Theoretische Informatik II

Einheit 4

Allgemeine Berechenbarkeit

1. Turingmaschinen und Typ-0 Sprachen

---

2. Turing-Berechenbarkeit
3. Rekursive Funktionen
4. Funktionale und logische Programme
5. Elementare Berechenbarkeitstheorie
6. Unlösbare Probleme

Theorie der Berechenbarkeit

- **Welche Arten von Berechnung sind denkbar?**
  - Es gibt weit mehr Modelle als nur die Standard PC Architektur
  - Lisp Maschinen, Parallelrechner, Neuronale Netze, (Quantencomputer)
  - Sind manche Modelle leistungsstärker als andere?
- **Welche allgemeingültigen Zusammenhänge gibt es?**
  - Welche Grundeigenschaften muß jedes Berechnungsmodell erfüllen?
  - Wie kann man Lösungen schematisch wiederverwenden?

    Abschlußeigenschaften und Problemtransformation ("Reduktion")
- **Wo liegen die Grenzen für Computer?**
  - Gibt es Funktionen, die prinzipiell nicht berechenbar sind?
  - Eigenschaften, die unentscheidbar sind?
  - Sprachen, deren Elemente nicht vollständig aufgezählt werden können?

Mit welchen Techniken kann man dies beweisen?

Warum braucht man Berechenbarkeitsmodelle?

Welche der folgenden Funktionen ist berechenbar? Warum?

$$f(x) = \begin{cases} 1 & \text{wenn ein Anfangssegment der Dezimalentwicklung von } \pi \\ & \text{(unter Ignorierung des Punktes) identisch mit } x \text{ ist,} \\ 0 & \text{sonst} \end{cases}$$

$$g(x) = \begin{cases} 1 & \text{wenn ein beliebiges Teilsegment der Dezimalentwicklung} \\ & \text{von } \pi \text{ identisch mit der Zahl } x \text{ ist,} \\ 0 & \text{sonst} \end{cases}$$

$$h(x) = \begin{cases} 1 & \text{wenn in der Dezimalentwicklung von } \pi \text{ mindestens} \\ & x \text{ aufeinanderfolgende Neunen vorkommen,} \\ 0 & \text{sonst} \end{cases}$$

$$\pi = 3.14159265358978984519916502904379740357398986 8\ldots$$

Keine Aussage möglich ohne Präzisierung des Begriffs

Es gibt viele Modelle für Berechenbarkeit ... lange bevor es die ersten Computer gab

- **Turingmaschine*** (Rechnen mit Papier und Bleistift)
- **Nichtdeterministische Turingmaschine*** (Parallelismus/Quantenrechner)
- **μ-rekursive Funktionen*** (Mathematisches Rechnen)
- **λ-Kalkül*** (Funktionale Sprachen, LISP)
- **Logische Repräsentierbarkeit*** (Logikprogrammierung, PROLOG)
- **Markov-Algorithmen (Typ-0 Grammatiken)** (Regelbasierte Sprachen)
- **Abakus** (Das älteste mechanische Hilfsmittel)
- **PASCAL-reduziert** (Imperative höhere Sprachen)
- **Registermaschine** (Assembler-/Maschinenprogrammierung)

Viele Formalisierungen eines intuitiven Begriffes

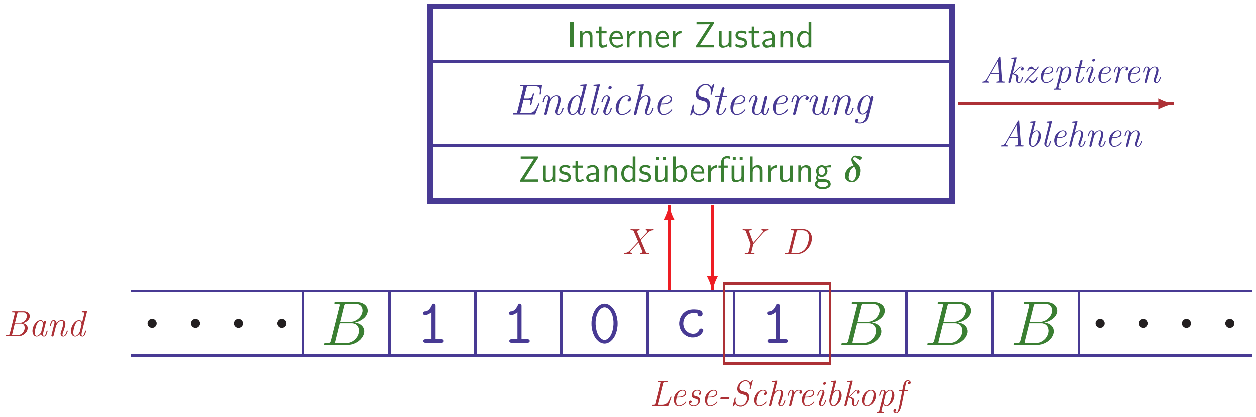
Theoretische Informatik II

Einheit 4.2

Turing-Berechenbarkeit

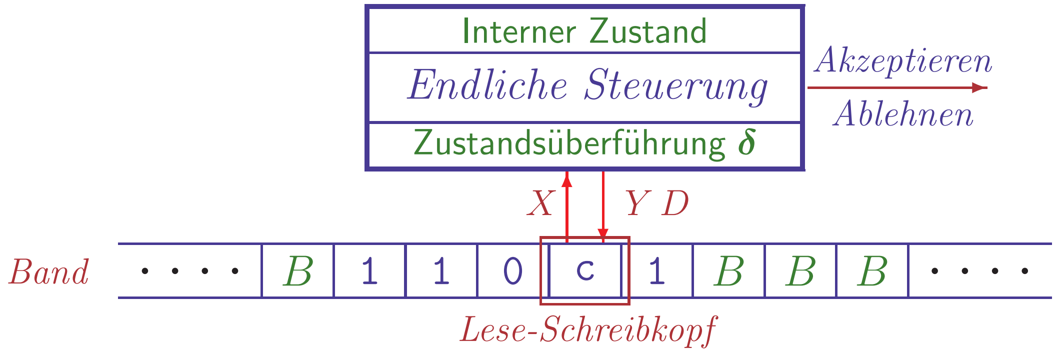
1. Rückblick: Turingmaschinen und Sprachen
2. Turing-berechenbare Funktionen
3. Zusammenhang: Berechnen vs. Akzeptieren

Rückblick: Turingmaschinen intuitiv



- **Endlicher Automat + lineares Band**
  - Endliche Steuerung liest Eingabesymbole
  - Gleichzeitig wird Bandsymbol unter Lese-Schreibkopf gelesen
- **Vereinfachung: keine separate Eingabe**
  - Eingabewort steht zu Anfang bereits auf dem Band
- **Einfacher Verarbeitungsmechanismus**
  - Bandsymbol X wird gelesen
  - Interner Zustand q wird zu q' verändert
  - Neues Symbol Y wird auf das Band geschrieben
  - Kopf wird in eine Richtung D (rechts oder links) bewegt

Rückblick: Turingmaschinen präzisiert



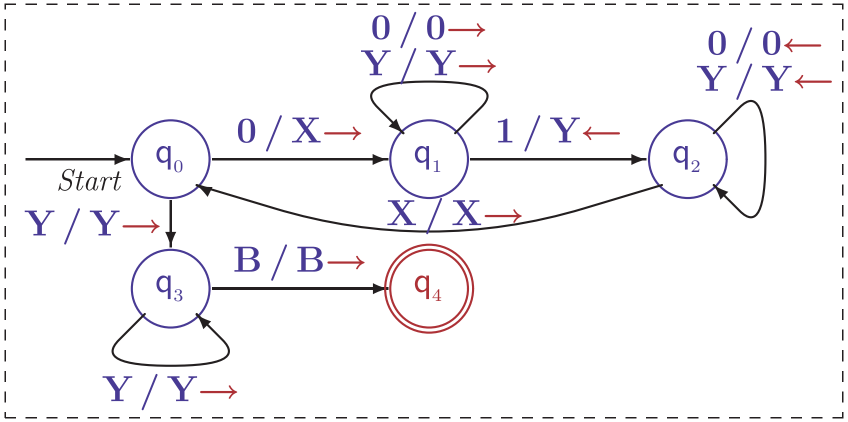
Eine **Turingmaschine** ist ein 7-Tupel

$\boldsymbol{M} = (\boldsymbol{Q}, \boldsymbol{\Sigma}, \boldsymbol{\Gamma}, \boldsymbol{\delta}, \boldsymbol{q_0}, \boldsymbol{B}, \boldsymbol{F})$ mit

- Q nichtleere endliche **Zustandsmenge**
- Σ endliches **Eingabealphabet**
- $\Gamma \supseteq \Sigma$ endliches **Bandalphabet**
- $\delta: Q \times \Gamma \to Q \times \Gamma \times \{L, R\}$ (partielle) **Überführungsfunktion**
- $q_0 \in Q$ **Startzustand**
- $B \in \Gamma \backslash \Sigma$ **Leersymbol des Bands** (“blank”)
- $F \subseteq Q$ Menge von **akzeptierenden** (End-)**Zuständen**

Rückblick: Beschreibung von Turingmaschinen

- **Übergangsdiagramme**
  - Zustände durch Knoten dargestellt
  - q_0 markiert durch *Start*-Pfeil,
    Endzustände durch doppelte Kreise
  - Für $\delta(q, X) = (p, Y, D)$ hat das
    Diagramm eine Kante $q \xrightarrow{X/Y\,D} p$
  - Σ und Γ implizit durch Diagramm bestimmt, Leersymbol heißt B



- **Übergangstabellen**
  - Funktionstabelle für δ
    — heißt "δ nicht definiert"
  - Pfeil → kennzeichnet q_0
  - Stern * kennzeichnet F
  - Σ, Γ und B implizit bestimmt

$Q \setminus \Gamma$	0	1	X	Y	B
→ q_0	(q_1,X,R)	—	—	(q_3,Y,R)	—
q_1	$(q_1,0,R)$	(q_2,Y,L)	—	(q_1,Y,R)	—
q_2	$(q_2,0,L)$	—	(q_0,X,R)	(q_2,Y,L)	—
q_3	—	—	—	(q_3,Y,R)	(q_4,B,R)
* q_4	—	—	—	—	—

- **Konvention:** $\delta(q,X)$ undefiniert für Endzustände $q \in F$

Rückblick: Arbeitsweise von Turingmaschinen

- **Erweiterter Begriff der Konfiguration**
  - Zustand q, Inhalt des Bandes und Kopfposition
  - Formal dargestellt als Tripel $\boldsymbol{K} = (\boldsymbol{u},\boldsymbol{q},\boldsymbol{v}) \in \Gamma^*\times Q\times\Gamma^+$
    - u, v: String links/rechts vom Kopf
  - Nur der bereits 'besuchten' Teil des Bandes wird betrachtet

    Blanks am Anfang von u oder am Ende von v entfallen, wo möglich

    Achtung: im Buch wird das Tripel als ein (!) String $\boldsymbol{uqv}$ geschrieben

- **Konfigurationsübergangsrelation $\vdash^*$**
  - $(\boldsymbol{uZ}, \boldsymbol{q}, \boldsymbol{Xv}) \vdash (\boldsymbol{u}, \boldsymbol{p}, \boldsymbol{ZYv})$, falls $\delta(q,X) = (p,Y,L)$
  - $(\boldsymbol{u}, \boldsymbol{q}, \boldsymbol{Xv}) \vdash (\boldsymbol{uY}, \boldsymbol{p}, \boldsymbol{v})$, falls $\delta(q,X) = (p,Y,R)$

  Sonderfälle
  - $(\boldsymbol{\epsilon}, \boldsymbol{q}, \boldsymbol{Xv}) \vdash (\boldsymbol{\epsilon}, \boldsymbol{p}, \boldsymbol{BYv})$, falls $\delta(q,X) = (p,Y,L)$
  - $(\boldsymbol{uZ}, \boldsymbol{q}, \boldsymbol{X}) \vdash (\boldsymbol{u}, \boldsymbol{p}, \boldsymbol{Z})$, falls $\delta(q,X) = (p,B,L)$
  - $(\boldsymbol{u}, \boldsymbol{q}, \boldsymbol{X}) \vdash (\boldsymbol{uY}, \boldsymbol{p}, \boldsymbol{B})$, falls $\delta(q,X) = (p,Y,R)$
  - $(\boldsymbol{\epsilon}, \boldsymbol{q}, \boldsymbol{Xv}) \vdash (\boldsymbol{\epsilon}, \boldsymbol{p}, \boldsymbol{v})$, falls $\delta(q,X) = (p,B,R)$

  $\boldsymbol{K_1} \vdash^* \boldsymbol{K_2}$, falls $K_1{=}K_2$ oder es gibt ein K mit $K_1 \vdash K$ und $K \vdash^* K_2$

Rückblick: Sprache einer Turingmaschine

- **Akzeptierte Sprache**
  - Menge der Eingaben, für die $\vdash^*$ zu akzeptierendem Zustand führt

    $$L(M) = \{w \in \Sigma^* \mid \exists p \in F.\, \exists u, v \in \Gamma^*.\ (\epsilon, q_0, w) \vdash^* (u, p, v)\}$$

  - Bei Einhalten der Konvention hält M im akzeptierenden Zustand an

- **Semi-entscheidbare Sprache**
  - Sprache, die von einer Turingmaschine M akzeptiert wird
  - Alternative Bezeichnung: **(rekursiv) aufzählbare Sprache**
  - Äquivalent zu Typ-0 Sprachen

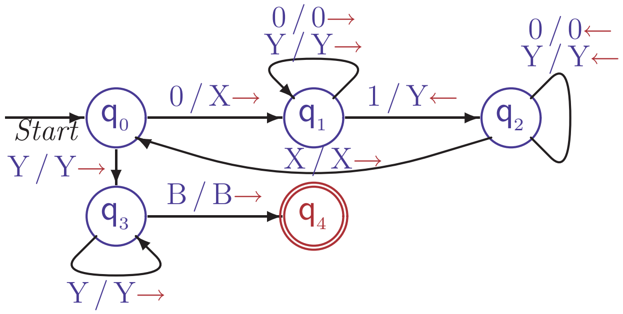
- **Entscheidbare Sprache**
  - Sprache, die von einer Turingmaschine M akzeptiert wird, die bei jeder Eingabe terminiert
  - Alternative Bezeichnung: **rekursive Sprache**

Zeit- und Platzbedarf von Turingmaschinen

- **Rechenzeit $t_M(w)$**
  - Anzahl der Konfigurationsübergänge bis M bei Eingabe w anhält
- **Speicherbedarf $s_M(w)$**
  - Anzahl der Bandzellen, die M während der Berechnung aufsucht
- **Komplexität: Bedarf relativ zur Größe**
  - $T_M(n) = \max\{t_M(w) \mid |w|=n\}$
  - $S_M(n) = \max\{s_M(w) \mid |w|=n\}$

  Maximaler Bedarf relativ zur Länge eines Eingabewortes (worst-case)
  - Die Größenordnung der Funktionen (linear, quadratisch, kubisch,...) ist aussagekräftiger als die genauen Werte

  $\mapsto$ Komplexitätstheorie
- **Komplexität der Turingmaschine für $\{0^n1^n | n{\geq}1\}$**
  - Zeitaufwand für Schleife q_0, q_1, q_2, q_0: $2n$
  - Gesamter Zeitaufwand quadratisch $(2n^2)$
  - Platzbedarf nicht größer als die Eingabe
  - Lineare Speicherplatzkomplexität



Die berechnete Funktion einer Turingmaschine

- **Turingmaschinen berechnen Funktionen auf Σ^***
  - Eingabe der Funktion wird aufs Band geschrieben
  - Bandinhalt wird durch Abarbeitung des Programms verändert
  - Wenn Maschine anhält, kann Bandinhalt ausgegeben werden

  Die ursprünglich vorgesehene Verwendung von Turingmaschinen

- **Beschreibung mittels Konfigurationen**
  - Anfangskonfiguration: $\boldsymbol{\alpha(w)} := (\epsilon, q_0, w)$
  - Rechenzeit $\boldsymbol{t_M(w)} := \max\{j \mid \alpha(w) \vdash^j (u,q,Xv) \wedge \delta(q,X) \text{ undefiniert}\}$

    Rechenzeit ist undefiniert falls dieses Maximum nicht existiert (d.h. M hält nicht)
  - Ausgabefunktion: $\boldsymbol{\omega(u,q,v)} := \boldsymbol{v|_\Sigma}$ (längster Präfix von v der zu Σ^* gehört)

    Ausgabe beginnt unter dem Kopf bis ein Symbol nicht aus Σ erreicht wird
  - Berechnete Funktion: $\boldsymbol{f_M(w)} := \omega(\kappa)$, wobei $\alpha(w) \vdash^{t_M(w)} \kappa$

    f_M: $\Sigma^* \to \Sigma^*$ ist wohldefiniert, da κ durch $\alpha(w) \vdash^{t_M(w)} \kappa$ eindeutig bestimmt

Berechnung mit Turing-Maschinen am Beispiel

- $M_1 = (\{q_0,q_1,q_2\}, \{1\}, \{1,B\}, \delta_1, q_0, B, \{q_2\})$ mit

δ_1	1	B
$\rightarrow q_0$	$(q_0,1,R)$	$(q_1,1,L)$
q_1	$(q_1,1,L)$	(q_2,B,R)
$* \; q_2$	—	—

Abarbeitungsbeispiel: $(\epsilon,q_0,111) \vdash^8 (\epsilon,q_2,1111)$

Fügt am Ende eines Wortes $w \in \{1\}^*$ eine 1 an ("Bierdeckelmaschine")

- **Mathematische Analyse**:
  - Anfangskonfiguration: $\alpha(1^n) = (\epsilon,q_0,1^n)$,
  - Nachfolgekonfigurationen: $(\epsilon,q_0,1^n) \vdash (1,q_0,1^{n-1}) \vdash^{n-1} (1^n,q_0,B)$
    $\vdash (1^{n-1},q_1,11) \vdash^n (\epsilon,q_1,B1^{n+1}) \vdash (\epsilon,q_2,1^{n+1})$
  - Terminierung: $\min\{j \mid \alpha(w) \vdash^j (u,q,Xv) \wedge \delta(q,X) \text{ undefiniert}\} = 2n+2$
  - Ergebnis: $(\epsilon,q_0,1^n) \vdash^{2n+2} (\epsilon,q_2,1^{n+1})$
  - Ausgabefunktion: $\omega(\epsilon,q_2,1^{n+1}) = 1^{n+1}$

$\boxed{f_{M_1}(1^n) = 1^{n+1} \text{ für alle } n, \quad \text{Definitionsbereich } \{1\}^*, \quad \text{Wertebereich } \{1\}^+}$

Beispiele für Turing-Maschinen

- $M_2 = (\{q_0,q_1\}, \{1\}, \{1,B\}, \delta_2, q_0, B, \{q_1\})$

mit

δ_2	1	B
$\rightarrow q_0$	(q_0,B,R)	(q_1,B,L)
$* \; q_1$	—	—

Abarbeitungsbeispiel:

$(\epsilon,q_0,111) \vdash^4 (\epsilon,q_1,B)$

Löscht ein Wort vom Band: $f_{M_2}(w) = \epsilon$ für alle $w \in \{1\}^*$

- $M_3 = (\{q_0,q_1,q_2\}, \{1\}, \{1,B\}, \delta_3, q_0, B, \{q_2\})$

mit

δ_3	1	B
$\rightarrow q_0$	$(q_1,1,R)$	(q_2,B,R)
q_1	$(q_0,1,R)$	(q_1,B,R)
$* \; q_2$	—	—

Abarbeitungsbeispiele:

$(\epsilon,q_0,1111) \vdash^5 (1111B,q_2,B)$

$(\epsilon,q_0,111) \vdash^n (111BBB...BB,q_1,B)$

Testet, ob Anzahl der Einsen in $w \in \{1\}^*$ gerade ist

$$f_{M_3}(1^n) = \begin{cases} \epsilon & \text{falls } n \text{ gerade,} \\ \perp & \text{sonst} \end{cases}$$

($\perp$ steht für "undefiniert")

Beispiele für Turing-Maschinen II

- $M_4 = (\{q_0,q_1,q_2,q_3,q_4\}, \{1\}, \{1,B\}, \delta_4, q_0, B, \{q_4\})$

  mit

δ_4	1	B
$\rightarrow q_0$	$(q_0,1,R)$	(q_1,B,L)
q_1	(q_2,B,R)	(q_4,B,R)
q_2	$(q_2,1,R)$	$(q_3,1,L)$
q_3	$(q_3,1,L)$	$(q_1,1,L)$
$* q_4$	—	—

  Abarbeitungsbeispiel:

  $(\epsilon,q_0,11) \vdash^{14} (\epsilon,q_4,1111)$

  Verdoppelt Anzahl der Einsen: $f_{M_4}(1^n) = 1^{2n}$

- $M_5 = (\{q_0,q_1,q_2,q_3\}, \{0,1\}, \{0,1,B\}, \delta_5, q_0, B, \{q_3\})$

  mit

δ_5	0	1	B
$\rightarrow q_0$	$(q_0,0,R)$	$(q_0,1,R)$	(q_1,B,L)
q_1	$(q_2,1,L)$	$(q_1,0,L)$	$(q_2,1,L)$
q_2	$(q_2,0,L)$	$(q_2,1,L)$	(q_3,B,R)
$* q_3$	—	—	—

  Abarbeitungsbeispiel:

  $(\epsilon,q_0,10011) \vdash^{12} (\epsilon,q_3,10100)$

  Addiert 1 auf die Binärdarstellung einer natürlichen Zahl

Turing-Berechenbare Funktionen

- $\boldsymbol{f{:}\Sigma^*{\rightarrow}\Delta^*}$ **Turing-berechenbar**
  – $f{=}f_M$ für eine Turingmaschine $M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$ mit $\Delta{\subseteq}\Gamma$
  $\mathcal{T}$: Menge der Turing-berechenbaren Funktionen

- **Berechenbarkeit auf Zahlen:** $\boldsymbol{f{:}\mathbb{N}{\rightarrow}\mathbb{N}}$
  $\hat{=}$ Berechenbarkeit von $f_r{:}\Sigma^*{\rightarrow}\Sigma^*$ mit $f_r(w) = r(f(r^{-1}(w)))$
  wobei $r{:}\mathbb{N}{\rightarrow}\Sigma^*$ injektive Repräsentation von Zahlen durch Wörter
  · unäre Darstellung $r_u{:}\mathbb{N}{\rightarrow}\{\mathtt{1}\}^*$ mit $r_u(n) = \mathtt{1}^n$
  · binäre Codierung $r_b{:}\mathbb{N}{\rightarrow}\{\mathtt{0},\mathtt{1}\}^*$ (ohne führende Nullen)

Berechenbarkeit auf anderen Mengen analog

Berechenbarkeit arithmetischer Funktionen

- **Nachfolgerfunktion $s:\mathbb{N}\to\mathbb{N}$ mit $s(n) = n+1$**
  - Bei unärer Codierung: berechne $s_u:\{1\}^*\to\{1\}^*$ mit $s_u(1^n) = 1^{n+1}$
    - Turingmaschine muß eine 1 anhängen: $s_u = f_{M_1}$
  - Bei binärer Codierung: $s_b = f_{M_5}$
    - M muß Ziffern von rechts beginnend umwandeln, ggf. mit Übertrag
- **Division durch 2: $div_2:\mathbb{N}\to\mathbb{N}$ mit $div_2(n)=\lfloor n/2\rfloor$**
  - Bei unärer Codierung: M muß (analog zu M_4) je zwei Einsen löschen und eine neue hinter dem Ende des Wortes schreiben
  - Bei binärer Codierung: M muß nur die letzte Ziffer löschen

Komplexere arithmetische Operationen benötigen Programmiertechniken für Turingmaschinen

Rückblick: Programmiertechniken für Turingmaschinen

- **Datenregister speichern Werte aus Menge Δ**
  – Simulation durch erweiterte Zustandsmenge $Q' := Q \times \Delta^k$
- **Mehrspur-Maschinen mit k Datenspuren**
  – Simulation durch erweitertes Bandalphabet $\Sigma' := \Sigma^k$
- **Mehrband-Maschinen**
  – Simulation von k unabhängigen Bändern mit $2k+1$ Spuren
  – Je eine Spur für Kopfposition eines Bandes + Endmarker für Suche
- **Unterprogramme**
  – Simulation wie bei Unterprogrammen in Assemblersprachen
- **Beschränkte Modelle für Beweise**
  – Halbseitig unendliches Band kann beidseitiges Band simulieren
  – Jedes Alphabet kann über Bandalphabet $\Gamma = \{1, \mathrm{B}\}$ codiert werden
  – 2 Stacks können jede Konfiguration einer Turingmaschine simulieren

Genauso leistungsfähig wie konventionelle Computer

Akzeptieren oder Berechnen?

- **Jede Funktion ist als Menge beschreibbar**
  – **graph(f)** $= \{(x,y) \mid f(x) = y\}$
  – Akzeptierende Maschinen erkennen Graphen berechenbarer Funktionen

  Satz: **f berechenbar $\Leftrightarrow$ graph(f) semi-entscheidbar**

- **Jede Menge ist als Funktion beschreibbar**
  – $\boldsymbol{\chi_L(w)} = \begin{cases} 1 & \text{falls } w \in L, \\ 0 & \text{sonst} \end{cases}$ Charakteristische Funktion der Sprache L

  – $\boldsymbol{\psi_L(w)} = \begin{cases} 1 & \text{falls } w \in L, \\ \perp & \text{sonst} \end{cases}$ Partiell-charakteristische Funktion von L

  – Charakteristische Funktionen erkannter Sprachen sind berechenbar

  Satz: **L semi-entscheidbar $\Leftrightarrow$ ψ_L berechenbar**
  L entscheidbar $\Leftrightarrow$ χ_L berechenbar

Akzeptieren vs. Berechnen: Beweisideen

Simuliere Abarbeitung der jeweils anderen Maschine

- **f berechenbar $\Leftrightarrow$ graph(f) semi-entscheidbar**

  $\Rightarrow$: Bei Eingabe (w, v) "teste" ob $f(w) = v$ ergibt

  $\Leftarrow$: Bei Eingabe w suche das "erste" Wort v mit $(w, v) \in \text{graph}(f)$

  Suche muß Werte für w,v und Rechenzeitgrenze simultan durchlaufen !!

  Maschinen müssen nicht bei jeder Eingabe anhalten

- **L semi-entscheidbar $\Leftrightarrow$ ψ_L berechenbar**

  $\Rightarrow$: Bei Eingabe w "teste" ob w akzeptiert wird und gebe ggf. 1 aus

  $\Leftarrow$: Bei Eingabe w "teste" ob $\psi_L(w) = 1$ ergibt

  Maschinen müssen nicht bei jeder Eingabe anhalten

- **L entscheidbar $\Leftrightarrow$ χ_L berechenbar**

  – Wie oben, aber beide Maschinen müssen bei jeder Eingabe anhalten

Turingmaschinen im Rückblick

- **Allgemeinstes Maschinenmodell**
  - Deterministischer endlicher Automat mit unendlichem Speicherband
  - "Beliebiger" Zugriff auf Speicherzellen
  - Erkennung von Wörtern durch Endzustand
  - Berechnen von Werten durch Ausgabe nach Terminierung
  - Beide Modelle sind gleich mächtig
- **Verhaltensanalyse durch Konfigurationsübergänge**
  - Konfigurationen beschreiben Zustand, Bandinhalt & Kopfposition
- **Äquivalent zu realen Computern**
  - Register, mehrere Bänder, Unterprogramme, etc. simulierbar
- **Äquivalent zu Typ-0 Grammatiken**
  - Umwandlung von Konfigurationsübergängen in Regeln und umgekehrt
- **Nichtdeterministische Variante ist gleich stark**
  - Simulationsaufwand durch deterministische Maschine ist exponentiell