N} \to U$ gibt.
- U ist genau dann aufzählbar, wenn es einen Algorithmus gibt, der genau alle Elemente von U produziert:
  - Wir nehmen dazu eine Schreibanweisung „Write <n>" an:
    n := 0 ; While true Do { Write(f(n)); n := n+1 }
- Beispiele:
  - Die Menge aller Primzahlen ist aufzählbar.
  - Die Menge aller Teilworte von π ist aufzählbar
  - Die Menge aller n mit Ulam(n)=1 ist aufzählbar
- Satz: Eine Menge $U \subseteq \mathbb{N}$ ist genau dann aufzählbar, wenn sie semi-entscheidbar ist.

Aufzählbarkeit

- Sei $U \subseteq \mathbb{N}$ (bzw. $U \subseteq \Sigma^*$)
  - Äquivalent
    - U ist semi-entscheidbar
    - U ist aufzählbar (Bild einer totalen berechenbaren Funktion)
    - U ist Definitionsbereich einer partiellen berechenbaren Funktion
  - Es sind äquivalent
    - U ist entscheidbar
    - U und ℕ-U sind aufzählbar
    - U ist monoton aufzählbar

U heißt monoton aufzählbar, falls die Elemente von U in der natürlichen Reihenfolge aufgezählt werden.

Gödelisierung

- Jedes RM-Programm P ist eines der P_i.
  i ist dann die Gödelnummer von P (i = GN(P))
- Jede partielle berechenbare k-stellige Funktion f erscheint (mehrfach) als ein $\varphi_i^{(k)}$.
  i ist dann eine Gödelnummer von f (i ∈ GN(f))

mu-rekursiv

$ld : \mathbb{N} \to \mathbb{N}$

$$ld(x) = \begin{cases} y & \text{mit } 2^y = x, \text{ falls } y \text{ existiert} \\ \perp & \text{sonst} \end{cases}$$

Z.: ld ist μ-rekursiv:

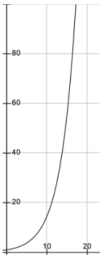
$$f(y, x) = (2^y - x) + (x \dot{-} 2^y)$$
$$(\mu f)(x) = ld(x)$$

Halteproblem

- Allgemeines Halteproblem
  - Gegeben
    - Programm P_m (Programm mit Gödelnummer m)
    - Input n
  - Gefragt:
    - Hält P mit Input n ?
  - Konkret:
    - Sei $A(m,n) := \begin{cases} 0 & \text{falls } \varphi_m(n) = \perp \\ 1 & \text{sonst} \end{cases}$
    - Ist A(m,n) berechenbar?
- Spezielles Halteproblem
  - Gegeben
    - P_m
  - Gefragt
    - Hält P_m mit Input m (m als Stellvertreter des Programmtextes selbst)
    - Ist A(m,m) berechenbar?

Komplexitätsklasse P

- P = Klasse aller Sprachen mit polynomieller Komplexität
  - Palindrom ∈ P
  - SAT : unbekannt
    - Vermutung: SAT ∉ P
  - CLIQUE, TSP : unbekannt
    - Vermutung: CLIQUE ∉ P, TSP ∉ P
- Wir werden sehen:
  - Falls SAT ∈ P ⇔ CLIQUE ∈ P ⇔ TSP ∈ P ⇔ …
    - Wenn eine davon in P ist, dann alle
    - Wenn eine davon nicht in P ist, dann alle
  - Bis heute ungelöstes Problem
    - Wenn Sie berühmt (und reich) werden wollen:
      - lösen Sie es
      - sonst wird es jemand anderes tun …



SAT Heuristiken
$O(1.308^n)$

Komplexitätsklasse NP

- P
  - Probleme, die effizient lösbar sind
    - in polynomieller Zeit
- NP (Nichtdeterministisch Polynomiell)
  - Probleme, deren Lösungskandidaten effizient überprüft werden können
    - in polynomieller Zeit
  - Es dürfen exponentiell viele Lösungskandidaten sein
    - hierachische Verteilung

Satz von Rice

$A = \{ n \mid \exists x. \ \varphi_n(x) = 3 \}$
$E_A = \{ f \in \mathcal{R}^n \mid \exists x. \ f(x) = 3 \}$

$g(x) = 0 \quad \notin E_A$
$f(x) = 3 \quad \in E_A$

⇒ nicht trivial $\quad \emptyset \neq E_A \neq \mathcal{R}$

Rice ⇒ A ist nicht entscheidbar

Myhill-Nerode Schema

3.2 $L_2 = \{ wcw \mid w \in \{a,b\}^* \}$

nicht regulär. (1P)
uvndl. viele Klassen (1P)

z.B. $[a^i b^i]$ für alle $i \geq 0$. (1P)

Begründung: paarweise Inäquivalent:
$i \neq j$: $a^i b^i c a^j b^i \in L_2$ (3P)
aber $a^i b^j c a^j b^i \notin L_2$
unterschiedl. Präfix — gleiches Suffix

Reduktion

$D = \{ n \mid |L(M_n)| \geq 3 \}$ n. ent.

$H \leq D$ bew. f: $n \in H \Leftrightarrow f(n) \in D$.

red. ⇒ ent.

$f(n) = n'$:
$M_{n'}$: input w; delete w; write n; run M_n; accept w

$n \in H \Rightarrow M_n(n) \neq \perp \Rightarrow L(M_{n'}) = \Sigma^* \Rightarrow |L(M_{n'})| \geq 3 \Rightarrow n' \in D$

$n \notin H \Rightarrow M_n(n) = \perp \Rightarrow L(M_{n'}) = \emptyset \Rightarrow |L(M_{n'})| = 0 < 3 \Rightarrow n' \notin D$

⇒ D n. ent.

Entscheidbarkeit prüfen

Ist U entscheidbar ?

1. U bekannt ? z.B. Primzahl (Aufzählbar)
2. U unterteilen $U = \{ n \mid A(n) \wedge B(n) \} = U_1 \cap U_2$
   $U_1 = \{ n \mid A(n) \}$ ✓ $U_2 = \{ n \mid B(n) \}$
   - 3.1 $U = \{ n \mid \exists x. \ P_n$ enthält in Zeile x eine 3 $\}$
     ⇒ entscheidbar (syntaktisch ⇒ Programm durchgehen z.B. mit einer TM)
   - 3.2 $U = \{ n \mid \exists x. \ \varphi_n(x) = 3 \}$ ⇒ Satz von Rice
   - 3.3. $U = \{ n \mid \varphi_n(n) = n^2 \}$ ⇒ Diagonalisierung
4. Reduktion $H \leq U$ ⇒ nicht entscheidbar
   $\overline{H} \leq U$ ⇒ nicht semi-entscheidbar

Diagonalisierung

$C = \{ n \mid \varphi_n(n) = n^2 \}$

Angenommen: C entscheidbar, χ_C berechenbar,

$$d(n) = \begin{cases} n^2 + 1, & \chi_C(n) = 1 \Leftrightarrow n \in C \\ n^2, & \chi_C(n) = 0 \Leftrightarrow n \notin C \end{cases}$$

⇒ d berechenbar
⇒ $d = \varphi_j$ für ein j
Fall 1: $\varphi_j(j) = j^2 \Rightarrow \chi_C(j) = 1 \Rightarrow d(j) = \varphi_j(j) = j^2 + 1$ ↯
Fall 2: $\varphi_j(j) \neq j^2 \Rightarrow \chi_C(j) = 0 \Rightarrow d(j) = j^2$ ↯

⇒ d nicht berechenbar
⇒ χ_C nicht berechenbar
⇒ C nicht entscheidbar