Überblick

Inhalt

1. Verschiedene (universelle) Berechnungsmodelle
2. Berechenbare Funktionen
3. Komplexitätstheorie

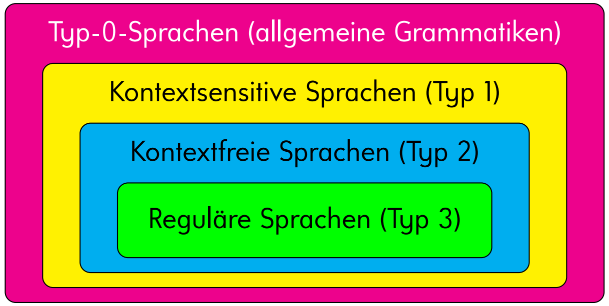
Plakative Fragestellungen

1. Was ist "Berechnung"? (Algorithmus)
2. Was ist "berechenbar"?
3. Wie teuer ist "Berechnung"? (Dimensionen: Zeit und Speicher)

Chomsky-Hierarchie

Noam Chomsky (* 1928)

- Amer. Linguist & Philosoph
- Verfechter Präzision
- Einführung Chomsky-Hierarchie



Chomsky-Grammatik (VL Automaten & Sprachen)

§1.1 Definition (Grammatik; *grammar*)

Grammatik ist Tupel (N, Σ, S, P)

1. endliche Menge N von **Nichtterminalen** *(nonterminals)*
2. endliche Menge Σ von **Terminalen** *(terminals)* mit $N \cap \Sigma = \emptyset$
3. **Startnichtterminal** $S \in N$ *(initial nonterminal)*
4. endliche Menge P von **Produktionen** *(productions)* der Form $\ell \to r$ mit $\ell \in (N \cup \Sigma)^+ \setminus \Sigma^+$ und $r \in ((N \setminus \{S\}) \cup \Sigma)^*$

Notizen

- Linke Produktionsseite ℓ = Sequenz Nichtterminale & Terminale (mind. 1 Nichtterminal)
- Rechte Produktionsseite r = Sequenz Nichtterminale & Terminale (Startnichtterminal darf nicht vorkommen)

Chomsky-Grammatik (VL Automaten & Sprachen)

§1.2 Definition (kontextsensitiv, kontextfrei, regulär)

Grammatik (N, Σ, S, P) ist

- **kontextsensitiv** *(context-sensitive)* falls $|\ell| \leq |r|$,
- **kontextfrei** *(context-free)* falls $\ell \in N$ und $r \neq \varepsilon$, und
- **regulär** *(regular)* falls $\ell \in N$ und $r \in (\Sigma \times N) \cup \Sigma$

für jede Produktion $(\ell \to r) \in P \setminus \{S \to \varepsilon\}$

§1.3 Beispiel

Grammatik $G = (\{S, S'\}, \{a, b\}, S, P)$ mit Produktionen P

$$S \to \varepsilon \quad S \to S' \quad S' \to aS'a \quad S' \to bS'b \quad S' \to aa \quad S' \to bb$$

kontextsensitiv	kontextfrei	regulär
✓	✓	✗

Chomsky-Grammatik

§1.4 Beispiel

Grammatik $G = (\{S, S', A, B, E\}, \{a, b\}, S, P)$ mit Produktionen P

$$\begin{array}{llll} S \to S'E & S' \to aS'a & S' \to bS'b & S' \to E \\ Ea \to EA & Aa \to aA & Ab \to bA & AE \to Ea \\ Eb \to EB & Ba \to aB & Bb \to bB & BE \to Eb \\ EE \to \varepsilon & & & \end{array}$$

kontextsensitiv	kontextfrei	regulär
✗	✗	✗

Semantik (VL Automaten & Sprachen)

§1.5 Definition (Ableitungsschritt; *derivation step*)

Sei $G = (N, \Sigma, S, P)$ Grammatik
Ableitungsrelation $\Rightarrow_G \subseteq (N \cup \Sigma)^* \times (N \cup \Sigma)^*$ ist

$$\Rightarrow_G = \{(v\ell v', vrv') \mid (\ell \to r) \in P,\ v, v' \in (N \cup \Sigma)^*\}$$

Illustration

- Produktion $\ell \to r \in P$
- Ableitungsschritt $\cdots \ell \cdots \Rightarrow_G \cdots r \cdots$

Semantik

Beispiel (§1.3)

Kontextfreie Grammatik $G = (\{S, S'\}, \{a, b\}, S, P)$ mit Produktionen P

$$S \to \varepsilon \quad S \to S' \quad S' \to aS'a \quad S' \to bS'b \quad S' \to aa \quad S' \to bb$$

Ableitungsschritte

- Ableitung von $v = abbaabba$

$$S \Rightarrow_G S' \Rightarrow_G aS'a \Rightarrow_G abS'ba \Rightarrow_G abbS'bba \Rightarrow_G \underbrace{abbaabba}_{=v}$$

- Allgemein $S \Rightarrow_G^* ww^R$ für alle $w \in \{a, b\}^*$

Semantik

Beispiel (§1.4)

Grammatik $G = (\{S, S', A, B, E\}, \{a, b\}, S, P)$ mit Produktionen P

$$\begin{array}{llll} S \to S'E & S' \to aS'a & S' \to bS'b & S' \to E \\ Ea \to EA & Aa \to aA & Ab \to bA & AE \to Ea \\ Eb \to EB & Ba \to aB & Bb \to bB & BE \to Eb \\ EE \to \varepsilon & & & \end{array}$$

$$\begin{aligned} & S \Rightarrow_G S'E \Rightarrow_G aS'aE \Rightarrow_G abS'baE \Rightarrow_G abEbaE \\ & \Rightarrow_G abEBaE \Rightarrow_G abEaBE \Rightarrow_G abEaEb \Rightarrow_G abEAEb \\ & \Rightarrow_G abEEab \Rightarrow_G ab\varepsilon ab = abab \end{aligned}$$

Allgemein $S \Rightarrow_G^* ww$ für alle $w \in \{a, b\}^*$

Erzeugte Sprache (VL Automaten & Sprachen)

§1.6 Definition (Erzeugte Sprache; *generated language*)

Sei $G = (N, \Sigma, S, P)$ Grammatik
Die von G **erzeugte Sprache** $L(G) \subseteq \Sigma^*$ ist

$$L(G) = \{w \in \Sigma^* \mid S \Rightarrow_G^* w\}$$

Beispiel (§1.3)

Kontextfreie Grammatik $G = \big(\{S, S'\}, \{a, b\}, S, P\big)$ mit Produktionen P

$$S \to \varepsilon \quad S \to S' \quad S' \to aS'a \quad S' \to bS'b \quad S' \to aa \quad S' \to bb$$

Erzeugte Sprache $L(G) = \{ww^R \mid w \in \{a, b\}^*\}$

Erzeugte Sprache

Beispiel (§1.4)

Grammatik $G = \big(\{S, S', A, B, E\}, \{a, b\}, S, P\big)$ mit Produktionen P

$$\begin{array}{llll} S \to S'E & S' \to aS'a & S' \to bS'b & S' \to E \\ Ea \to EA & Aa \to aA & Ab \to bA & AE \to Ea \\ Eb \to EB & Ba \to aB & Bb \to bB & BE \to Eb \\ EE \to \varepsilon & & & \end{array}$$

Erzeugte Sprache $L(G) = \{ww \mid w \in \{a, b\}^*\}$

Sprachklassen (VL Automaten & Sprachen)

§1.7 Definition (Sprachklassen; *language classes*)

Sprache $L \subseteq \Sigma^*$ ist

- **regulär** *(regular)*,
  falls reguläre Grammatik G mit $L(G) = L$ existiert
- **kontextfrei** *(context-free)*,
  falls kontextfreie Grammatik G mit $L(G) = L$ existiert
- **kontextsensitiv** *(context-sensitive)*,
  falls kontextsensitive Grammatik G mit $L(G) = L$ existiert

Notizen

- Sprache regulär falls erzeugbar von regulärer Grammatik
- Analog für weitere Sprachklassen

Sprachklassen

Beispiel (§1.3)

Kontextfreie Grammatik $G = \big(\{S, S'\}, \{a, b\}, S, P\big)$ mit Produktionen P

$$S \to \varepsilon \quad S \to S' \quad S' \to aS'a \quad S' \to bS'b \quad S' \to aa \quad S' \to bb$$

Erzeugte Sprache $L(G) = \{ww^R \mid w \in \{a, b\}^*\}$

- Frage: Ist Sprache $L(G)$ kontextfrei?
- Antwort: Ja, denn kontextfreie Grammatik G erzeugt $L(G)$

Sprachklassen

Beispiel (§1.4)

Grammatik $G = \big(\{S, S', A, B, E\}, \{a, b\}, S, P\big)$ mit Produktionen P

$$
\begin{array}{cccc}
S \to S'E & S' \to aS'a & S' \to bS'b & S' \to E \\
Ea \to EA & Aa \to aA & Ab \to bA & AE \to Ea \\
Eb \to EB & Ba \to aB & Bb \to bB & BE \to Eb \\
EE \to \varepsilon & & &
\end{array}
$$

Erzeugte Sprache $L(G) = \{ww \mid w \in \{a, b\}^*\}$

- Frage: Ist $L(G)$ nicht kontextsensitiv, da G nicht kontextsensitiv?
- Antwort: Nein, nur falls keine kontextsen. Grammatik $L(G)$ erzeugt

Reguläre Sprachen (VL Automaten & Sprachen)

Beschreibung

- Reguläre Grammatik
- Endlicher Automat (nichtdeterministisch oder deterministisch)
- Regulärer Ausdruck

Stichworte

- Normalformen, Determinisierung & Minimierung
- Abschluss- & Entscheidbarkeitsresultate
- Pumping-Lemma

Kontextfreie Sprachen (VL Automaten & Sprachen)

Beschreibung

- Kontextfreie Grammatik
- Kellerautomat (nichtdeterministisch)
- Deterministischer Kellerautomat (strikt schwächer)

Stichworte

- Normalformen & Parsing-Algorithmen
- Abschluss- & Entscheidbarkeitsresultate
- Pumping-Lemma

Kontextsensitive Sprachen

Beschreibung

- Kontextsensitive Grammatik
- Linear beschränkte Turingmaschine (nichtdeterministisch)
- Linear beschränkte det. Turingmaschine (Mächtigkeit unklar)

Stichworte

- Normalformen & Parsing-Algorithmus
- Abschluss- & Entscheidbarkeitsresultate

Typ-0-Sprachen

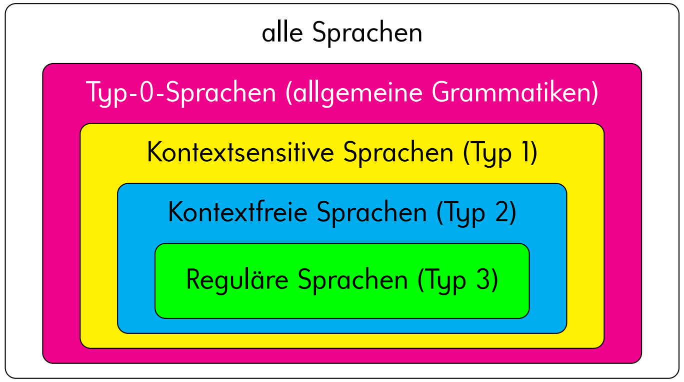
Beschreibung

- Chomsky-Grammatik
- Turingmaschine (nichtdeterministisch oder deterministisch)
- While-Programm, μ-rekursive Funktion (berechenbare Funktion)

Stichworte

- Normalformen & Determinisierung
- Abschluss- & Entscheidbarkeitsresultate

Ausblick — Chomsky-Sprachklassen



$$\text{Typ-3}(\Sigma) \subsetneq \text{Typ-2}(\Sigma) \subsetneq \text{Typ-1}(\Sigma) \subsetneq \text{Typ-0}(\Sigma) \subsetneq \mathcal{P}(\Sigma^*)$$

Ausblick — Chomsky-Sprachklassen

Eigenschaften

- Es gibt Sprachen, die nicht Typ-0 sind
- Es gibt Typ-0-Sprachen mit unentscheidbarem Wortproblem

§1.8 Definition (abzählbar; *countable* — VL Diskrete Strukturen)

Menge M ist **abzählbar** falls injektive Funktion $f\colon M \to \mathbb{N}$ existiert

Notizen

- M abzählbar gdw. jedem $m \in M$ eigene natürliche Zahl zuweisbar
- Natürliche Zahlen $\mathbb{N} = \{0, 1, 2, 3, \dots\}$ abzählbar

Abzählbarkeit

§1.9 Theorem (Abzählbarkeit von $\mathbb{N}^k$)

Menge $\mathbb{N}^k$ abzählbar für alle $k \in \mathbb{N}$

Beweis

Seien $p_1, \dots, p_k$ verschiedene Primzahlen. Definiere $f\colon \mathbb{N}^k \to \mathbb{N}$ durch

$$f(n_1, \dots, n_k) = \prod_{i=1}^{k} p_i^{n_i} = p_1^{n_1} \cdots p_k^{n_k} \qquad \text{für alle } n_1, \dots, n_k \in \mathbb{N}$$

Falls $f(m_1, \dots, m_k) = f(n_1, \dots, n_k)$ für $m_1, \dots, m_k, n_1, \dots, n_k \in \mathbb{N}$, dann $m_i = n_i$ für alle $1 \leq i \leq k$ da Primfaktorenzerlegung eindeutig. Also ist f injektiv und $\mathbb{N}^k$ damit abzählbar. $\square$

Abzählbarkeit

§1.10 Theorem (Abzählbarkeit von $\mathbb{N}^*$)

Menge $\mathbb{N}^* = \bigcup_{k \in \mathbb{N}} \mathbb{N}^k$ abzählbar

Beweis

Menge $\mathbb{N}^k$ abzählbar via injektiver Funktion $f_k \colon \mathbb{N}^k \to \mathbb{N}$ für alle $k \in \mathbb{N}$ nach Thm §1.9. Sei $f \colon \mathbb{N}^* \to \mathbb{N}$ gegeben durch

$$f(w) = f_2(|w|, f_{|w|}(w)) \qquad \text{für alle } w \in \mathbb{N}^*$$

Falls $f(w) = f(w')$ für $w, w' \in \mathbb{N}^*$, dann folgen aus Injektivität von f_2 sowohl $|w| = |w'|$ als auch $f_{|w|}(w) = f_{|w'|}(w')$. Weiterhin folgt $w = w'$ aus Injektivität von $f_{|w|} = f_{|w'|}$. Also ist f injektiv und $\mathbb{N}^*$ abzählbar. □

Abzählbarkeit der Grammatiken

1. Kodiere Terminale durch ungerade Zahlen ($a = 1$; $b = 3$)
2. Gerade positive Zahlen für Nichtterminale ($S = 2$; $S' = 4$; $\dots$) (beginnend mit Startnichtterminal)
3. 0 als Trennzeichen

Beispiel (§1.4)

Grammatik $G = \big(\{S, S', A, B, E\}, \{a, b\}, S, P\big)$ mit Produktionen P

$$\begin{array}{cccc}
S \to S'E & S' \to aS'a & S' \to bS'b & S' \to E \\
Ea \to EA & Aa \to aA & Ab \to bA & AE \to Ea \\
Eb \to EB & Ba \to aB & Bb \to bB & BE \to Eb \\
EE \to \varepsilon & & &
\end{array}$$

$$c(G) = \underbrace{2.0.4.10}_{S \to S'E}.0.\underbrace{4.0.1.4.1}_{S' \to aS'a}.0.\underbrace{4.0.3.4.3}_{S' \to bS'b}.0.\cdots$$

Abzählbarkeit Typ-0-Sprachen

§1.11 Theorem (Abzählbarkeit Typ-0-Sprachen)

Typ-0-Sprachen $\text{Typ-0}(\Sigma) \subseteq \mathcal{P}(\Sigma^*)$ über Alphabet Σ abzählbar

Beweis (nutzt Auswahlaxiom)

Jede Grammatik G kann als Element $c(G) \in \mathbb{N}^*$ kodiert werden (c nicht injektiv). Sei Kodierung der Terminale aus Σ fest.

$$c \colon \{G \mid G \text{ Grammatik über } \Sigma\} \to \mathbb{N}^*$$

Für alle Grammatiken G und G' mit $c(G) = c(G')$ gilt $L(G) = L(G')$. Menge $C = \{c(G) \mid G \text{ Grammatik über } \Sigma\}$ abzählbar da $C \subseteq \mathbb{N}^*$ und $\mathbb{N}^*$ abzählbar gemäß Thm §1.10. Also ist Relation

$$\rho = \big\{(c(G), L(G)) \mid G \text{ Grammatik über } \Sigma\big\}$$

surjektive Funktion $\rho \colon C \to \text{Typ-0}(\Sigma)$. Mit Auswahlaxiom existiert $g \colon \text{Typ-0}(\Sigma) \to C$ injektiv (VL Diskrete Strukturen). Sei f aus Thm §1.10. Dann ist $(g\,;f) \colon \text{Typ-0}(\Sigma) \to \mathbb{N}$ injektiv und $\text{Typ-0}(\Sigma)$ abzählbar. □

Abzählbarkeit Typ-0-Sprachen

Notizen

- Thm §1.11 gilt auch ohne Auswahlaxiom
- Betrachte längen-lexikographische Ordnung $\preceq \subseteq \mathbb{N}^* \times \mathbb{N}^*$ auf $\mathbb{N}^*$ (ordne Elemente zunächst nach Länge, dann lexikographisch)
- Ordnung $\preceq$ total und Wohlordnung (für nichtleere Teilmenge $N \subseteq \mathbb{N}^*$ existiert $\min_{\preceq}(N) \in N$)
- Für $\rho \colon C \to \text{Typ-0}(\Sigma)$ surjektiv, definiere $\bar{\rho} \colon \text{Typ-0}(\Sigma) \to \mathbb{N}^*$

$$\bar{\rho}(L) = \min_{\preceq}\{w \in \mathbb{N}^* \mid \rho(w) = L\} \qquad \text{für alle } L \in \text{Typ-0}(\Sigma)$$

- $\bar{\rho}$ wohldefiniert, da $\{w \in \mathbb{N}^* \mid \rho(w) = L\}$ nichtleer (Surjektivität ρ) und somit existiert Minimum
- $\bar{\rho}$ ist offensichtlich injektiv (gleiche Kodierung $\to$ gleiche Sprache)
- Also $\text{Typ-0}(\Sigma)$ abzählbar

Überabzählbarkeit aller Sprachen

§1.12 Lemma

Unendliche Menge M abzählbar gdw. Bijektion $f\colon \mathbb{N} \to M$ existiert

§1.13 Theorem (Überabzählbarkeit aller Sprachen)

Menge $\mathcal{P}(\Sigma^*)$ aller Sprachen über Σ <u>nicht</u> abzählbar

Beweis

Σ^* abzählbar und unendlich. Cantors Theorem (VL Diskrete Strukturen) zeigt, dass $\mathcal{P}(\Sigma^*)$ strikt mächtiger als Σ^*. Also $\mathcal{P}(\Sigma^*)$ <u>nicht</u> abzählbar (d.h. überabzählbar). $\square$

Überabzählbarkeit aller Sprachen

Theorem (§1.13 Überabzählbarkeit aller Sprachen)

Menge $\mathcal{P}(\Sigma^*)$ aller Sprachen über Σ <u>nicht</u> abzählbar

Beweis (detailliert)

Da Σ^* abzählbar unendlich, existiert $f\colon \mathbb{N} \to \Sigma^*$ bijektiv gem. Lm §1.12. Offenbar ist $\mathcal{P}(\Sigma^*)$ unendlich. Sei $\mathcal{P}(\Sigma^*)$ abzählbar; d.h. gem. Lm §1.12 existiert $g\colon \mathbb{N} \to \mathcal{P}(\Sigma^*)$ bijektiv.

Betrachte Sprache $L = \{f(i) \mid i \in \mathbb{N},\ f(i) \notin g(i)\}$.

Da g bijektiv, existiert $i \in \mathbb{N}$ mit $L = g(i)$.
Dann $f(i) \in L$ gdw. $f(i) \notin g(i) = L$. Widerspruch ↯
Also $\mathcal{P}(\Sigma^*)$ <u>nicht</u> abzählbar. $\square$

Überabzählbarkeit aller Sprachen

Diagonalisierung

$L' \setminus w$	$f(0)$	$f(1)$	$f(2)$	$f(3)$	$\cdots$
$g(0)$	$\boxed{✗}$	✗	✓	✓	$\cdots$
$g(1)$	✗	$\boxed{✓}$	✓	✗	$\cdots$
$g(2)$	✗	✗	$\boxed{✓}$	✗	$\cdots$
$g(3)$	✓	✓	✗	$\boxed{✓}$	$\cdots$
$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$
L	✓	✗	✗	✗	$\cdots$

$$L = \{f(i) \mid i \in \mathbb{N},\ f(i) \notin g(i)\}$$

1. Hauptsatz

§1.14 Theorem

Nicht alle Sprachen sind Typ-0

Beweis

Typ-0-Sprachen $\text{Typ-0}(\Sigma)$ über Σ abzählbar gemäß Thm §1.11.
Menge $\mathcal{P}(\Sigma^*)$ aller Sprachen über Σ überabzählbar gemäß Thm §1.13.
Also $\text{Typ-0}(\Sigma) \subsetneq \mathcal{P}(\Sigma^*)$. $\square$

Kodierungen

§2.1 Definition (partielle Funktion; *partial function*)

Seien A, B Mengen. Relation $\rho \subseteq A \times B$ ist **partielle Funktion**, geschrieben $\rho\colon A \dashrightarrow B$, falls für jedes $a \in A$ höchstens ein $b \in B$ mit $(a, b) \in \rho$ existiert.

Notizen

- Übliche Funktionsschreibweisen auch für partielle Funktionen
- Jede Funktion ist partielle Funktion
- **Definitionsbereich** partieller Funktion $f\colon A \dashrightarrow B$ ist $f^{-1}(B)$
  (Elemente des Vorbereiches A, für die f definiert ist)

$$f^{-1}(B) = \{a \in A \mid \exists b \in B\colon f(a) = b\}$$

- $f^{-1}(B) = A$ für jede Funktion $f\colon A \to B$

Kodierungen

Vereinbarungen

- Beschränkung auf partielle Funktionen

$$f\colon \mathbb{N}^k \dashrightarrow \mathbb{N} \qquad \text{und} \qquad g\colon \Sigma^* \dashrightarrow \Delta^* \qquad \text{(für Alphabete } \Sigma, \Delta)$$

- 2 Kodierungen für natürliche Zahlen
  - **Unäre** Kodierung: $n \in \mathbb{N}$ repräsentiert durch $a^n = \underbrace{a \cdots a}_{n \text{ mal}}$

    Aus $f\colon \mathbb{N}^k \dashrightarrow \mathbb{N}$ wird $g\colon \{a, \#\}^* \dashrightarrow \{a\}^*$ mit
    $$g(a^{n_1} \# a^{n_2} \# \cdots \# a^{n_k}) = a^{f(n_1, \dots, n_k)}$$

  - **Binäre** Kodierung: $n \in \mathbb{N}$ repräsentiert durch $\text{bin}(n) \in \{0,1\}^*$

    Aus $f\colon \mathbb{N}^k \dashrightarrow \mathbb{N}$ wird $g\colon \{0, 1, \#\}^* \to \{0,1\}^*$ mit
    $$g(\text{bin}(n_1) \# \text{bin}(n_2) \# \cdots \# \text{bin}(n_k)) = \text{bin}(f(n_1, \dots, n_k))$$

Kodierungen

Kodierung von $f(3, 4) = 7$

- Unäre Kodierung

$$g(\underbrace{aaa}_{3} \# \underbrace{aaaa}_{4}) = \underbrace{aaaaaaa}_{7}$$

- Binäre Kodierung

$$g(\underbrace{11}_{2+1} \# \underbrace{100}_{4+0+0}) = \underbrace{111}_{4+2+1}$$

- Andere berechenbare Kodierungen auch möglich

  Dezimalkodierung: $\quad g\colon \{0, 1, \dots, 9, \#\}^* \dashrightarrow \{0, 1, \dots, 9\}^*$

Kodierung von Sprachen

§2.2 Definition (Sprachenkodierung)

Für jede Sprache $L \subseteq \Sigma^*$ ist $\text{id}_L\colon \Sigma^* \dashrightarrow \Sigma^*$ gegeben durch

$$\text{id}_L = \{(w, w) \mid w \in L\}$$

Notizen

- 'undef' (oder $\perp$) steht für nicht definierte Funktionswerte
- Alternative Definition

$$\text{id}_L(w) = \begin{cases} w & \text{falls } w \in L \\ \text{undef} & \text{sonst} \end{cases}$$

- Also $\text{id}_L^{-1}(\Sigma^*) = L$

Intuitive Berechenbarkeit

Algorithmus = endliche & eindeutige Handlungsbeschreibung

§2.3 Definition (intuitive Berechenbarkeit; *computability*)

Funktion $f\colon \Sigma^* \dashrightarrow \Delta^*$ **intuitiv berechenbar** *(computable)*, falls Algorithmus A_f existiert, so dass für jede Eingabe $w \in \Sigma^*$

- A_f produziert Ergebnis nach endlicher Zeit gdw. $w \in f^{-1}(\Delta^*)$
- A_f produziert Ergebnis $f(w)$ falls $w \in f^{-1}(\Delta^*)$

Notizen

- $w \in f^{-1}(\Delta^*)$ bedeutet "$f(w)$ definiert"
- A_f muss bei Eingabe $w \in f^{-1}(\Delta^*)$ Ergebnis $f(w)$ liefern
- A_f darf bei Eingabe $w \in \Sigma^* \setminus f^{-1}(\Delta^*)$ kein Ergebnis liefern (Endlosschleife, Absturz, Exception, etc.)

Intuitive Berechenbarkeit

Weitere Notizen

- Mathematische Existenz ausreichend (kann Funktion 2 Formen annehmen, also $f = f_1$ oder $f = f_2$, dann reicht intuitive Berechenbarkeit von f_1 und f_2)
- Beschreibungssprache beliebig (C++, Java, Pseudocode, etc.)
- Hardware irrelevant (Architektur, Ablaufmechanismus, etc.)
- Keine Zeit- oder Speicherbeschränkung (aber A_f muss bei Eingabe $w \in f^{-1}(\Delta^*)$ letztlich terminieren)

Intuitive Berechenbarkeit

Erklärungsversuch

- E sei Eigenschaft der Welt und $f\colon \Sigma^* \dashrightarrow \Delta^*$ (z.B. E = Gültigkeit der Goldbachschen Vermutung)
- Weiterhin gelten $E \to \text{Berechenbar}(f)$ und $\neg E \to \text{Berechenbar}(f)$

$$
\begin{aligned}
& (E \to \text{Berechenbar}(f)) \wedge (\neg E \to \text{Berechenbar}(f)) \\
\equiv\; & (\neg E \vee \text{Berechenbar}(f)) \wedge (E \vee \text{Berechenbar}(f)) \\
\equiv\; & (\neg E \wedge E) \vee \text{Berechenbar}(f) \\
\equiv\; & \text{Berechenbar}(f)
\end{aligned}
$$

- Also gilt $\text{Berechenbar}(f)$

Intuitive Berechenbarkeit

- Addition: Funktion $+\colon \mathbb{N}^2 \to \mathbb{N}$ intuitiv berechenbar
  - Schulmethode
  - x_1 mal Erhöhung von x_2 für $x_1 + x_2$
- Format-Prüfung: Funktion $\text{id}_L\colon \{0,1,\#\}^* \dashrightarrow \{0,1,\#\}^*$ mit

$$L = \underbrace{1\,(0 \mid 1)^* (\#\, 1\,(0 \mid 1)^*)^*}_{\text{(1, beliebig viele 0 und 1, \# und weitere solche Blöcke)}}$$

intuitiv berechenbar (L regulär)

Intuitive Berechenbarkeit

$\pi[n]$ = erste n Stellen in Dezimalbruchdarstellung von π für alle $n \in \mathbb{N}$

$$\pi[3] = 314 \qquad \pi[6] = 314159 \qquad \pi[1] = 3$$

- Approximation π: Funktion $\pi\colon \{a\}^* \to \{0, 1, \dots, 9\}^*$ mit

$$\pi(a^n) = \pi[n] \qquad \text{für alle } n \in \mathbb{N}$$

intuitiv berechenbar
- Approximationsalgorithmus für π
- Ausgabe erste n Stellen sobald ausreichende Genauigkeit

Intuitive Berechenbarkeit

- Teilstrings von π: Funktion $\mathrm{sub}_\pi\colon \{0, 1, \dots, 9\}^* \longrightarrow \{0, 1\}^*$ mit

$$\mathrm{sub}_\pi(w) = \begin{cases} 1 & \text{falls } w \text{ in } \pi \text{ vorkommt} \\ 0 & \text{sonst} \end{cases} \qquad \text{für alle } w \in \{0, \dots, 9\}^*$$

Intuitive Berechenbarkeit unklar

Beispiele

$$\mathrm{sub}_\pi(314) = 1 \qquad \mathrm{sub}_\pi(15) = 1 \qquad \mathrm{sub}_\pi(41) = 1$$

Intuitive Berechenbarkeit

- Teilstrings von π: Funktion $\mathrm{sub}_\pi\colon \{0, 1, \dots, 9\}^* \dashrightarrow \{0, 1\}^*$ mit

$$\mathrm{sub}_\pi(w) = \begin{cases} 1 & \text{falls } w \text{ in } \pi \text{ vorkommt} \\ \text{undef} & \text{sonst} \end{cases} \qquad \text{für alle } w \in \{0, \dots, 9\}^*$$

intuitive Berechenbarkeit: intuitiv berechenbar

Beispiele:

$$\mathrm{sub}_\pi(314) = 1 \qquad \mathrm{sub}_\pi(15) = 1 \qquad \mathrm{sub}_\pi(41) = 1$$

Intuitive Berechenbarkeit

- Länge von Nichtteilstrings von π: Funktion $\ell_\pi\colon \mathbb{N} \dashrightarrow \mathbb{N}$ mit

$$\ell_\pi(n) = \begin{cases} n & \text{falls Sequenz der Länge } n \text{ existiert,} \\ & \text{die nicht in } \pi \text{ vorkommt} \\ \text{undef} & \text{sonst} \end{cases} \qquad \text{für alle } n \in \mathbb{N}$$

Intuitive Berechenbarkeit intuitiv berechenbar
- Falls alle Sequenzen in π vorkommen, (Eigenschaft E) dann ℓ_π überall undefiniert & intuitiv berechenbar
- Sonst existiert kürzeste Sequenz der Länge k, die nicht in π vorkommt & ℓ_π intuitiv berechenbar, da

$$\ell_\pi(n) = f_k(n) = \begin{cases} n & \text{falls } n \geq k \\ \text{undef} & \text{sonst} \end{cases}$$

$(\neg E \to \exists k((\ell_\pi = f_k) \land \text{Berechenbar}(f_k))$ also $\neg E \to \text{Berechenbar}(\ell_\pi))$

Intuitive Berechenbarkeit

- Wortproblem Sprache $L \subseteq \Sigma^*$: Funktion $\chi_L \colon \Sigma^* \to \{0,1\}^*$ mit

$$\chi_L(w) = \begin{cases} 1 & \text{falls } w \in L \\ 0 & \text{sonst} \end{cases} \qquad \text{für alle } w \in \Sigma^*$$

Intuitive Berechenbarkeit
- L kontextsensitiv: intuitiv berechenbar
- Typ-0-Sprache L: unklar/nicht intuitiv berechenbar

Intuitive Berechenbarkeit

- Aufzählung einer Sprache $L \subseteq \Sigma^*$: Funktion $\rho_L \colon \Sigma^* \dashrightarrow \{0,1\}^*$ mit

$$\rho_L(w) = \begin{cases} 1 & \text{falls } w \in L \\ \text{undef} & \text{sonst} \end{cases} \qquad \text{für alle } w \in \Sigma^*$$

intuitive Berechenbarkeit:
- für kontextsensitive Sprache L: intuitiv berechenbar
- für Typ-0-Sprache L: intuitiv berechenbar

Intuitive Berechenbarkeit

Problem
- Wie argumentiert man “nicht intuitiv berechenbar”?
  (muss für beliebige Algorithmen funktionieren)

Ansatz der modellbezogenen Berechenbarkeit
- Festlegung Berechnungsmodell (Grammatik, Turingmaschine, etc.)
- Klärt Begriff ‘Algorithmus’

Wiederholung: Chomsky-Grammatik

Beispiel (§1.4)

Grammatik $G = \big(\{S, S', A, B, E\}, \{a, b\}, S, P\big)$ mit Produktionen P

$$\begin{array}{cccc} S \to S'E & S' \to aS'a & S' \to bS'b & S' \to E \\ Ea \to EA & Aa \to aA & Ab \to bA & AE \to Ea \\ Eb \to EB & Ba \to aB & Bb \to bB & BE \to Eb \\ EE \to \varepsilon & & & \end{array}$$

Ableitungsschritte

$$\begin{aligned} & S \Rightarrow_G S'E \Rightarrow_G aS'aE \Rightarrow_G abS'baE \Rightarrow_G abEbaE \\ & \Rightarrow_G abEBaE \Rightarrow_G abEaBE \Rightarrow_G abEaEb \Rightarrow_G abEAEb \\ & \Rightarrow_G abEEab \Rightarrow_G ab\varepsilon ab = abab \end{aligned}$$

Wiederholung: Chomsky-Grammatik

Analyse der Funktionsweise

- Ziel ww mit $w \in \{a, b\}^*$
- Erzeuge zunächst $wEw^R E$

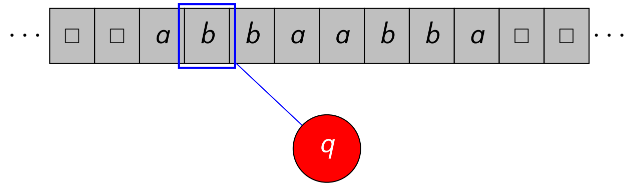
$$S \to S'E \qquad S' \to aS'a \qquad S' \to bS'b \qquad S' \to E$$

- Symbol hinter linkem E direkt hinter rechtes E bewegen

$$Ea \to EA \qquad Aa \to aA \qquad Ab \to bA \qquad AE \to Ea$$
$$Eb \to EB \qquad Ba \to aB \qquad Bb \to bB \qquad BE \to Eb$$

- Invertiert w^R; liefert w und Satzform $wEEw$
- Löschen Begrenzer EE mit Produktion $EE \to \varepsilon$

Turingmaschine

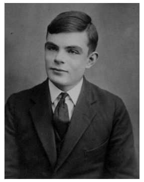


Notizen

- Beidseitig unbeschränktes Arbeitsband
- Endliche Kontrolle (zustandsgesteuert)
- Mobiler Lese- & Schreibkopf
- Eingabe auf Band; Symbole überschreibbar (Speicher)

Alan Turing (* 1912; † 1954)

- Engl. Informatiker
- Brach dtsch. Enigma-Verschlüsselung
- Verurteilt wegen Homosexualität; akzeptierte Kastration; 2013 offiziell rehabilitiert



Turingmaschine

§2.4 Definition (Turingmaschine; *Turing machine*)

Turingmaschine ist Tupel $M = (Q, \Sigma, \Gamma, \Delta, \Box, q_0, q_+, q_-)$

- endl. Menge Q von Zuständen *(states)* mit $Q \cap \Gamma = \emptyset$
- endl. Menge Σ von Eingabesymbolen *(input symbols)*
- endl. Menge Γ von Arbeitssymbolen *(work symbols)* mit $\Sigma \subseteq \Gamma$
- Übergangsrelation *(transition relation)*
  $\Delta \subseteq \Big(\big(Q \setminus \{q_+, q_-\}\big) \times \Gamma\Big) \times \Big(Q \times \Gamma \times \{\triangleleft, \triangleright, \diamond\}\Big)$
- Leersymbol *(blank)* $\Box \in \Gamma \setminus \Sigma$ $\quad (\Gamma_M = \Gamma \setminus \{\Box\})$
- Startzustand *(initial state)* $q_0 \in Q$
- Akzeptierender Zustand *(accepting state)* $q_+ \in Q$
- Ablehnender Zustand *(rejecting state)* $q_- \in Q$

$\triangleleft$ = gehe nach links; $\triangleright$ = gehe nach rechts; $\diamond$ = keine Bewegung

Turingmaschine

Damit programmieren?

- Einfaches Modell (vereinfacht Beweise Nichtberechenbarkeit)
- Gleichmächtig wie gebräuchliche Programmiersprachen (C++, Java, Perl, Python, etc.)
- Nicht komfortabel
- Übergangsrelation $\widehat{=}$ Programm
- Arbeitsband $\widehat{=}$ Speicher (kein Direktzugriff)

Turingmaschine

Notation: $(q,\gamma) \to (q',\gamma',d) \in \Delta$ statt $((q,\gamma),(q',\gamma',d)) \in \Delta$

§2.5 Beispiel (Turingmaschine = TM)

TM $M = (\{q_0, q, q_a, q'_a, q_b, q'_b, f, \perp\}, \{a,b\}, \{a,b,\square\}, \Delta, \square, q_0, f, \perp)$
mit den Übergängen Δ

$$\begin{array}{lll}
(q_0,a) \to (q_a,\square,\triangleright) & (q_0,b) \to (q_b,\square,\triangleright) & (q_0,\square) \to (f,\square,\diamond) \\
(q_a,a) \to (q_a,a,\triangleright) & (q_a,b) \to (q_a,b,\triangleright) & (q_a,\square) \to (q'_a,\square,\triangleleft) \\
(q_b,a) \to (q_b,a,\triangleright) & (q_b,b) \to (q_b,b,\triangleright) & (q_b,\square) \to (q'_b,\square,\triangleleft) \\
(q'_a,a) \to (q,\square,\triangleleft) & (q'_b,b) \to (q,\square,\triangleleft) & \\
(q,a) \to (q,a,\triangleleft) & (q,b) \to (q,b,\triangleleft) & (q,\square) \to (q_0,\square,\triangleright)
\end{array}$$

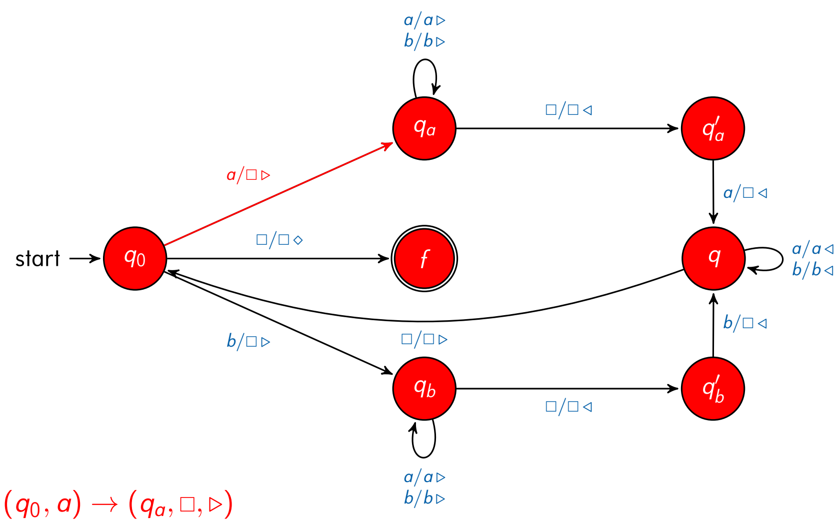
Turingmaschine

Notizen

- Übergang $(q,\gamma) \to (q',\gamma',d)$
  - Vorbedingungen
    1. Aktueller Zustand q
    2. Zeichen γ in Bandzelle, auf der der Kopf steht
  - Konsequenzen
    1. TM wechselt in Zustand q'
    2. γ' überschreibt Inhalt aktueller Bandzelle (ersetzt γ)
    3. Kopf bewegt sich Richtung $d \in \{\triangleleft, \triangleright, \diamond\}$
- Übergänge mit aktuellem Zustand $q \in \{q_+, q_-\}$ verboten
  (Übergänge aus Finalzustand heraus nicht erlaubt)

$\triangleleft$ = gehe nach links; $\triangleright$ = gehe nach rechts; $\diamond$ = keine Bewegung

Turingmaschine



$(q_0,a) \to (q_a,\square,\triangleright)$

Turingmaschine

1. Ausgangssituation
   - Eingabe auf Band (andere Zellen $\square$)
   - TM in Startzustand q_0
   - Kopf auf erstem Symbol der Eingabe (auf $\square$ falls Eingabe leer)
2. Übergänge gemäß Δ
3. Haltebedingung
   - Aktueller Zustand final; akzeptierend q_+ oder ablehnend q_-
   - Kein passender Übergang $\to$ TM hält nicht ordnungsgemäß (Ausnahme)

Akzeptanz Eingabe

Existenz Übergänge von Ausgangssituation in akzeptierenden Zustand

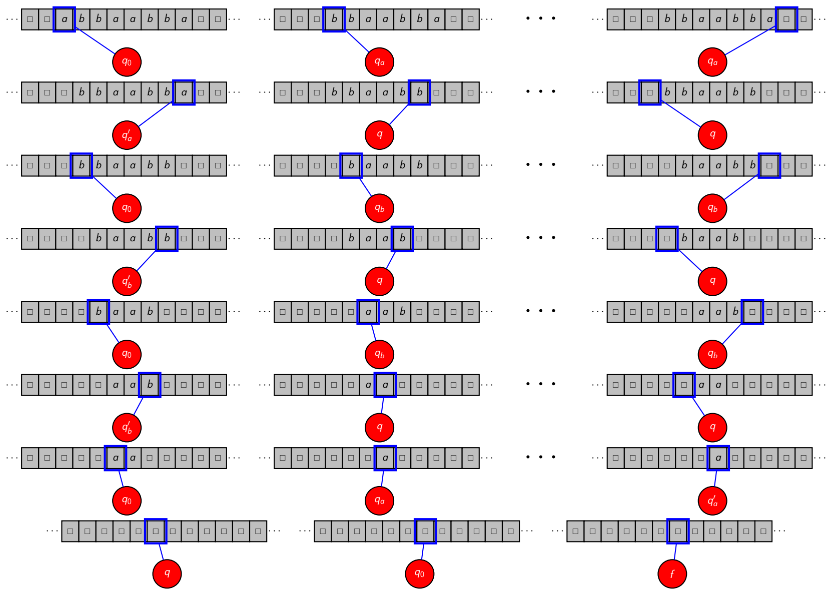
Turingmaschine

Beispiel (§2.5)

TM $M = (\{q_0, q, q_a, q'_a, q_b, q'_b, f, \bot\}, \{a, b\}, \{a, b, \Box\}, \Delta, \Box, q_0, f, \bot)$

$$
\begin{array}{lll}
(q_0, a) \to (q_a, \Box, \triangleright) & (q_0, b) \to (q_b, \Box, \triangleright) & (q_0, \Box) \to (f, \Box, \diamond) \\
(q_a, a) \to (q_a, a, \triangleright) & (q_a, b) \to (q_a, b, \triangleright) & (q_a, \Box) \to (q'_a, \Box, \triangleleft) \\
(q_b, a) \to (q_b, a, \triangleright) & (q_b, b) \to (q_b, b, \triangleright) & (q_b, \Box) \to (q'_b, \Box, \triangleleft) \\
(q'_a, a) \to (q, \Box, \triangleleft) & (q'_b, b) \to (q, \Box, \triangleleft) & \\
(q, a) \to (q, a, \triangleleft) & (q, b) \to (q, b, \triangleleft) & (q, \Box) \to (q_0, \Box, \triangleright)
\end{array}
$$

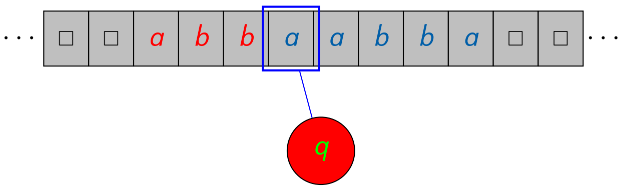
Turingmaschine



Turingmaschine

Satzform

- Globale Systemsituation als Wort
  (Arbeitsband, Position des Kopfes und interner Zustand)
- Kürzen von $\Box$ vom linken und rechten Rand, aber nicht unter Kopf
- Satzform ist $u\, q\, w$
  1. Arbeitsbandbereich $u \in \Gamma^*$ links des Kopfes
  2. Zustand $q \in Q$
  3. Arbeitsbandbereich $w \in \Gamma^+$ unter und rechts des Kopfes
- Situation $abb\ q\ aabba$

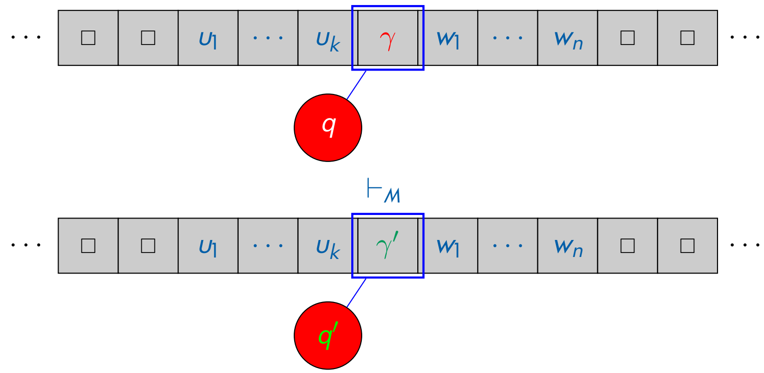


Turingmaschine

§2.6 Definition (Ableitungsrelation — keine Bewegung)

$$u\, q\, \gamma w \vdash_M u\, q'\, \gamma' w$$

falls $(q, \gamma) \to (q', \gamma', \diamond) \in \Delta$

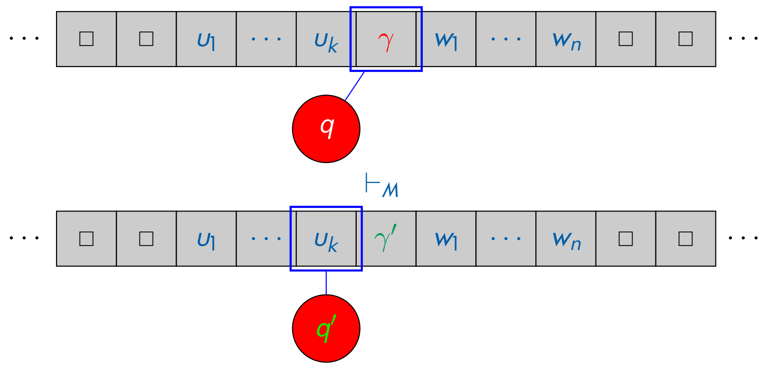


Turingmaschine

§2.6 Definition (Ableitungsrelation — Schritt nach links)

$$u\ q\ \gamma w \vdash_M \begin{cases} \varepsilon\ q'\ \Box\gamma' w & \text{falls } u = \varepsilon \\ u'\ q'\ \gamma''\gamma' w & \text{falls } u = u'\gamma'' \text{ mit } \gamma'' \in \Gamma \end{cases}$$

falls $(q, \gamma) \to (q', \gamma', \triangleleft) \in \Delta$

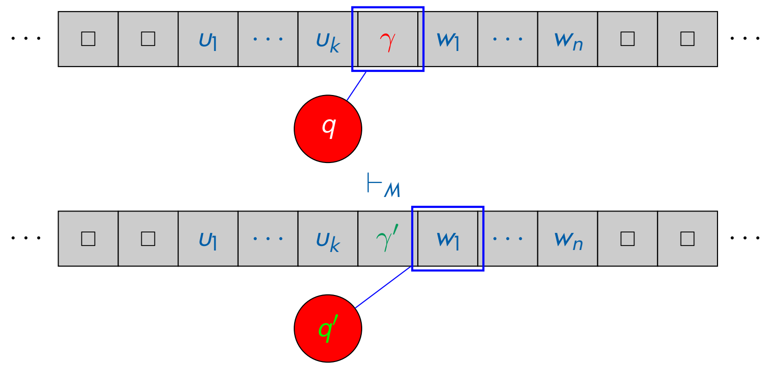


Turingmaschine

§2.6 Definition (Ableitungsrelation — Schritt nach rechts)

$$u\ q\ \gamma w \vdash_M \begin{cases} u\gamma'\ q'\ \Box & \text{falls } w = \varepsilon \\ u\gamma'\ q'\ w & \text{sonst} \end{cases}$$

falls $(q, \gamma) \to (q', \gamma', \triangleright) \in \Delta$



Turingmaschine

§2.7 Definition (akzeptierte Sprache; *accepted language*)

Akzeptierte Sprache von TM $M = (Q, \Sigma, \Gamma, \Delta, \Box, q_0, q_+, q_-)$ ist

$$L(M) = \{w \in \Sigma^* \mid \exists u, v \in \Gamma^* \colon \varepsilon\ q_0\ w\Box \vdash_M^* u\ q_+\ v\}$$

Akzeptanz Eingabe

- Ausgangssituation $\varepsilon\ q_0\ w$ für Eingabe w
- TM **akzeptiert** Eingabe w falls Übergänge von Ausgangssituation $\varepsilon\ q_0\ w$ in akzeptierenden Zustand q_+ existieren

Turingmaschine

Beispiel (§2.5)

TM $M = (\{q_0, q, q_a, q'_a, q_b, q'_b, f, \bot\}, \{a, b\}, \{a, b, \Box\}, \Delta, \Box, q_0, f, \bot)$

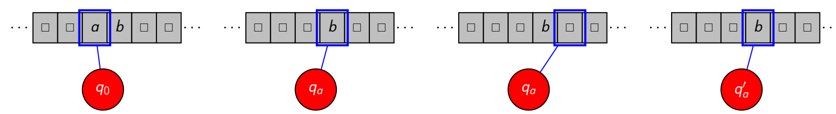
$(q_0, a) \to (q_a, \Box, \triangleright)$ $\quad (q_0, b) \to (q_b, \Box, \triangleright)$ $\quad (q_0, \Box) \to (f, \Box, \diamond)$

$(q_a, a) \to (q_a, a, \triangleright)$ $\quad (q_a, b) \to (q_a, b, \triangleright)$ $\quad (q_a, \Box) \to (q'_a, \Box, \triangleleft)$

$(q_b, a) \to (q_b, a, \triangleright)$ $\quad (q_b, b) \to (q_b, b, \triangleright)$ $\quad (q_b, \Box) \to (q'_b, \Box, \triangleleft)$

$(q'_a, a) \to (q, \Box, \triangleleft)$ $\quad (q'_b, b) \to (q, \Box, \triangleleft)$

$(q, a) \to (q, a, \triangleleft)$ $\quad (q, b) \to (q, b, \triangleleft)$ $\quad (q, \Box) \to (q_0, \Box, \triangleright)$



Turingmaschine

Transformationssemantik

- Für Berechnung Funktionen & Modularität
- Eingabe übersetzt in Bandinhalt bei Akzeptanz
  - Band vor Kopf leer
  - Ausgabe beginnend unter Kopf bis zum ersten $\Box$
  - Band dahinter leer
- Beispiel §2.5 aus VL 2 berechnet

$$\{(ww^R, \varepsilon) \mid w \in \{a, b\}^*\}$$

§3.1 Definition (Transformationssemantik; *input-output relation*)

Sei $M = (Q, \Sigma, \Gamma, \Delta, \Box, q_0, q_+, q_-)$ TM und $\Gamma_M = \Gamma \setminus \{\Box\}$

$$T(M) = \{(w, u) \in \Sigma^* \times \Gamma_M^* \mid \exists x, y \in \{\Box\}^* \colon \varepsilon\ q_0\ w\Box \vdash_M^* x\ q_+\ uy\}$$

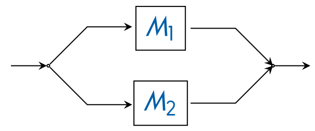
Operationen auf Turingmaschinen

§3.2 Theorem (Vereinigung)

Gegeben Turingmaschinen
$M_1 = (Q, \Sigma, \Gamma, \Delta, \Box, q_0, q_+, q_-)$ und $M_2 = (P, \Sigma, \Gamma, \nabla, \Box, p_0, p_+, p_-)$
existiert TM M mit $L(M) = L(M_1) \cup L(M_2)$ und $T(M) = T(M_1) \cup T(M_2)$

Beweisansatz

1. Nutze neuen Startzustand r_0
2. Neue Übergänge ohne Änderungen zu alten Startzuständen q_0 und p_0
3. M_1 und M_2 laufen normal, wobei alle Übergänge in p_+ oder p_- stattdessen in q_+ bzw. q_- gehen



Operationen auf Turingmaschinen

§3.2 Theorem (Vereinigung)

Gegeben Turingmaschinen
$M_1 = (Q, \Sigma, \Gamma, \Delta, \Box, q_0, q_+, q_-)$ und $M_2 = (P, \Sigma, \Gamma, \nabla, \Box, p_0, p_+, p_-)$
existiert TM M mit $L(M) = L(M_1) \cup L(M_2)$ und $T(M) = T(M_1) \cup T(M_2)$

Beweis

OBdA sei $Q \cap P = \emptyset$ und $r_0 \notin Q \cup P$. Konstruiere TM

$$\begin{aligned} M &= \big(Q \cup P \cup \{r_0\}, \Sigma, \Gamma, \Delta \cup \nabla \cup R, \Box, r_0, q_+, q_-\big) \\ R &= \big\{(r_0, \gamma) \to (q_0, \gamma, \diamond) \mid \gamma \in \Gamma\big\} \cup \\ &\quad \big\{(r_0, \gamma) \to (p_0, \gamma, \diamond) \mid \gamma \in \Gamma\big\} \cup \\ &\quad \big\{(p, \gamma) \to (q_+, \gamma', d) \mid (p, \gamma) \to (p_+, \gamma', d) \in \nabla\big\} \cup \\ &\quad \big\{(p, \gamma) \to (q_-, \gamma', d) \mid (p, \gamma) \to (p_-, \gamma', d) \in \nabla\big\} \end{aligned}$$

Dann $L(M) = L(M_1) \cup L(M_2)$ und $T(M) = T(M_1) \cup T(M_2)$ $\Box$

Operationen auf Turingmaschinen

$\Gamma_M = \Gamma \setminus \{\Box\}$

§3.3 Definition (normierte TM; *standardized TM*)

TM $M = (Q, \Sigma, \Gamma, \Delta, \Box, q_0, q_+, q_-)$ **normiert**, falls $u \in \{\Box\}^*$ und $v \in \Gamma_M^*\{\Box\}^*$ für alle $w \in \Sigma^*$, $u, v \in \Gamma^*$ mit $\varepsilon\ q_0\ w\Box \vdash_M^*\ u\ q_+\ v$

Notizen

- Normierte TM kann nur akzeptieren, falls Band links des Kopfes aus $\{\Box\}^*$ und Band unter und rechts des Kopfes aus $\Gamma_M^*\{\Box\}^*$
- Konstruieren meist normierte TM
- Vereinigung normierter TM gemäß Theorem §3.2 ist normiert

Operationen auf Turingmaschinen

§3.4 Definition (Verkettung; *composition*)

Verkettung $R_1 \,;\, R_2$ von Relationen $R_1 \subseteq A \times B$ und $R_2 \subseteq B \times C$

$$R_1 \,;\, R_2 = \{(a, c) \in A \times C \mid \exists b \in B \colon (a, b) \in R_1,\ (b, c) \in R_2\}$$

Notizen

- Reihenschaltung (Hintereinanderschaltung)
- Erhalten für $\text{verdoppeln} = \{(n, 2n) \mid n \in \mathbb{N}\} \subseteq \mathbb{N} \times \mathbb{N}$

$$\text{verdoppeln} \,;\, \text{verdoppeln} = \{(n, 4n) \mid n \in \mathbb{N}\}$$

Operationen auf Turingmaschinen

$\Gamma_{M_1} = \Gamma \setminus \{\square\}$

§3.5 Theorem (Verkettung)

Gegeben TM
$M_1 = (Q, \Sigma, \Gamma, \Delta, \square, q_0, q_+, q_-)$ und $M_2 = (P, \Gamma_{M_1}, \Psi, \nabla, \square, p_0, p_+, p_-)$
wobei M_1 normiert. Dann existiert TM M mit $T(M) = T(M_1) \,;\, T(M_2)$.
Falls M_2 normiert ist, dann ist M normiert.

Beweisansatz

1. Starte M_1
2. Starte M_2 bei Akzeptanz von M_1
   (Normierung für Ausgangssituation)
3. M_2 läuft normal

$$\longrightarrow \boxed{M_1} \longrightarrow \boxed{M_2} \longrightarrow$$

Operationen auf Turingmaschinen

§3.5 Theorem (Verkettung)

Gegeben TM
$M_1 = (Q, \Sigma, \Gamma, \Delta, \square, q_0, q_+, q_-)$ und $M_2 = (P, \Gamma_{M_1}, \Psi, \nabla, \square, p_0, p_+, p_-)$
wobei M_1 normiert. Dann existiert TM M mit $T(M) = T(M_1) \,;\, T(M_2)$.
Falls M_2 normiert ist, dann ist M normiert.

Beweis

OBdA sei $Q \cap P = \emptyset$. Wir konstruieren TM

$$M = (Q \cup P, \Sigma, \Psi, \Delta \cup \nabla \cup R, \square, q_0, p_+, p_-)$$
$$R = \{(q_+, \gamma) \to (p_0, \gamma, \diamond) \mid \gamma \in \Gamma\}$$

Dann $T(M) = T(M_1) \,;\, T(M_2)$ $\square$

Operationen auf Turingmaschinen

§3.6 Definition (Iteration; *iteration*)

Iteration R^* (reflexive, transitive Hülle) der Relation $R \subseteq A \times A$

$$R^* = \bigcup_{n \in \mathbb{N}} R^n \qquad \text{mit} \qquad R^0 = \mathrm{id}_A \qquad \text{und} \qquad R^{n+1} = R^n \,;\, R$$

Notizen

- Beliebig häufige Wiederholung der Relation
- Erhalten für $\text{verdoppeln} = \{(n, 2n) \mid n \in \mathbb{N}\} \subseteq \mathbb{N} \times \mathbb{N}$

$$\text{verdoppeln}^* = \{(n, 2^m \cdot n) \mid m, n \in \mathbb{N}\}$$

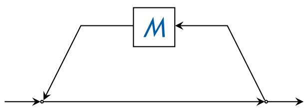
Operationen auf Turingmaschinen

§3.7 Theorem (Iteration)

Sei $M = (Q, \Gamma_M, \Gamma, \Delta, \square, q_0, q_+, q_-)$ normierte TM.
Dann existiert normierte TM N mit $T(N) = T(M)^*$

Beweisansatz

1. Nutze neuen Startzustand p_0 und neuen Akzeptanzzustand p_+
2. Übergang von p_0 zu p_+ (Abbruch)
3. Übergang von p_0 zu q_0 (Iteration)
4. M läuft normal; bei Erreichen von q_+ zurück in Startzustand p_0



Operationen auf Turingmaschinen

§3.7 Theorem (Iteration)

Sei $M = (Q, \Gamma_M, \Gamma, \Delta, \square, q_0, q_+, q_-)$ normierte TM.
Dann existiert normierte TM N mit $T(N) = T(M)^*$

Beweis

Seien $p_0 \notin Q$ und $p_+ \notin Q$ mit $p_0 \neq p_+$. Wir konstruieren TM

$$\begin{aligned} N &= (Q \cup \{p_0, p_+\}, \Gamma_M, \Gamma, \Delta \cup R, \square, p_0, p_+, q_-) \\ R &= \{(p_0, \gamma) \to (p_+, \gamma, \diamond) \mid \gamma \in \Gamma\} \cup \\ &\quad \{(p_0, \gamma) \to (q_0, \gamma, \diamond) \mid \gamma \in \Gamma\} \cup \\ &\quad \{(q_+, \gamma) \to (p_0, \gamma, \diamond) \mid \gamma \in \Gamma\} \end{aligned}$$

Dann $T(N) = T(M)^*$ $\square$

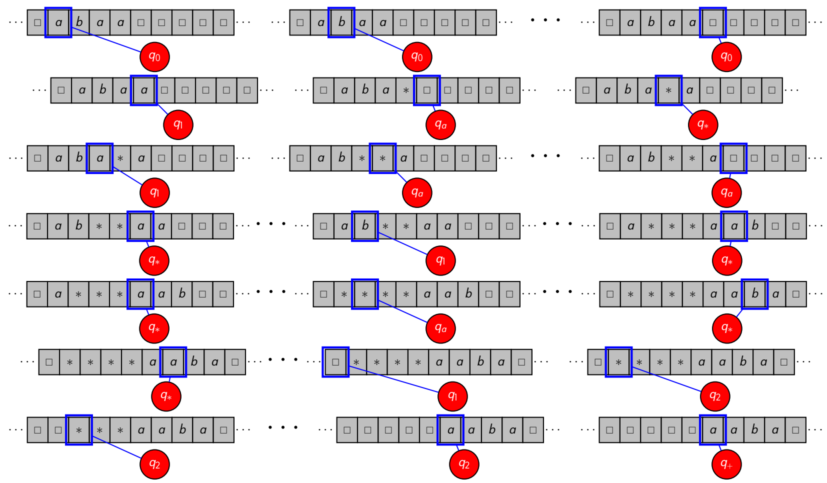
Mehrband-Turingmaschinen

§3.8 Beispiel (Reversal-Turingmaschine)

TM $(\{q_0, q_1, q_a, q_b, q_*, q_2, q_+, q_-\}, \{a, b\}, \{a, b, *, \square\}, \Delta, \square, q_0, q_+, q_-)$

$(q_0, a) \to (q_0, a, \triangleright)$	$(q_0, b) \to (q_0, b, \triangleright)$	$(q_0, \square) \to (q_1, \square, \triangleleft)$
$(q_1, a) \to (q_a, *, \triangleright)$	$(q_1, b) \to (q_b, *, \triangleright)$	$(q_1, *) \to (q_1, *, \triangleleft)$
$(q_1, \square) \to (q_2, \square, \triangleright)$	$(q_a, \square) \to (q_*, a, \triangleleft)$	$(q_b, \square) \to (q_*, b, \triangleleft)$
$(q_a, a) \to (q_a, a, \triangleright)$	$(q_a, b) \to (q_a, b, \triangleright)$	$(q_a, *) \to (q_a, *, \triangleright)$
$(q_b, a) \to (q_b, a, \triangleright)$	$(q_b, b) \to (q_b, b, \triangleright)$	$(q_b, *) \to (q_b, *, \triangleright)$
$(q_*, a) \to (q_*, a, \triangleleft)$	$(q_*, b) \to (q_*, b, \triangleleft)$	$(q_*, *) \to (q_1, *, \triangleleft)$
$(q_2, a) \to (q_+, a, \diamond)$	$(q_2, b) \to (q_+, b, \diamond)$	$(q_2, *) \to (q_2, \square, \triangleright)$

Mehrband-Turingmaschinen



Mehrband-Turingmaschinen

Notizen

- Viele Operationen nötig für Navigation
- Oft viele Läufe zwischen Ein- & Ausgabe nötig
- Erhöhter Komfort durch mehrere Bänder (und intuitiver)

Mehrband-Turingmaschinen

§3.9 Definition (k-Band-Turingmaschine; *k-tape Turing machine*)

k-Band-Turingmaschine ist Tupel $M = (Q, \Sigma, \Gamma, \Delta, \square, q_0, q_+, q_-)$

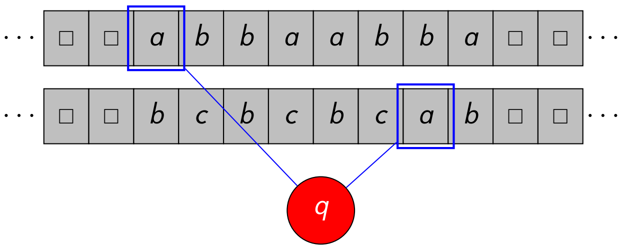
- endl. Menge Q von Zuständen mit $Q \cap \Gamma = \emptyset$
- endl. Menge Σ von Eingabesymbolen
- endl. Menge Γ von Arbeitssymbolen mit $\Sigma \subseteq \Gamma$
- Übergangsrelation $\Delta \subseteq$
  $\Big(\big(Q \setminus \{q_+, q_-\}\big) \times \Gamma^k\Big) \times \Big(Q \times \big(\Gamma \times \{\triangleleft, \triangleright, \diamond\}\big)^k\Big)$
- Leersymbol $\square \in \Gamma \setminus \Sigma$ $\qquad (\Gamma_M = \Gamma \setminus \{\square\})$
- Startzustand $q_0 \in Q$
- Akzeptierender Zustand $q_+ \in Q$
- Ablehnender Zustand $q_- \in Q$

$\triangleleft$ = gehe nach links; $\triangleright$ = gehe nach rechts; $\diamond$ = keine Bewegung

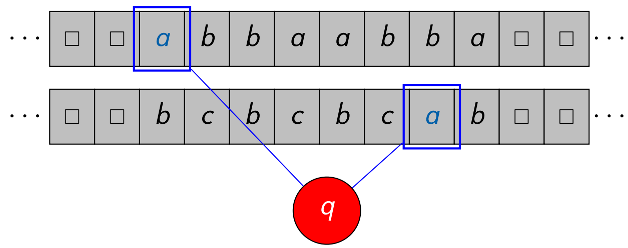
Mehrband-Turingmaschinen

Notizen

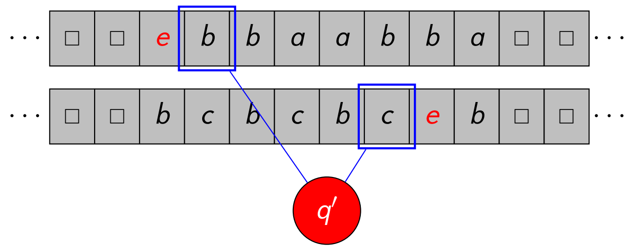
- k Arbeitsbänder (gleiches Arbeitsalphabet)
- k unabhängige Lese- & Schreibköpfe (unabhängig beweglich)
- Übergänge $\tau \in \Big(\big(Q \setminus \{q_+, q_-\}\big) \times \Gamma^k\Big) \times \Big(Q \times \big(\Gamma \times \{\triangleleft, \triangleright, \diamond\}\big)^k\Big)$
  - Aktueller globaler Zustand
  - Inhalt aktuellen Zellen auf allen k Bändern
  - Globaler Zielzustand
  - Neuer Inhalt aller k Zellen
  - k Bewegungsrichtungen für k Köpfe



Mehrband-Turingmaschinen



vom Übergang $(q, \langle a, a\rangle) \to (q', \langle (e, \triangleright), (e, \triangleleft)\rangle)$ überführt in



Mehrband-Turingmaschinen

- Ausgangssituation
  - Eingabe auf <u>erstem</u> Band; andere Zellen & Bänder enthalten □
  - TM in Startzustand q_0
  - Kopf <u>erstes Band</u> auf erstem Symbol der Eingabe
- Übergänge gemäß Δ
- Haltebedingung
  - Aktueller Zustand final; akzeptierend q_+ oder ablehnend q_-
  - Kein passender Übergang $\to$ TM hält <u>nicht</u> ordnungsgemäß

Akzeptanz Eingabe
Existenz Übergänge von Ausgangssituation in akzeptierenden Zustand

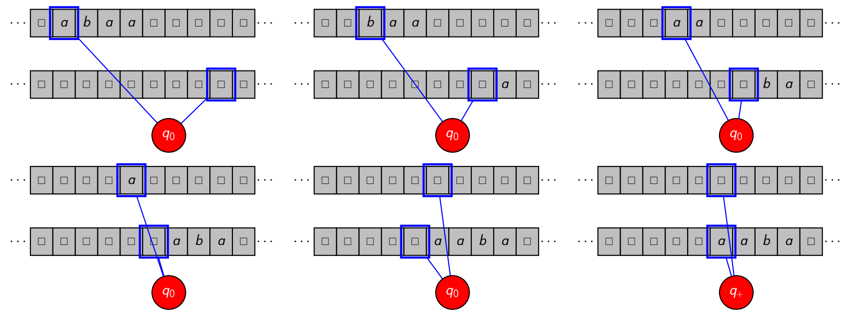
Ausgabe auf <u>letztem</u> Band (Band k) (normiert mind. auf letztem Band)

Mehrband-Turingmaschinen

§3.10 Beispiel (2-Band-Turingmaschine)

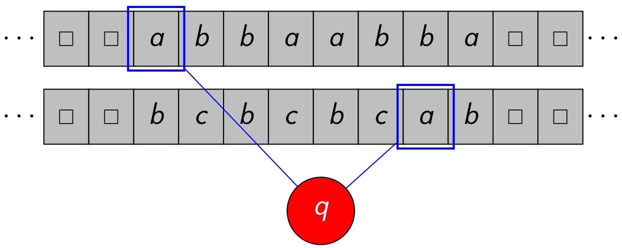
2-Band-TM $M = (\{q_0, q_+, q_-\}, \{a, b\}, \{a, b, \Box\}, \Delta, \Box, q_0, q_+, q_-)$

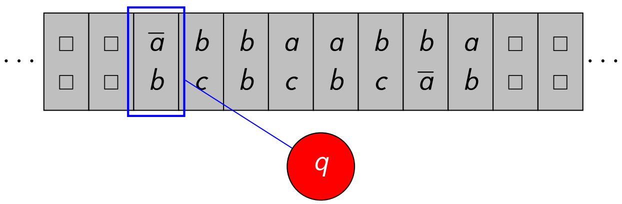
$(q_0, \langle a, \Box\rangle) \to (q_0, \langle(\Box, \triangleright), (a, \triangleleft)\rangle)$ $(q_0, \langle b, \Box\rangle) \to (q_0, \langle(\Box, \triangleright), (b, \triangleleft)\rangle)$
$(q_0, \langle \Box, \Box\rangle) \to (q_+, \langle(\Box, \diamond), (\Box, \triangleright)\rangle)$



Mehrband-Turingmaschinen

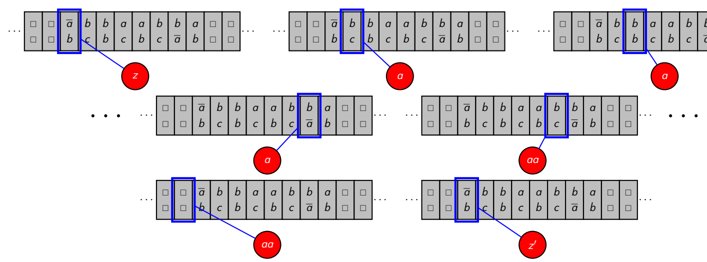
Simulation der k-Band-TM $(Q, \Sigma, \Gamma, \Delta, \Box, q_0, q_+, q_-)$ durch TM
- Kodiere k Bänder durch 1 Band $\Gamma' = \Gamma \cup (\Gamma \cup \overline{\Gamma})^k$ (Tupelsymbole)
- Kodierung Position k Köpfe (Überstrich)





Mehrband-Turingmaschinen

Simulation Ableitungsschritt k-Band-TM durch TM
1. Merken aktueller Zustand in Zuständen $(q, p, \dots)$
   1.1 Zustand q k-Band-TM
   1.2 Phase p in Bearbeitung mit weiteren Informationen
2. Aufsammeln Symbole unter Köpfen durch Ablaufen Band



$z = (q, \text{lese}, \langle\star, \star\rangle)$ $a = (q, \text{lese}, \langle a, \star\rangle)$
$aa = (q, \text{zurück}, \langle a, a\rangle)$ $z' = (q, \text{wähle}, \langle a, a\rangle)$

Mehrband-Turingmaschinen

Simulation Ableitungsschritt k-Band-TM durch TM

1. ...
2. ...
3. Nichtdeterministische Auswahl passender Übergang

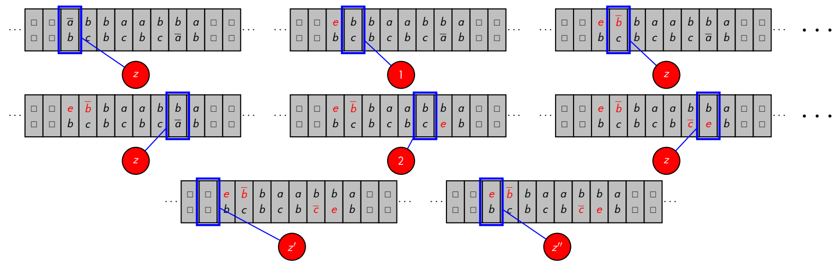
$$\big((q, \text{wähle}, \langle s_1, \ldots, s_k\rangle), \vec{a}\big) \to \big((q', \text{schreibe}, \vec{r}), \vec{a}, \diamond\big) \in \Delta$$

für alle Übergänge $(q, \langle s_1, \ldots, s_k\rangle) \to (q', \vec{r})$ der k-Band-TM

Mehrband-Turingmaschinen

Simulation Ableitungsschritt k-Band-TM durch TM

1. ...
2. ...
3. ...
4. Anpassen Arbeitsband (Schreibvorgänge & Bewegungen)



$z = (q', \text{schreibe}, \langle (e, \triangleright), (e, \triangleleft)\rangle)$ $\qquad$ $z'' = (q', \text{lese}, \langle \star, \star\rangle)$

Mehrband-Turingmaschinen

§3.11 Theorem

Für (normierte) k-Band-TM M existiert (norm.) TM N mit $T(N) = T(M)$

$$\longrightarrow \boxed{M_{\text{start}}} \longrightarrow \boxed{M_{\text{simul}}} \longrightarrow \boxed{M_{\text{ausgabe}}} \longrightarrow$$

Beweisskizze

1. M_{start}: Einrichten Ausgangssituation (Erweitern Eingabe auf Tupel)
2. M_{simul}: Simulation Ableitungsschritte (wie gerade illustriert)
3. M_{ausgabe}: Ausgabe letztes Band (Reduktion Tupel, Löschen) □

Mehrband-Turingmaschinen

Standard-Operationen

- Band auf anderes Band kopieren
- TM M auf Band i laufen lassen ($M(i)$ ist diese k-Band-TM)

Konsequenzen

- Verwende Bänder wie Variablen
- <u>Verwende k-Band-TM statt TM</u> (äquivalente TM existiert)

Mächtigkeit Turingmaschine

§4.1 Theorem

Für jede Grammatik G existiert normierte TM M mit $L(M) = L(G)$

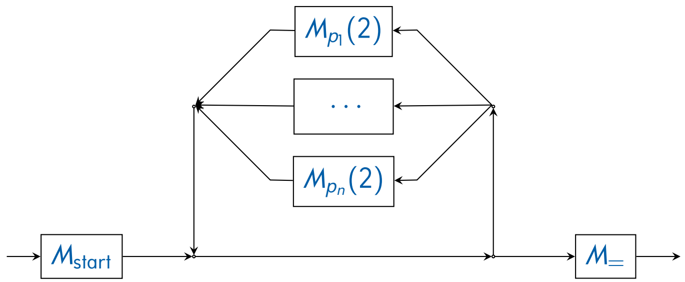
Beweisansatz mit 2-Band-TM

Sei $G = (N, \Sigma, S, P)$

1. Falls $S \to \varepsilon \in P$ und Eingabe ε, dann akzeptiere (d.h. Kopf steht auf $\Box$)
2. Sonst schreibe Startnichtterminal S auf Band 2
3. Wende Produktionen P auf Band 2 an
4. Vergleiche Bänder und akzeptiere bei Gleichheit $\Box$

Mächtigkeit Turingmaschine

$P = \{p_1, \dots, p_n\}$

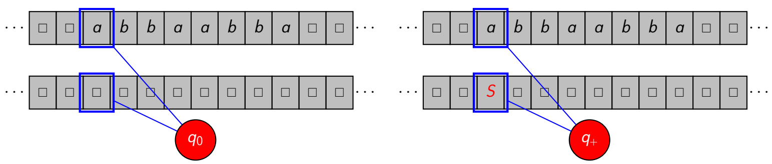


Mächtigkeit Turingmaschine

2-Band-TM $M_{\text{start}} = (\{q_0, q_+, q_-\}, \Sigma, \Gamma, \Delta, \Box, q_0, q_+, q_-)$

- $\Gamma = \{\Box\} \cup \Sigma \cup N$
- Übergänge

$$\Delta = \{(q_0, \langle \Box, \Box \rangle) \to (q_+, \langle (\Box, \diamond), (\Box, \diamond) \rangle) \mid S \to \varepsilon \in P\} \cup$$
$$\{(q_0, \langle \sigma, \Box \rangle) \to (q_+, \langle (\sigma, \diamond), (S, \diamond) \rangle) \mid \sigma \in \Sigma\}$$

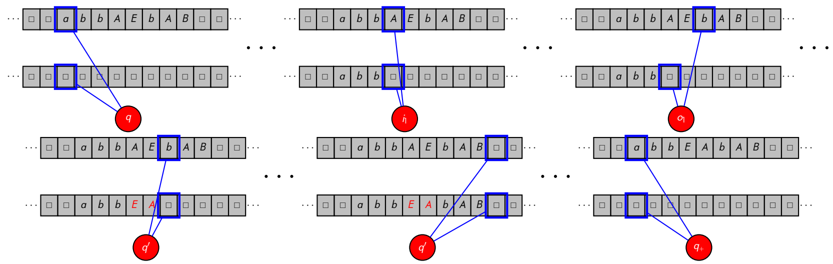


Mächtigkeit Turingmaschine

2-Band-TM M'_p für Übergang $p = \ell \to r \in P$

- Kopiere Symbole Band 1 $\to$ 2 mit Halt auf bel. Symbol (außer $\Box$)
- Lese ℓ auf Band 1 (ohne Aktionen auf Band 2)
- Bei Erfolg schreibe r auf Band 2 (ohne Aktionen auf Band 1)
- Kopiere verbleibende Symbole Band 1 $\to$ 2

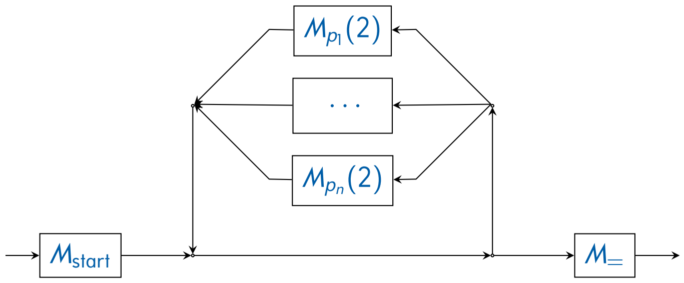
Illustration für Produktion $AE \to EA$



Mächtigkeit Turingmaschine

Ableitungsschritt-TM M_p

- Umwandlung 2-Band-TM M'_p in TM M_p
- Realisiert Anwendung Übergang p auf Arbeitsband
- Angewandt auf Band 2 der Gesamt-TM

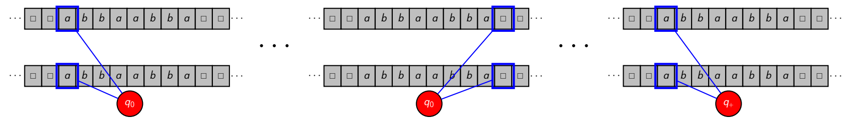


Mächtigkeit Turingmaschine

2-Band-TM $M_= = (\{q_0, q, q_+, q_-\}, \Gamma \setminus \{\square\}, \Gamma, \Delta, \square, q_0, q_+, q_-)$

- $\Gamma = \Sigma \cup N \cup \{\square\}$
- Übergänge

$$\begin{aligned}\Delta = {} & \{(q_0, \langle\sigma, \sigma\rangle) \to (q_0, \langle(\sigma, \triangleright), (\sigma, \triangleright)\rangle) \mid \sigma \in \Sigma\} \cup \\ & \{(q_0, \langle\square, \square\rangle) \to (q, \langle(\square, \triangleleft), (\square, \triangleleft)\rangle)\} \cup \\ & \{(q, \langle\sigma, \sigma\rangle) \to (q, \langle(\sigma, \triangleleft), (\sigma, \triangleleft)\rangle) \mid \sigma \in \Sigma\} \cup \\ & \{(q, \langle\square, \square\rangle) \to (q_+, \langle(\square, \triangleright), (\square, \triangleright)\rangle)\}\end{aligned}$$



Mächtigkeit Turingmaschine

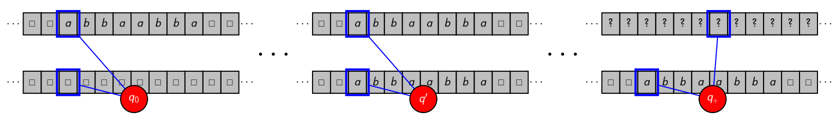
§4.2 Lemma

Sei M TM. Dann existiert TM M' mit $T(M') = \mathrm{id}_{L(M)}$

Beweisansatz

Nutze 2-Band-TM

- Kopiere Eingabe auf Band 2 (und Rücklauf auf 1. Zeichen)
- Lasse M auf Band 1 laufen □



Mächtigkeit Turingmaschine

§4.3 Theorem

Für jede TM M existiert Grammatik G mit $L(G) = L(M)$

Beweisansatz

Es existiert TM M' mit $T(M') = \{(w, w) \mid w \in L(M)\}$ via Lemma §4.2

1. Erzeuge Ausgangssituation mit markierten Rändern (linker Rand überstrichen; rechter Rand unterstrichen)
2. Simuliere Schritte der TM M'
3. Lösche überzählige $\square$

Notizen

- Grammatik-Satzform entspricht TM-Satzform (Systemsituation)
- Symbol unter Lesekopf und TM-Zustand in Nichtterminal kodiert

Mächtigkeit Turingmaschine

Beweisskizze (1/3)

1. Erzeuge Ausgangssituation mit markierten Rändern
   - Eingabealphabet Σ und Arbeitsbandalphabet Γ
   - Nichtterminale $\Gamma' \cup (Q \times (\Gamma' \cup \Sigma))$ mit $\Gamma' = (\Gamma \setminus \Sigma) \cup \overline{\Gamma} \cup \underline{\Gamma} \cup \underline{\overline{\Gamma}}$
   - Produktionen

$$P_1 = \{S \to S'\underline{\Box},\ S \to (q_0, \underline{\overline{\Box}})\} \cup$$
$$\{S' \to S'a \mid a \in \Sigma\} \cup \{S' \to (q_0, \overline{a}) \mid a \in \Sigma\}$$

   - Ableitungen der Form: $S \Rightarrow_G^* (q_0, \overline{a})w\underline{\Box}$ (Ausgangssituation TM M')

Notizen

- Erzeugt geratene Eingabe aw mit markierten Rändern
- Beispielableitung (Startzustand q_0 und Eingabe $abaa$)

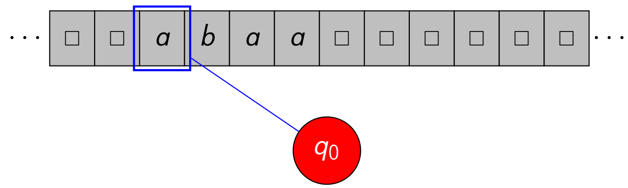
$$S \Rightarrow_G S'\underline{\Box} \Rightarrow_G S'a\underline{\Box} \Rightarrow_G S'aa\underline{\Box} \Rightarrow_G S'baa\underline{\Box} \Rightarrow_G (q_0, \overline{a})baa\underline{\Box}$$

Mächtigkeit Turingmaschine

Grammatiksatzform

$$(q_0, \overline{a})baa\underline{\Box}$$

TM-Systemsituation



Mächtigkeit Turingmaschine

Beweisskizze (2/3)

2. Simuliere Schritte TM M'
   - Produktionen

$$P_2 = \{a(q,b) \to (q',a)b' \mid (q,b) \to (q',b',\triangleleft) \in \Delta,\ a \in \Gamma\} \cup$$
$$\{(q,b) \to (q',b') \mid (q,b) \to (q',b',\diamond) \in \Delta\} \cup$$
$$\{(q,b)c \to b'(q',c) \mid (q,b) \to (q',b',\triangleright) \in \Delta,\ c \in \Gamma\} \cup$$
$$\{(q,\overline{b}) \to (q',\overline{\Box})b' \mid (q,b) \to (q',b',\triangleleft) \in \Delta\} \cup$$
$$\{(q,\overline{b}) \to (q',\overline{b'}) \mid (q,b) \to (q',b',\diamond) \in \Delta\} \cup$$
$$\{(q,\overline{b})c \to \overline{b'}(q',c) \mid (q,b) \to (q',b',\triangleright) \in \Delta,\ c \in \Gamma\} \cup$$
$$\cdots \quad \text{(viele weitere Varianten)}$$

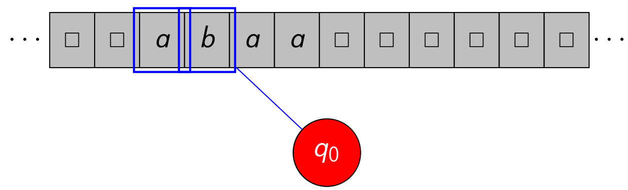
   - Beispielableitung
     $(q_0, \overline{a})bbaabba\underline{\Box} \Rightarrow_G \overline{\Box}(q_a, b)baabba\underline{\Box} \Rightarrow_G \overline{\Box}b(q_a, b)aabba\underline{\Box}$ □

Mächtigkeit Turingmaschine

Notizen

- Produktionen P_2 bilden Semantik Übergänge ab
- Varianten durch verschiedene Randsituationen
- $(q_0, a) \to (q_0, a, \triangleright)$ wird am linken Rand zu $(q_0, \overline{a})b \to \overline{a}(q_0, b)$

$$(q_0, \overline{a})baa\underline{\Box} \Rightarrow_G \overline{a}(q_0, b)aa\underline{\Box}$$



Mächtigkeit Turingmaschine

Beweisskizze (3/3)

3. Lösche überzählige $\Box$
   - Produktionen

$$
\begin{aligned}
P_3 = {} & \{\Box(q_+, b) \to (q_+, b) \mid b \in \Gamma \cup \underline{\Gamma}\} \cup \\
& \{\overline{\Box}(q_+, b) \to (\bot, b) \mid b \in \Gamma \cup \underline{\Gamma}\} \cup \\
& \{(q_+, \overline{b}) \to (\bot, b) \mid b \in \Gamma \cup \underline{\Gamma}\} \cup \\
& \{(\bot, \underline{b}) \to b \mid b \in \Gamma\} \cup \\
& \{(\bot, b)c \to b(\bot, c) \mid b \in \Gamma,\ c \in \Gamma \cup \underline{\Gamma}\} \cup \\
& \{(\bot, \Box)c \to (\top, c) \mid c \in \{\Box, \underline{\Box}\}\} \cup \\
& \{(\top, \Box)c \to (\top, c) \mid c \in \{\Box, \underline{\Box}\}\} \cup \{(\top, \underline{\Box}) \to \varepsilon\}
\end{aligned}
$$

   - Beispielableitung
     $\overline{\Box}\Box(q_+, a)bbaab\Box\underline{\Box} \Rightarrow_G^2 (\bot, a)bbaab\Box\underline{\Box} \Rightarrow_G^* abbaab(\bot, \Box)\underline{\Box}$
     $\Rightarrow_G abbaab(\top, \underline{\Box}) \Rightarrow_G abbaab$ $\Box$

Deterministische Turingmaschinen

§4.5 Definition (deterministische TM; *deterministic TM*)

TM $(Q, \Sigma, \Gamma, \Delta, \Box, q_0, q_+, q_-)$ **deterministisch** (*deterministic*)
falls für alle $(q, \gamma) \in (Q \setminus \{q_+, q_-\}) \times \Gamma$ genau ein (q', γ', d) existiert
mit $(q, \gamma) \to (q', \gamma', d) \in \Delta$
d.h. $\Delta \colon \big((Q \setminus \{q_+, q_-\}) \times \Gamma\big) \to \big(Q \times \Gamma \times \{\triangleleft, \triangleright, \diamond\}\big)$

Notizen

- Jede Eingabe erzeugt 1 Lauf deterministischer TM
- Det. TM kann nur in q_+ und q_- halten (akzeptiert bzw. lehnt ab)
- Endlosschleifen weiterhin möglich
- Simulator `https://turingmachinesimulator.com/`

Deterministische Turingmaschinen

§4.6 Theorem

TM und deterministische TM gleichmächtig (für Sprachen)

Beweisskizze

1. Schreibe Initialzustand vor Eingabe w $q_0 w \Box$
2. Erzeuge nächste Berechnung
3. Prüfe Gültigkeit Berechnung
4. Akzeptiere Eingabe bei Gültigkeit
5. Zurück zu 2. $\Box$

Deterministische Turingmaschinen

Geg. TM $M = (Q, \Sigma, \Gamma, \Delta, \Box, q_0, q_+, q_-)$ und Eingabe $w \in \Sigma^*$

Berechnung für w ist Zeichenkette

$$q_0 w \Box \# \xi_1 \# \xi_2 \# \cdots \# \xi_n$$

mit $\xi_1, \dots, \xi_n \in \Gamma^* Q \Gamma^*$ $\# \notin \Gamma \cup Q$

Notizen

- Zeichenketten deterministisch erzeugbar
  z.B. in längenlexikographischer Ordnung
  - ε, Worte der Länge 1, Worte der Länge 2, etc.
  - Worte der Länge k lexikographisch aufgelistet (wie im Duden)

Deterministische Turingmaschinen

Geg. TM $M = (Q, \Sigma, \Gamma, \Delta, \Box, q_0, q_+, q_-)$ und Eingabe $w \in \Sigma^*$

Gültige Berechnung $q_0 w\Box\#\xi_1\# \cdots \#\xi_n$ für w falls

- $\xi_1, \dots, \xi_n \in \Gamma^* Q \Gamma^*$
- $q_0 w\Box \vdash_M \xi_1 \vdash_M \cdots \vdash_M \xi_n$
- $\xi_n \in \Gamma^* \{q_+\} \Gamma^*$

Überprüfung Gültigkeit Berechnung mit det. TM möglich

Turing-Berechenbarkeit

§4.7 Beobachtung

Für jede deterministische TM M ist $T(M)$ partielle Funktion

§4.8 Definition (Turing-berechenbar; *Turing-computable*)

Partielle Funktion $f \colon \Sigma^* \dashrightarrow \Gamma^*$ **Turing-berechenbar**
falls deterministische TM M mit $f = T(M)$ existiert

Notiz

- Turing-berechenbare Funktionen $f \colon \mathbb{N}^k \to \mathbb{N}$ per Kodierung

Loop-Programme

Konventionen

- Alle Variablen $x_1, x_2, \dots$ vom Typ $\mathbb{N}$ (beliebige Größe)
- Addition auf $\mathbb{N}$ begrenzt

$$n \oplus z = \max(0, n + z) \qquad n \in \mathbb{N}, z \in \mathbb{Z}$$

- Wir schreiben einfach $+$ statt $\oplus$

§4.9 Definition (Zuweisung; *assignment*)

Zuweisung ist Anweisung der Form $x_i = x_\ell + z$ mit $i, \ell \geq 1$ und $z \in \mathbb{Z}$

Loop-Programme

§4.10 Definition (Loop-Programm; *Loop program*)

Loop-Programm P entweder

- **Zuweisung** $P = x_i = x_\ell + z$ für $i, \ell \geq 1$ und $z \in \mathbb{Z}$
- **Sequenz** $P = P_1 \,;\, P_2$ für Loop-Programme P_1 und P_2
- **Iteration** $P = \mathrm{LOOP}(x_i)\,\{P'\}$ für Loop-Programm P' und $i \in \mathbb{N}$

Beispiele

- $x_2 = x_1 + 2 \,;\, \mathrm{LOOP}(x_2)\,\{x_3 = x_3 + 1\} \,;\, x_1 = x_3 + 0$
- gleiches Programm, leichter lesbar

```
x2 = x1 + 2
LOOP(x2) {
  x3 = x3 + 1
}
x1 = x3 + 0
```

Loop-Programme

(Verzicht auf vollständige Quantifikation; $i, \ell \geq 1$, $z \in \mathbb{N}$, etc.)

§4.11 Definition (Variablen und maximaler Variablenindex)

Für Loop-Programm P seien $\mathrm{var}(P) \subseteq \mathbb{N}$ und $\max \mathrm{var}(P) \in \mathbb{N}$ verwendeten Variablenindices und größter verwendeter Variablenindex

$$\mathrm{var}(x_i = x_\ell + z) = \{i, \ell\}$$
$$\mathrm{var}(P_1 \,;\, P_2) = \mathrm{var}(P_1) \cup \mathrm{var}(P_2)$$
$$\mathrm{var}(\mathrm{LOOP}(x_i)\,\{P'\}) = \{i\} \cup \mathrm{var}(P')$$

$\mathrm{var}(P) = \{1, 2, 3\}$ und $\max \mathrm{var}(P) = 3$ für folgendes Programm P

$x_2 = x_1 + 2$
$\mathrm{LOOP}(x_2)\,\{ \quad x_3 = x_3 + 1 \quad \}$
$x_1 = x_3 + 0$

Loop-Programme

Überblick

- k Eingaben in Variablen $x_1, \ldots, x_k$
- Erwartete Semantik für Zuweisung
- $P_1 \,;\, P_2$ führt P_1 und danach P_2 aus
- $\mathrm{LOOP}(x_i)\,\{P'\}$ führt Programm P' so oft aus, wie Wert von x_i vor Beginn Schleife anzeigt
  (Änderungen an x_i in Schleife ändern Anzahl Durchläufe nicht)
- Funktionswert ist Wert von x_1 nach Ablauf Programm

Loop-Programme

§4.12 Definition (Programmsemantik; *program semantics*)

Für Loop-Programm P mit $\max \mathrm{var}(P) \leq n$ ist **Semantik** von P partielle Funktion $\|P\|_n \colon \mathbb{N}^n \dashrightarrow \mathbb{N}^n$

- $\|x_i = x_\ell + z\|_n(a_1, \ldots, a_n) = (a_1, \ldots, a_{i-1}, a_\ell + z, a_{i+1}, \ldots, a_n)$
- $\|P_1 \,;\, P_2\|_n(a_1, \ldots, a_n) = \|P_2\|_n(\|P_1\|_n(a_1, \ldots, a_n))$
- $\|\mathrm{LOOP}(x_i)\,\{P'\}\|_n(a_1, \ldots, a_n) = \|P'\|_n^{a_i}(a_1, \ldots, a_n)$

für alle $a_1, \ldots, a_n \in \mathbb{N}$

Notizen

- $\|x_2 = x_1 + 2\|_2(5, 2) = (5, 7)$
- $\|x_2 = x_1 + 2 \,;\, x_1 = x_1 - 5\|_2(5, 2) = \|x_1 = x_1 - 5\|_2(5, 7) = (0, 7)$
- $\|\mathrm{LOOP}(x_1)\,\{x_1 = x_1 + 1\}\|_2(5, 2) = (10, 2)$

Loop-Programme

§4.13 Definition (Projektion; *projection*)

Für $n \in \mathbb{N}$ und $1 \leq i \leq n$ ist $\pi_i^{(n)} \colon \mathbb{N}^n \to \mathbb{N}$ **n-stellige Projektion auf i-te Stelle**

$$\pi_i^{(n)}(a_1, \ldots, a_n) = a_i \qquad a_1, \ldots, a_n \in \mathbb{N}$$

Notizen

- $\pi_1^{(2)}(10, 2) = 10$
- $\pi_2^{(2)}(10, 2) = 2$

Loop-Programme

§4.14 Definition (berechnete Funktion; *computed function*)

Loop-Programm P mit $\max \mathrm{var}(P) = n$ **berechnet** k-stellige partielle Funktion $|P|_k \colon \mathbb{N}^k \dashrightarrow \mathbb{N}$ mit $k \leq n$ gegeben für alle $a_1, \ldots, a_k \in \mathbb{N}$

$$|P|_k(a_1, \ldots, a_k) = \pi_1^{(n)}\big(\|P\|_n(a_1, \ldots, a_k, \underbrace{0, \ldots, 0}_{(n-k)\text{ mal}})\big)$$

Notizen

- Eingaben $a_1, \ldots, a_k$ in ersten k Variablen $x_1, \ldots, x_k$
- Weitere Variablen $x_{k+1}, \ldots, x_n$ initial 0
- Auswertung Programm mit dieser initialen Variablenbelegung
- Ergebnis ist Inhalt erster Variable x_1 nach Ablauf

Loop-Berechenbarkeit

§4.15 Definition (Loop-Berechenbarkeit; *Loop-computable*)

Partielle Funktion $f \colon \mathbb{N}^k \dashrightarrow \mathbb{N}$ **Loop-berechenbar** falls Loop-Programm P mit $f = |P|_k$ existiert

Loop-Berechenbarkeit

Nullsetzen x_i

$\mathrm{LOOP}(x_i)\,\{x_i = x_i - 1\}$ Schreibweise: $x_i = 0$

Belegung x_i mit Konstante $n \in \mathbb{N}$

$x_i = 0\,;\, x_i = x_i + n$ Schreibweise: $x_i = n$

Kopieren x_ℓ nach x_i

$x_i = x_\ell + 0$ Schreibweise: $x_i = x_\ell$

Loop-Berechenbarkeit

Addition von x_k und x_ℓ in x_i $(i \neq \ell)$

$x_i = x_k\,;\,\mathrm{LOOP}(x_\ell)\,\{x_i = x_i + 1\}$ Schreibweise: $x_i = x_k + x_\ell$

Multiplikation von x_k und x_ℓ in x_i $(k \neq i \neq \ell)$

$x_i = 0\,;\,\mathrm{LOOP}(x_k)\,\{x_i = x_i + x_\ell\}$ Schreibweise: $x_i = x_k \cdot x_\ell$

Potenzieren von x_ℓ mit x_k in x_i $(k \neq i \neq \ell)$

$x_i = 1\,;\,\mathrm{LOOP}(x_k)\,\{x_i = x_i \cdot x_\ell\}$ Schreibweise: $x_i = x_\ell^{x_k}$

Loop-Berechenbarkeit

Multiplikation strenge Syntax

Zeile	Anweisung	Kommentar
1	$x_3 = x_1 + 0$	$x_3 = x_1$
2	$\mathrm{LOOP}(x_1)$	$x_1 = 0$
3	$\{x_1 = x_1 - 1\}$	
4	$\mathrm{LOOP}(x_2)\,\{$	(x_2 mal)
5	$\mathrm{LOOP}(x_3)$	$x_1 = x_1 + x_3$
6	$\{x_1 = x_1 + 1\}\,\}$	

Berechnung Semantik (Zeilennummern über Pfeil)

$$(2,3,0) \xrightarrow{1} (2,3,2) \underbrace{\xrightarrow{3} (1,3,2) \xrightarrow{3} (0,3,2)}_{\text{Schleife in 2}} \underbrace{\xrightarrow{6} (1,3,2) \xrightarrow{6} (2,3,2)}_{\text{Schleife in 5}}$$

$$\underbrace{\xrightarrow{6} (3,3,2) \xrightarrow{6} (4,3,2)}_{\text{Schleife in 5}} \underbrace{\xrightarrow{6} (5,3,2) \xrightarrow{6} (6,3,2)}_{\text{Schleife in 5}}$$ Ergebnis 6

Loop-Berechenbarkeit

Simulation "If-Then-Else" (x_k, x_ℓ unbenutzt)

$x_k = 1\,;\, x_\ell = 0$
$\mathrm{LOOP}(x_i)\,\{x_k = 0\,;\, x_\ell = 1\}$
$\mathrm{LOOP}(x_k)\,\{P_1\}$
$\mathrm{LOOP}(x_\ell)\,\{P_2\}$

Schreibweise: $\mathrm{IF}(x_i = 0)\,\{P_1\}\ \mathrm{ELSE}\ \{P_2\}$

Notizen

- Falls $x_i > 0$
  - Zeile 2: $x_k = 0$ und $x_\ell = 1$
  - Zeile 3: P_1 nicht ausgeführt; Zeile 4: P_2 einmal ausgeführt
- Falls $x_i = 0$
  - Zeile 2: $x_k = 1$ und $x_\ell = 0$
  - Zeile 3: P_1 einmal ausgeführt; Zeile 4: P_2 nicht ausgeführt

Termination von Loop-Programmen

§4.16 Beobachtung

Jedes Loop-Programm P terminiert nach endlich vielen Schritten
d.h. $|P|_k \colon \mathbb{N}^k \to \mathbb{N}$ (totale) Funktion für jedes $k \in \mathbb{N}$

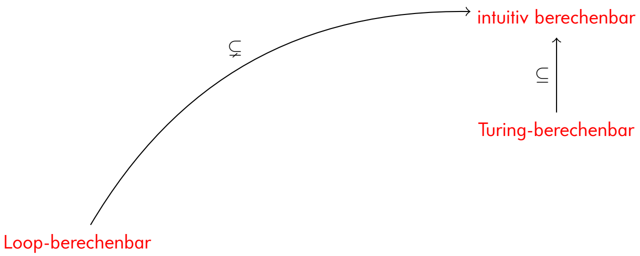
Folgerung

Nicht jede Turing-berechenbare partielle Funktion Loop-berechenbar

Frage

Ist jede intuitiv berechenbare (totale) Funktion Loop-berechenbar?

Wiederholung — Berechenbarkeit



Ackermann-Funktion

§5.1 Definition (Ackermann-Funktion; *Ackermann function*)

Für alle $x, y \in \mathbb{N}$ sei

$$a(x,y) = \begin{cases} y+1 & \text{falls } x = 0 \\ a(x-1,1) & \text{falls } x \neq 0 \text{ und } y = 0 \\ a(x-1, a(x, y-1)) & \text{sonst} \end{cases}$$

Wilhelm Ackermann (* 1896; † 1962)

- Dtsch. Mathematiker
- Student von David Hilbert
- Gymnasiallehrer & Ehrenprofessor Uni Münster

Ackermann-Funktion

§5.2 Theorem

Ackermann-Funktion total; d.h. $a \colon \mathbb{N}^2 \to \mathbb{N}$

Beweis (vollständige Induktion über 1. Argument)

IA: $a(0,y) = y+1$ für alle $y \in \mathbb{N}$ definiert

IS: Sei $a(x,y)$ für alle $y \in \mathbb{N}$ definiert. Dann

$$\begin{aligned} a(x+1,y) &= a(x, a(x+1, y-1)) = a(x, a(x, a(x+1, y-2))) \\ &= \cdots = \underbrace{a(x, a(x, \cdots a}_{(y+1) \text{ mal}}(x+1, 0)\cdots)) \\ &= \underbrace{a(x, a(x, \cdots a}_{(y+1) \text{ mal}}(x, 1)\cdots)) \end{aligned}$$

für alle $y \in \mathbb{N}$ definiert □

Ackermann-Funktion

Problem Ist Ackermann-Funktion Loop-berechenbar?

$x \setminus y$	0	1	2	3	4
0	1	2	3	4	5
1	2	3	4	5	6
2	3	5	7	9	11
3	5	13	29	61	125
4	13	65.533	$\gg 10^{10.000}$	...	...

Ackermann-Funktion

§5.3 Lemma

$a(x, y) > y$ für alle $x, y \in \mathbb{N}$

Beweis (vollständige Induktion über 1. Argument)

IA: $a(0, y) = y + 1 > y$ für alle $y \in \mathbb{N}$

IS: Sei $a(x, y) > y$ für alle $y \in \mathbb{N}$. Vollständige Induktion über y

- IA: $a(x+1, 0) = a(x, 1) \overset{\text{IH}}{>} 1 > 0$ nach IH $a(x, y) > y$
- IS: Sei $a(x+1, y) > y$

$$a(x+1, y+1) = a(x, a(x+1, y)) \overset{\text{IH}}{>} a(x+1, y) \overset{\text{IH}}{\geq} y + 1$$

nach äußerer und danach innerer IH □

Ackermann-Funktion

§5.4 Lemma

$a(x, y+1) > a(x, y)$ für alle $x, y \in \mathbb{N}$

Beweis (mit Hilfe von §5.3)

- Sei $x = 0$

$$a(0, y+1) = y + 2 > y + 1 = a(0, y)$$

- Sei $x > 0$

$$a(x, y+1) = a(x-1, a(x, y)) \overset{\S5.3}{>} a(x, y)$$

□

Ackermann-Funktion

§5.5 Lemma

$a(x, y') > a(x, y)$ für alle $x, y, y' \in \mathbb{N}$ mit $y' > y$

Beweis

Leichte Übung mit Hilfe von §5.4 □

Ackermann-Funktion

§5.6 Lemma

$a(x+1, y) \geq a(x, y+1)$ für alle $x, y \in \mathbb{N}$

Beweis (vollständige Induktion über 2. Argument)

IA: $a(x+1, 0) = a(x, 1)$ für alle $x \in \mathbb{N}$

IS: Sei $a(x+1, y) \geq a(x, y+1)$ für alle $x \in \mathbb{N}$

$$a(x+1, y+1) = a(x, a(x+1, y)) \overset{\S5.5}{\geq} a(x, a(x, y+1)) \overset{\S5.5}{\geq} a(x, y+2)$$

unter Nutzung der IH und §5.4 □

Ackermann-Funktion

§5.7 Lemma

$a(x+1, y) > a(x, y)$ für alle $x, y \in \mathbb{N}$

Beweis

$$a(x+1, y) \overset{\S5.6}{\geq} a(x, y+1) \overset{\S5.4}{>} a(x, y) \qquad \square$$

§5.8 Theorem (Monotonie der Ackermann-Funktion)

$a(x', y') \geq a(x, y)$ für alle $x, x', y, y' \in \mathbb{N}$ mit $x' \geq x$ und $y' \geq y$

Beweis

$$a(x', y') \overset{\S5.7}{\geq} a(x, y') \overset{\S5.5}{\geq} a(x, y) \qquad \square$$

Loop-Berechenbarkeit

§5.9 Definition (norm. Loop-Programm; *unitary Loop program*)

Normiertes Loop-Programm P entweder

- $P = x_i = x_\ell + z$ mit $i, \ell \geq 1$ und $z \in \{-1, 0, 1\}$
- $P = P_1 \,;\, P_2$ für normierte Loop-Programme P_1 und P_2
- $P = \mathbf{LOOP}(x_i)\,\{P'\}$ für normiertes Loop-Programm P', $i \notin \mathrm{var}(P')$

Notizen

- Zuweisungen nur mit Addition von $\{-1, 0, 1\}$
- Schleifenvariable nicht in Schleifenkörper

Loop-Berechenbarkeit

§5.10 Theorem

Jedes Loop-Programm P hat äquiv. normiertes Loop-Programm

Beweisskizze

- Ersetze Zuweisung $x_i = x_\ell + n$ mit $n \in \mathbb{N}$ durch

$$x_i = x_\ell + 1 \,;\, x_i = x_i + 1 \,;\, \ldots \,;\, x_i = x_i + 1$$

- (analog für $n < 0$)
- Ersetze $\mathbf{LOOP}(x_i)\,\{P'\}$ durch

$$x_\ell = x_i \,;\, \mathbf{LOOP}(x_\ell)\,\{P'\}$$

wobei $x_\ell \notin \mathrm{var}(P)$ im Gesamtprogramm P nicht vorkommt $\square$

Loop-Berechenbarkeit

§5.11 Definition (Größenbegrenzung)

Sei P normiertes Loop-Programm mit $\max \mathrm{var}(P) \leq n$.
Definiere $f_P^{(n)} \colon \mathbb{N} \to \mathbb{N}$ für alle $s \in \mathbb{N}$ durch

$$f_P^{(n)}(s) = \max \left\{ \sum_{i=1}^{n} r_i \;\middle|\; s_1, \ldots, s_n \in \mathbb{N},\ \sum_{i=1}^{n} s_i \leq s,\ (r_1, \ldots, r_n) = \|P\|_n(s_1, \ldots, s_n) \right\}$$

Notizen

- $f_P^{(n)}(s)$ maximale Summe Variablenendwerte bei Eingaben $(s_1, \ldots, s_n)$ deren Summe $\sum_{i=1}^{n} s_i$ höchstens s ist
- Kein Variablenendwert oder Funktionsergebnis größer als $f_P(s)$ (bei Eingaben, die sich auf höchstens s summieren)

Loop-Berechenbarkeit

§5.12 Theorem

Für jedes normierte Loop-Programm P mit $\max \mathrm{var}(P) \leq n$ existiert $k \in \mathbb{N}$ mit $f_P^{(n)}(s) < a(k, s)$ für alle $s \in \mathbb{N}$

Beweis (Induktion über Struktur normierter Programme; 1/3)

1. Sei $P = x_i = x_\ell + z$ mit $z \in \{-1, 0, 1\}$. Dann $f_P^{(n)}(s) \leq 2s + 1$.
   Wir wählen $k = 2$

$$f_P^{(n)}(s) \leq 2s + 1 < 2s + 3 = a(2, s)$$

mit $2s + 3 = a(2, s)$ unbewiesen (nette Übung)

Loop-Berechenbarkeit

Beweis (Induktion über Struktur normierter Programme; 2/3)

2. Seien P_1 und P_2 normierte Loop-Programme und $k_1, k_2 \in \mathbb{N}$ mit $f_{P_1}^{(n)}(s) < a(k_1, s)$ und $f_{P_2}^{(n)}(s) < a(k_2, s)$ für alle $s \in \mathbb{N}$.
   Sei $k' = \max(k_1 - 1, k_2)$. Dann für $P = P_1 \,;\, P_2$

$$\begin{aligned} f_P(s) &\leq f_{P_2}^{(n)}(f_{P_1}^{(n)}(s)) \\ &< a(k_2, a(k_1, s)) && \text{(Monotonie)} \\ &\leq a(k', a(k' + 1, s)) && \text{(Monotonie)} \\ &= a(k' + 1, s + 1) \\ &\leq a(k' + 2, s) && \text{(§5.6)} \end{aligned}$$

Aussage gilt für $k = k' + 2$

Loop-Berechenbarkeit

Beweis (Induktion über Struktur normierter Programme; 3/3)

3. Sei P' normiertes Loop-Programm, $k' \in \mathbb{N}$ mit $f_{P'}^{(n)}(s) < a(k', s)$ für alle $s \in \mathbb{N}$. Sei $P = \textbf{LOOP}(x_i)\,\{P'\}$ mit $i \notin \mathrm{var}(P')$ und $s \in \mathbb{N}$.
   Sei m_s Wert von x_i der zum Maximum $f_P^{(n)}(s)$ führt

$$\begin{aligned} f_P^{(n)}(s) &\leq \underbrace{f_{P'}^{(n)}(f_{P'}^{(n)}(\cdots f_{P'}^{(n)}}_{m_s \text{ mal}}(s - m_s)\cdots)) + m_s \\ &\leq \cdots \leq \underbrace{a(k', a(k', \cdots a(k'}_{m_s \text{ mal}}, s - m_s)\cdots)) && \text{(§5.4)} \\ &< \underbrace{a(k', a(k', \cdots a(k'}_{m_s \text{ mal}} + 1, s - m_s)\cdots)) && \text{(§5.7 + §5.4)} \\ &= a(k' + 1, s - 1) < a(k' + 1, s) && \text{(§5.4)} \end{aligned}$$

Aussage gilt für $k = k' + 1$ □

Loop-Berechenbarkeit der Ackermann-Funktion

§5.13 Theorem

Ackermann-Funktion <u>nicht</u> Loop-berechenbar

Beweis

Angenommen $a \colon \mathbb{N}^2 \to \mathbb{N}$ wäre Loop-berechenbar mit Programm P und $n = \max \mathrm{var}(P)$. Dann ist $g \colon \mathbb{N} \to \mathbb{N}$ mit $g(s) = a(s, s)$ für alle $s \in \mathbb{N}$ Loop-berechenbar via $P' = x_2 = x_1 \,;\, P$. Gemäß §5.12 existiert $k \in \mathbb{N}$ mit

$$a(s, s) = g(s) \leq f_{P'}^{(n)}(s) < a(k, s)$$

für alle $s \in \mathbb{N}$. Für $s = k$ entsteht Widerspruch

$$a(k, k) = g(k) \leq f_{P'}^{(n)}(k) < a(k, k) \quad ↯$$

Also Ackermann-Funktion a nicht Loop-berechenbar □

Loop-Berechenbarkeit der Ackermann-Funktion

Konsequenz

Nicht jede intuitiv berechenbare (totale) Funktion Loop-berechenbar

Originaldefinition

$$\begin{aligned}
\varphi(x, y, 0) &= x + y \\
\varphi(x, y, 1) &= x \cdot y \\
\varphi(x, y, 2) &= x^y \\
&\dots \\
\varphi(x, y, z) &= x \uparrow^{z-1} y
\end{aligned}$$

- Iteriert jeweilig vorherige Operation
- Verschachtelungstiefe Schleifen abhängig von Eingabe z
- Nicht Loop-berechenbar

While-Programme

Konventionen

- Alle Variablen $x_1, x_2, \dots$ vom Typ $\mathbb{N}$ (beliebige Größe)
- Addition auf $\mathbb{N}$ begrenzt

$$n \oplus z = \max(0, n + z) \qquad n \in \mathbb{N}, z \in \mathbb{Z}$$

- Wir schreiben einfach $+$ statt $\oplus$

Definition (§4.9 Zuweisung; *assignment*)

Zuweisung ist Anweisung der Form $x_i = x_\ell + z$ mit $i, \ell \geq 1$ und $z \in \mathbb{Z}$

While-Programme

§5.14 Definition (While-Programm; *While program*)

While-Programm P entweder

- **Zuweisung** $P = x_i = x_\ell + z$ für $i, \ell \geq 1$ und $z \in \mathbb{Z}$
- **Sequenz** $P = P_1 \,;\, P_2$ für While-Programme P_1 und P_2
- **While-Schleife** $P = \mathbf{WHILE}(x_i \neq 0)\,\{P'\}$ für While-Programm P', $i \in \mathbb{N}$

Beispiele

- $\mathbf{WHILE}(x_1 \neq 0)\,\{x_2 = x_1 + 5 \,;\, x_1 = x_3 + 1\} \,;\, x_1 = x_3 + 0$
- $\mathbf{WHILE}(x_1 \neq 0)\,\{$ gleiches Programm, leichter lesbar
  $x_2 = x_1 + 5$
  $x_1 = x_3 + 1\}$
  $x_1 = x_3 + 0$

While-Programme

(Verzicht auf vollständige Quantifikation)

§5.15 Definition (Variablen und maximaler Variablenindex)

Für While-Programm P seien $\mathrm{var}(P) \subseteq \mathbb{N}$ und $\max \mathrm{var}(P) \in \mathbb{N}$ verwendeten Variablenindices und größter verwendeter Variablenindex

$$\begin{aligned}
\mathrm{var}(x_i = x_\ell + z) &= \{i, \ell\} \\
\mathrm{var}(P_1 \,;\, P_2) &= \mathrm{var}(P_1) \cup \mathrm{var}(P_2) \\
\mathrm{var}(\mathbf{WHILE}(x_i \neq 0)\,\{P'\}) &= \{i\} \cup \mathrm{var}(P')
\end{aligned}$$

$\mathrm{var}(P) = \{1, 2, 3\}$ und $\max \mathrm{var}(P) = 3$ für folgendes Programm P

$\mathbf{WHILE}(x_1 \neq 0)\,\{x_2 = x_1 + 5 \,;\, x_1 = x_3 + 1\} \,;\, x_1 = x_3 + 0$

While-Programme

Überblick

- k Eingaben in Variablen $x_1, \dots, x_k$ hinterlegt
- Erwartete Semantik für Zuweisung (wie bisher)
- $P_1\,;\,P_2$ führt P_1 und danach P_2 aus (wie bisher)
- **WHILE**$(x_i \neq 0)\,\{P'\}$ wiederholt P' bis $0 =$ aktueller Wert von x_i
  (Änderungen an x_i ändern Anzahl Schleifendurchläufe)
- Funktionswert ist Wert von x_1 nach Ablauf Programms

While-Programme

§5.16 Definition (Programmsemantik; *program semantics*)

Für While-Programm P und $\max \operatorname{var}(P) \leq n$ ist **Semantik** von P partielle Funktion $\|P\|_n \colon \mathbb{N}^n \dashrightarrow \mathbb{N}^n$ für alle $a_1, \dots, a_n \in \mathbb{N}$

- $\|x_i = x_\ell + z\|_n(a_1, \dots, a_n) = (a_1, \dots, a_{i-1}, a_\ell + z, a_{i+1}, \dots, a_n)$
- $\|P_1\,;\,P_2\|_n(a_1, \dots, a_n) = \|P_2\|_n(\|P_1\|_n(a_1, \dots, a_n))$
- $\|\mathbf{WHILE}(x_i \neq 0)\,\{P'\}\|_n(a_1, \dots, a_n)$

$$= \begin{cases} \|P'\|_n^t(a_1, \dots, a_n) & \text{falls } t \in \mathbb{N} \text{ existiert und für alle } s < t \\ & \pi_i^{(n)}(\|P'\|_n^t(a_1, \dots, a_n)) = 0 \\ & \pi_i^{(n)}(\|P'\|_n^s(a_1, \dots, a_n)) \neq 0 \\ \text{undef} & \text{sonst} \end{cases}$$

While-Programme

Semantik der While-Schleife

- Finde Iterationsanzahl t mit
  - x_i enthält 0 nach t Iterationen
  - x_i enthält nicht 0 nach $s < t$ Iterationen
- Gesamtberechnung *undefiniert* falls Teilberechnung *undefiniert*
  (undef = Endlosschleife)

Beispiele

- $\|x_2 = x_1 + 5\,;\, x_1 = x_3 + 1\|_3(0, 3, 7) = (8, 5, 7)$
- $\|\mathbf{WHILE}(x_1 \neq 0)\,\{x_2 = x_1 + 5\,;\, x_1 = x_3 + 1\}\|_3(0, 3, 7) = (0, 3, 7)$
- $\|\mathbf{WHILE}(x_1 \neq 0)\,\{x_2 = x_1 + 5\,;\, x_1 = x_3 + 1\}\|_3(1, 3, 7) = \text{undef}$

While-Programme

§5.17 Definition (berechnete Funktion; *computed function*)

While-Programm P mit $\max \operatorname{var}(P) = n$ **berechnet** k-stellige partielle Funktion $|P|_k \colon \mathbb{N}^k \dashrightarrow \mathbb{N}$ mit $k \leq n$ gegeben für alle $a_1, \dots, a_k \in \mathbb{N}$

$$|P|_k(a_1, \dots, a_k) = \pi_1^{(n)}\big(\|P\|_n(a_1, \dots, a_k, \underbrace{0, \dots, 0}_{(n-k)\text{ mal}})\big)$$

Notizen

- Eingaben $a_1, \dots, a_k$ in ersten k Variablen $x_1, \dots, x_k$
- Weitere Variablen $x_{k+1}, \dots, x_n$ initial 0
- Auswertung Programm mit dieser initialen Variablenbelegung
- Ergebnis ist Inhalt erster Variable x_1 nach Ablauf

While-Berechenbarkeit

§5.18 Definition (While-Berechenbarkeit; *While-computability*)

Partielle Funktion $f\colon \mathbb{N}^k \dashrightarrow \mathbb{N}$ **While-berechenbar**
falls While-Programm P mit $f = |P|_k$ existiert

While-Berechenbarkeit

Vollständig undefinierte partielle Funktion

$x_1 = x_1 + 1$ $\qquad (x_1 > 0)$

$\mathbf{WHILE}(x_1 \neq 0)\,\{x_1 = x_1 + 1\}$ $\qquad (x_1 > 0)$

Auswertung für $a \in \mathbb{N}$

$$\begin{aligned} &\|x_1 = x_1 + 1\,;\, \mathbf{WHILE}(x_1 \neq 0)\,\{x_1 = x_1 + 1\}\|_1(a) \\ =\ &\|\mathbf{WHILE}(x_1 \neq 0)\,\{x_1 = x_1 + 1\}\|_1(a+1) \\ =\ &\text{undef} \end{aligned}$$

da $\|x_1 = x_1 + 1\|_1^t(a+1) = (a+1+t)$ für alle $t \in \mathbb{N}$

While-Berechenbarkeit

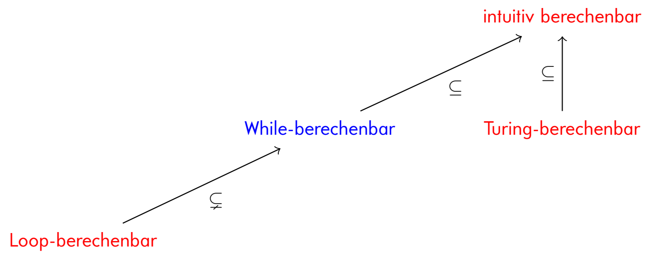
Iteration (Simulation von **LOOP**) $\qquad (x_\ell$ unbenutzt)

$x_\ell = x_i\,;\, \mathbf{WHILE}(x_\ell \neq 0)\,\{P'\,;\, x_\ell = x_\ell - 1\}$ Schreibweise: $\mathbf{LOOP}(x_i)\,\{P'\}$

Notizen

- Jedes Loop-Programm damit auch While-Programm
  (für jedes Loop-Programm existiert äquivalentes While-Programm)
- Loop-berechenbare Funktionen sind also While-berechenbar
- Schreibweisen Loop-Programme erlaubt (**IF-THEN-ELSE**, etc.)
- Nicht jede While-berechenbare partielle Funktion Loop-berechenbar (z.B. vollständig undefinierte partielle Funktion)

While-Berechenbarkeit



While-Berechenbarkeit

Komplexere Bedingung (x_k unbenutzt)

$x_k = x_i + 1\,;\, x_k = x_k - x_\ell$ ($x_k = 0$ gdw. $x_i < x_\ell$)
$\textbf{WHILE}(x_k \neq 0)\,\{$
$\quad P'\,;\, x_k = x_i + 1\,;\, x_k = x_k - x_\ell$ ($x_k = 0$ gdw. $x_i < x_\ell$)
$\}$

Schreibweise: $\textbf{WHILE}(x_i \geq x_\ell)\,\{P'\}$

Ganzzahlige Division von x_i durch x_m in x_ℓ (x_k unbenutzt)

$x_\ell = 0\,;\, x_k = x_i$
$\textbf{WHILE}(x_k \geq x_m)\,\{$
$\quad x_\ell = x_\ell + 1\,;\, x_k = x_k - x_m$
$\}$

Schreibweise: $x_\ell = x_i \,\textbf{DIV}\, x_m$

While-Berechenbarkeit

Ganzzahliger Rest von x_i durch x_m in x_ℓ

$x_\ell = x_i$
$\textbf{WHILE}(x_\ell \geq x_m)\,\{x_\ell = x_\ell - x_m\}$

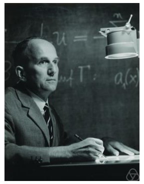
Schreibweise: $x_\ell = x_i \,\textbf{MOD}\, x_m$

Collatz-Iteration

$\textbf{WHILE}(x_1 > 1)\,\{$
$\quad \textbf{IF}(x_1 \,\textbf{MOD}\, 2 = 0)\,\{x_1 = x_1 \,\textbf{DIV}\, 2\}$ (halbiere x_1 falls gerade)
$\quad \textbf{ELSE}\,\{x_1 = 3 \cdot x_1 + 1\}$ (sonst verdreifache & addiere 1)
$\}$

Lothar Collatz (* 1910; † 1990)

- Dtsch. Mathematiker
- Formulierte ungelöste Collatz-Behauptung
- Ehrendoktorwürde TU Dresden



© Konrad Jacobs

While-Berechenbarkeit

Fallunterscheidung ($n \in \mathbb{N}$)

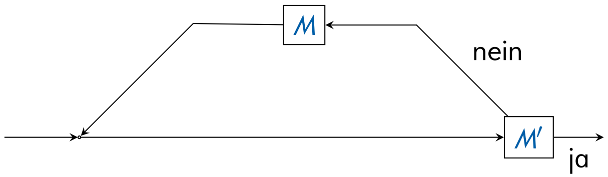
$\textbf{IF}(x_i = 0)\,\{P_0\}$ (Fall 0)
$\textbf{ELSE}\,\{\textbf{IF}(x_i - 1 = 0)\,\{P_1\}$ (Fall 1)
$\quad \textbf{ELSE}\,\{\cdots$
$\quad\quad \textbf{ELSE}\,\{\textbf{IF}(x_i - n = 0)\,\{P_n\}$ (Fall n)
$\quad\quad\quad \textbf{ELSE}\,\{P\}$
$\quad\quad \}$
$\quad\quad \cdots$
$\quad \}$
$\}$

Schreibweise: $\textbf{CASE}(x_i)\ \textbf{OF}\ 0 : \{P_0\} \ldots n : \{P_n\}\ \textbf{ELSE}\ \{P\}$

Bedingte Iteration

Notizen

- Turing-Berechenbarkeit benötigt deterministische TM
- Determinismus nicht erhalten unter Vereinigung & Iteration
- Bedingte Iteration = Abbruch Iteration bei Vorliegen Eigenschaft (bedingter Schleifenabbruch)



Bedingte Iteration

§6.1 Definition (bedingte Iteration; *conditional iteration*)

Seien $f\colon \Gamma^* \dashrightarrow \Gamma^*$ und $\chi\colon \Gamma^* \dashrightarrow \{0,1\}$ partielle Funktionen.
Bedingte Iteration $f_\chi\colon \Gamma^* \dashrightarrow \Gamma^*$ ist für alle $w \in \Gamma^*$

$$f_\chi^*(w) = \begin{cases} f^t(w) & \text{falls } t \in \mathbb{N} \text{ existiert mit} \\ & \chi(f^t(w)) = 1 \text{ und } \chi(f^s(w)) = 0 \text{ für alle } s < t \\ \text{undef} & \text{sonst} \end{cases}$$

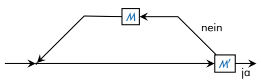
Bedingte Iteration

§6.2 Theorem

Seien $f\colon \Gamma^* \dashrightarrow \Gamma^*$ und $\chi\colon \Gamma^* \dashrightarrow \{0,1\}$ Turing-berechenbar.
Dann bedingte Iteration $f_\chi^*\colon \Gamma^* \dashrightarrow \Gamma^*$ Turing-berechenbar

Beweisansatz

1. Normierte det. TM M und M' für f und χ, wobei M' Band wiederherstellt und statt Ausgabe $1/0$ in Zustand q'_+/q'_- wechselt
2. Akzeptieren in q'_+ (ja-Zweig)
3. Starten M in q'_- (nein-Zweig)
4. Starten M' im akz. Zustand von M



Verzweigung

§6.2 Theorem

Seien $f\colon \Gamma^* \dashrightarrow \Gamma^*$ und $\chi\colon \Gamma^* \dashrightarrow \{0,1\}$ Turing-berechenbar.
Dann bedingte Iteration $f_\chi^*\colon \Gamma^* \dashrightarrow \Gamma^*$ Turing-berechenbar

Beweis

Seien $M = (Q, \Gamma', \Gamma, \Delta, \Box, q_0, q_+, q_-)$ und $M' = (Q', \Gamma', \Gamma, \Delta', \Box, q'_0, q'_+, q'_-)$ normierte det. TM mit $\Gamma' = \Gamma_M$, $T(M) = f$ und $T(M') = \chi$. Anpassung M' für Wiederherstellung Eingabe und Akzeptanz statt Ausgabe 1 und Ablehnung statt Ausgabe 0. O.B.d.A. sei $Q \cap Q' = \emptyset$. Wir konstruieren det. TM $N = (Q \cup Q', \Gamma', \Gamma, \Delta \cup \Delta' \cup R, \Box, q'_0, q'_+, q_-)$

$$R = \{(q'_-, \gamma) \to (q_0, \gamma, \diamond) \mid \gamma \in \Gamma\} \cup \{(q_+, \gamma) \to (q'_0, \gamma, \diamond) \mid \gamma \in \Gamma\}$$

Dann $T(N) = T(M)^*_{T(M')} = f_\chi^*$ $\Box$

While-Berechenbarkeit

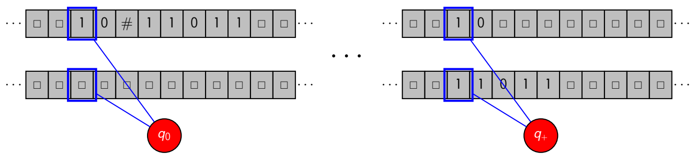
§6.3 Theorem

Jede While-berechenbare partielle Funktion ist Turing-berechenbar

Beweisskizze (1/4)

Sei P While-Programm mit $\max \operatorname{var}(P) = n$. Nutze 1 Band pro Variable und speichere x_i auf Band i. Wir konstruieren normierte det. TM

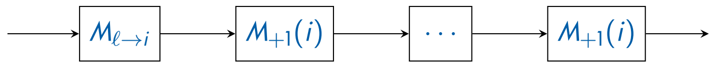
- M_{start} kopiert Startwerte auf korrekte Bänder
- Induktiv per Definition While-Programm



While-Berechenbarkeit

Beweisskizze (2/4)

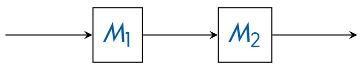
- Sei P Zuweisung $x_i = x_\ell + z$
- Simuliert durch
  1. Kopier-TM $M_{\ell \to i}$ kopiert Band ℓ auf Band i
  2. z mal Inkrement- oder Dekrement-TM auf Band i



While-Berechenbarkeit

Beweisskizze (3/4)

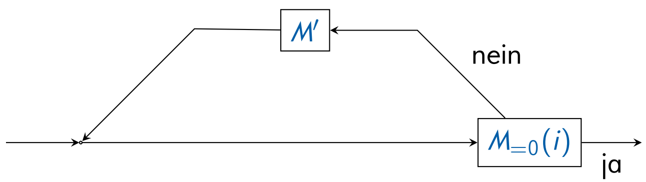
- Sei $P = P_1 \,;\, P_2$
- Simuliert durch Verkettung zugeh. det. TM M_1 und M_2



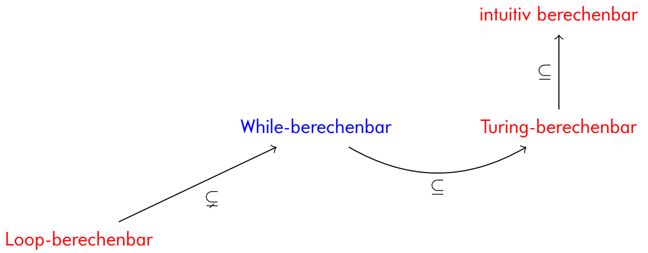
While-Berechenbarkeit

Beweisskizze (4/4)

- Sei $P = \textbf{WHILE}(x_i \neq 0)\,\{P'\}$
- Simuliert durch
  1. Sei M' det. TM für P'
  2. Sei $M_{=0}$ det. TM für Gleichheit mit 0
  3. Bedingte Iteration von M' mit Bedingung $M_{=0}(i)$ $\square$



While-Berechenbarkeit



Simulation Turingmaschine

Ansatz

- Simulation det. TM $(Q, \Sigma, \Gamma, \Delta, \Box, q_0, q_+, q_-)$ durch While-Programm
- Kodierung Zustand & Band benötigt
- Globalsituation $u\ q\ w$ in 3 Variablen: x_1 für w; x_2 für q; x_3 für u^R
- Nummerierung Zustände per Bijektion $h_Q \colon Q \to \{0, \ldots, |Q| - 1\}$ mit $h_Q(q_+) = 0$ und $h_Q(q_-) = 1$

Beispiel

- $Q = \{q_0, q, q_+, q_-\}$ kodiert via

$$q_0 \mapsto 3 \qquad q \mapsto 2 \qquad q_+ \mapsto 0 \qquad q_- \mapsto 1$$

Simulation Turingmaschine

Stellenwertsystem zur Basis $n = |\Gamma|$

- Nummerierung Symbole aus Γ per Bijektion $h_\Gamma : \Gamma \to \{0, \ldots, n-1\}$ mit $h_\Gamma(\Box) = 0$
- Kodiere Wort $w \in \Gamma^*$ im inversen Stellenwertsystem zur Basis n

$$\mathrm{code}_{h_\Gamma}(\gamma_1 \cdots \gamma_\ell) = \sum_{i=1}^{\ell} h_\Gamma(\gamma_i) \cdot n^{i-1}$$

Beispiel

- $\Gamma = \{\Box, a, b\}$ mit $h_\Gamma(\Box) = 0$, $h_\Gamma(a) = 1$ und $h_\Gamma(b) = 2$
- $w = abab$

$$\begin{aligned}\mathrm{code}_{h_\Gamma}(w) &= h_\Gamma(a) \cdot 3^0 + h_\Gamma(b) \cdot 3^1 + h_\Gamma(a) \cdot 3^2 + h_\Gamma(b) \cdot 3^3 \\ &= 1 + 2 \cdot 3 + 1 \cdot 9 + 2 \cdot 27 = 1 + 6 + 9 + 54 = 70\end{aligned}$$

Simulation Turingmaschine

Rechnen im Stellenwertsystem zur Basis n

- 1. Zeichen Kodierung k ist $h_\Gamma^{-1}(k\ \mathbf{MOD}\ n)$

$$h_\Gamma^{-1}(\mathrm{code}_{h_\Gamma}(\gamma w)\ \mathbf{MOD}\ n) = \gamma$$

<u>Schreibweise:</u> $\mathbf{TOP}(x_i) = x_i\ \mathbf{MOD}\ n$

- Entferne 1. Zeichen aus Kodierung k ist $k\ \mathbf{DIV}\ n$

$$\mathrm{code}_{h_\Gamma}(\gamma w)\ \mathbf{DIV}\ n = \mathrm{code}_{h_\Gamma}(w)$$

<u>Schreibweise:</u> $\mathbf{POP}(x_i) = x_i\ \mathbf{DIV}\ n$

- Einfügen γ als 1. Zeichen in Kodierung k ist $h_\Gamma(\gamma) + k \cdot n$

$$h_\Gamma(\gamma) + \mathrm{code}_{h_\Gamma}(w) \cdot n = \mathrm{code}_{h_\Gamma}(\gamma w)$$

<u>Schreibweise:</u> $\mathbf{PUSH}(x_i, z) = z + x_i \cdot n$

Simulation Turingmaschine

Sei $\Gamma = \{\square, a, b\}$ mit $h_\Gamma(\square) = 0$, $h_\Gamma(a) = 1$ und $h_\Gamma(b) = 2$

Beispiel

- Sei $x_1 = 70$ (Kodierung von "abab")
- $\mathbf{TOP}(x_1) = 70 \ \mathbf{MOD}\ 3 = 1$ (entspricht 'a')
- $\mathbf{POP}(x_1) = 70 \ \mathbf{DIV}\ 3 = 23$ (entspricht "bab" $[2 + 1 \cdot 3 + 2 \cdot 3^2]$)
- $\mathbf{PUSH}(x_1, 2) = 70 \cdot 3 + 2 = 212$ (entspricht "babab" $[2 + 1 \cdot 3 + 2 \cdot 3^2 + 1 \cdot 3^3 + 2 \cdot 3^4]$)

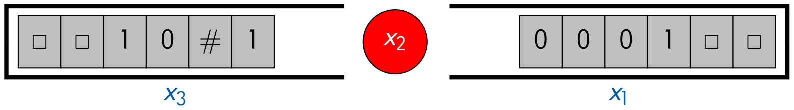
Notiz

- Kellerspeicher mit n Symbolen

Simulation Turingmaschine

Überblick

- Alle Komponenten beisammen
- Kodierung Zustand via h_Q (in x_2)
- Kodierung Band u links des Kopfes als Keller für u^R via h_Γ (in x_3)
- Kodierung sonstiges Band w als Keller via h_Γ (in x_1)



Simulation Turingmaschine

Hauptprogramm

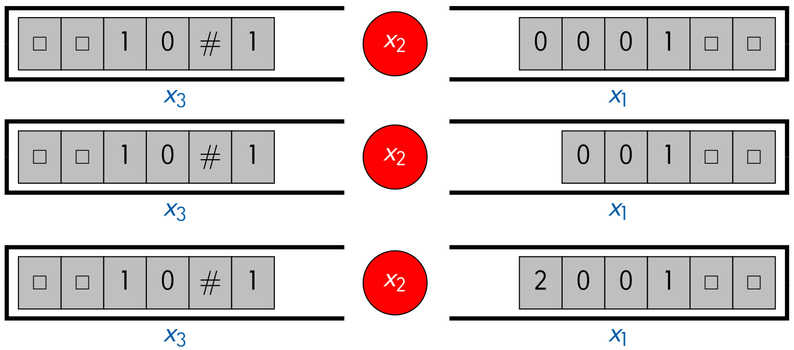
... Kodierung Eingabe in x_1 ...
$x_2 = h_Q(q_0)$; $x_3 = 0$ (Startzustand & leeres Band)
$\mathbf{WHILE}(x_2 > 1)\ \{$ (kein Endzustand)
  $\mathbf{CASE}(x_2, \mathbf{TOP}(x_1))\ \mathbf{OF}$ (Fallunterscheidung linke Seite Übergang)
  ...
  (q, γ): *... führe (q, γ)-Übergang aus ...*
  ...
  $\mathbf{ELSE}\ \{x_2 = h_Q(q_-)\}$
$\}$
... Dekodierung Band x_1 & Finalprüfung ...

Simulation Turingmaschine

Regelanwendung

Für jeden Übergang $(q, \gamma) \to (q', \gamma', d) \in \Delta$

$x_2 = h_Q(q')$ (Zustandswechsel)
$x_1 = \mathbf{POP}(x_1)$ (Entferne Zeichen unter Kopf)
$x_1 = \mathbf{PUSH}(x_1, h_\Gamma(\gamma'))$ (Füge neues Zeichen ein)

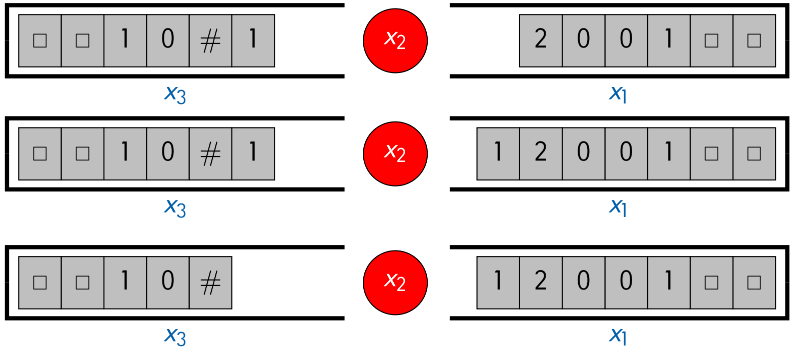


Simulation einer Turingmaschine

Regelanwendung

Falls $d = \triangleleft$ dann zusätzlich

$x_1 = \mathbf{PUSH}(x_1, \mathbf{TOP}(x_3))$ (Füge 1. Zeichen von links ein)
$x_3 = \mathbf{POP}(x_3)$ (Entferne 1. Zeichen von links)

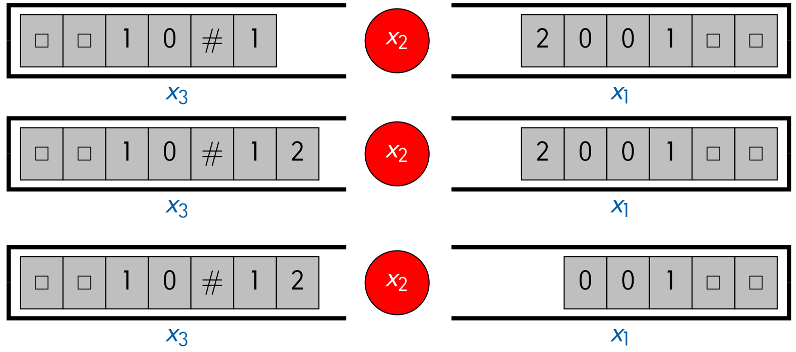


Simulation einer Turingmaschine

Regelanwendung

Falls $d = \triangleright$ dann zusätzlich

$x_3 = \mathbf{PUSH}(x_3, \mathbf{TOP}(x_1))$ (Füge 1. Zeichen von rechts ein)
$x_1 = \mathbf{POP}(x_1)$ (Entferne 1. Zeichen von rechts)



Simulation einer Turingmaschine

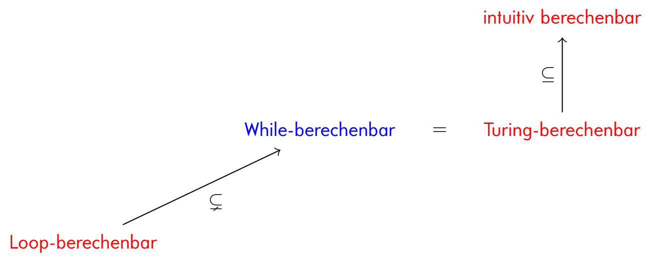
Banddekodierung für Binärzahl & Finalprüfung

$\mathbf{IF}(x_3 = 0 \text{ und } x_2 = 0)\,\{$ (teste linken Bandinhalt & Finalzustand)
  $x_4 = 0\,;\, x_5 = 0$ (Initialisierung Ausgabewert & Stellenwert)
  $\mathbf{WHILE}(x_1 \neq 0)\,\{$
    $\mathbf{IF}(1 \leq \mathbf{TOP}(x_1) \leq 2)\,\{$ (gültiges Bit [0 = □])
      $x_4 = x_4 + (\mathbf{TOP}(x_1) - 1) \cdot 2^{x_5}$ (dekodiere Binärdarstellung)
      $x_5 = x_5 + 1$ (nächste Stelle)
      $x_1 = \mathbf{POP}(x_1)$ (entferne erstes Bit)
    $\}$ **ELSE** … *Endlosschleife* …
  $\}$
  $x_1 = x_4$ (kopiere Ausgabewert)
$\}$ **ELSE** … *Endlosschleife* …

While-Berechenbarkeit

§6.4 Theorem

Jede Turing-berechenbare partielle Funktion ist While-berechenbar



Ackermann-Funktion

Kellerspeicher

- Implementiert für n Kellersymbole
- Speicherung Elemente von $\mathbb{N}$ per Binärkodierung über $\{0, 1, \#\}$
- Kellerende markiert durch 4. Symbol

Ackermann-Funktion

Einfügen natürliche Zahl

$\mathbf{PUSH}(x_i, 2)$ (Trennsymbol einfügen)
$x_\ell = x_k$ (kopiere x_k)
$\mathbf{WHILE}(x_\ell \neq 0)\ \{$
  $\mathbf{PUSH}(x_i, x_\ell\ \mathbf{MOD}\ 2)$ (letztes Bit speichern)
  $x_\ell = x_\ell\ \mathbf{DIV}\ 2$
$\}$

Schreibweise: $x_i = \mathbf{PUSH}_{\mathbb{N}}(x_i, x_k)$
($i \neq k$; x_ℓ unbenutzt)

Entfernen oberste natürliche Zahl

$\mathbf{WHILE}(\mathbf{TOP}(x_i) < 2)\ \{x_i = \mathbf{POP}(x_i)\}$ (entferne 0/1-Bits)
$\mathbf{IF}(\mathbf{TOP}(x_i) = 2)\ \{x_i = \mathbf{POP}(x_i)\}$ (teste auf & entferne Trennsymbol)
ELSE … *Endlosschleife* …

Schreibweise: $x_i = \mathbf{POP}_{\mathbb{N}}(x_i)$

Ackermann-Funktion

Auslesen oberste natürliche Zahl

$x_\ell = 0$ (initialisiere x_ℓ)
$\mathbf{WHILE}(\mathbf{TOP}(x_i) < 2)\ \{$ (bis Trenn- oder Endesymbol)
  $x_\ell = x_\ell \cdot 2 + \mathbf{TOP}(x_i)$ (dekodiere Binärzahl)
  $x_i = \mathbf{POP}(x_i)$ (erstes Bit entfernen)
$\}$
$\mathbf{IF}(\mathbf{TOP}(x_i) = 2)\ \{x_i = \mathbf{POP}(x_i)\}$ (teste auf & entferne Trennsymbol)
ELSE … *Endlosschleife* …
$x_i = \mathbf{PUSH}_{\mathbb{N}}(x_i, x_\ell)$ (Wert zurückschreiben)

Schreibweise: $x_\ell = \mathbf{TOP}_{\mathbb{N}}(x_i)$
($i \neq \ell$)

Ackermann-Funktion

Test Leerheit

$\mathbf{TOP}(x_i) \dot{-} 2$

- Liefert 1 falls leer
- Liefert 0 sonst

Schreibweise: $\mathbf{EMPTY}(x_i)$

Kellerspeicher für natürliche Zahlen

- Speicherung beliebiger natürliche Zahlen
- Unterstützung Standardoperationen
  (Test Leerheit, Einfügen, Entfernen, Auslesen)

Ackermann-Funktion

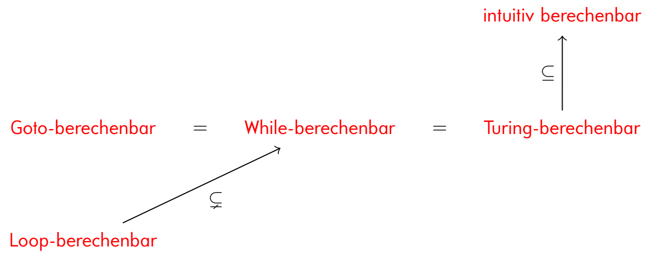
Implementation

$x_3 = 3$ (leerer Keller)
$\mathsf{PUSH}_{\mathbb{N}}(x_3, x_1) ; \mathsf{PUSH}_{\mathbb{N}}(x_3, x_2)$ (füge x_1 & x_2 ein)
$\mathsf{WHILE}(\mathsf{SIZE}_{\mathbb{N}}(x_3) > 1) \{$ (mind. 2 Elemente im Keller)
  $x_2 = \mathsf{TOP}_{\mathbb{N}}(x_3) ; x_3 = \mathsf{POP}_{\mathbb{N}}(x_3)$ (2. Parameter vom Keller)
  $x_1 = \mathsf{TOP}_{\mathbb{N}}(x_3) ; x_3 = \mathsf{POP}_{\mathbb{N}}(x_3)$ (1. Parameter vom Keller)
  $\mathsf{IF}(x_1 = 0) \{x_3 = \mathsf{PUSH}_{\mathbb{N}}(x_3, x_2 + 1)\}$ (liefere 2. Parameter + 1)
  $\mathsf{ELSE}\ \{$ (2. oder 3. Fall)
    $x_3 = \mathsf{PUSH}_{\mathbb{N}}(x_3, x_1 - 1)$ (Rekursion über 1. Parameter + 1)
    $\mathsf{IF}(x_2 = 0) \{x_3 = \mathsf{PUSH}_{\mathbb{N}}(x_3, 1)\}$ (2. Fall mit Konstante 1)
    $\mathsf{ELSE}\ \{x_3 = \mathsf{PUSH}_{\mathbb{N}}(x_3, x_1) ; x_3 = \mathsf{PUSH}_{\mathbb{N}}(x_3, x_2 - 1)\}$
  $\}$
$\}$
$x_1 = \mathsf{TOP}_{\mathbb{N}}(x_3)$ (Ergebnis im Keller)

Ackermann-Funktion

§6.5 Theorem

Ackermann-Funktion ist While-berechenbar



Rekursive Funktionen

Konventionen

- Partielle Funktionen des Typs $f \colon \mathbb{N}^k \dashrightarrow \mathbb{N}$
- Addition (und Subtraktion) weiterhin auf $\mathbb{N}$ begrenzt

Definition (§4.13 Projektion; *projection*)

Für $n \in \mathbb{N}$ und $1 \leq i \leq n$ ist $\pi_i^{(n)} \colon \mathbb{N}^n \to \mathbb{N}$ **n-stellige Projektion auf i-te Stelle**

$$\pi_i^{(n)}(a_1, \dots, a_n) = a_i \qquad a_1, \dots, a_n \in \mathbb{N}$$

Rekursive Funktionen

§6.6 Definition (rekursive Basisfunktionen; *recursive primitives*)

Folgende Funktionen sind **rekursive Basisfunktionen**

- n-stellige **a-konstante** Funktion $a^{(n)} \colon \mathbb{N}^n \to \mathbb{N}$
  mit $a^{(n)}(a_1, \dots, a_n) = a$ für alle $n, a, a_1, \dots, a_n \in \mathbb{N}$ (Konstanten)
- **Projektion** $\pi_i^{(n)} \colon \mathbb{N}^n \to \mathbb{N}$ für alle $n \in \mathbb{N}$ und $1 \leq i \leq n$
  (Projektionen)
- **Inkrementfunktion** $\mathrm{nf} \colon \mathbb{N} \to \mathbb{N}$
  mit $\mathrm{nf}(a) = a + 1$ für alle $a \in \mathbb{N}$ (Nachfolgerfunktion)

Keine weiteren rekursiven Basisfunktionen

Notizen

- Nutzung mathematischen Syntax & Funktionssemantik
- Basisfunktionen total (Vereinfachungen folgen)

Primitiv rekursive Funktionen

§6.7 Definition (primitiv rek. Fkt. [1/2]; *primitive rec. function*)

Genau folgende partielle Funktionen sind **primitiv rekursiv**

- Jede **rekursive Basisfunktion**
- Für alle $m, n \in \mathbb{N}$ und primitiv rekursiven partiellen Funktionen $f \colon \mathbb{N}^m \dashrightarrow \mathbb{N}$ und $g_1, \dots, g_m \colon \mathbb{N}^n \dashrightarrow \mathbb{N}$ ist **Komposition** $f\langle g_1, \dots, g_m\rangle \colon \mathbb{N}^n \dashrightarrow \mathbb{N}$ mit

$$f\langle g_1, \dots, g_m\rangle(a_1, \dots, a_n) = f\big(g_1(a_1, \dots, a_n), \dots, g_m(a_1, \dots, a_n)\big)$$

für alle $a_1, \dots, a_n \in \mathbb{N}$ primitiv rekursiv (Komposition)

Primitiv rekursive Funktionen

§6.7 Definition (primitiv rek. Fkt. [2/2]; *primitive rec. function*)

Genau folgende partielle Funktionen sind **primitiv rekursiv**

- Für alle $n \in \mathbb{N}$ und primitiv rekursiven partiellen Funktionen $f \colon \mathbb{N}^n \dashrightarrow \mathbb{N}$ und $g \colon \mathbb{N}^{n+2} \dashrightarrow \mathbb{N}$ ist durch **Schema primitive Rekursion** definierte partielle Funktion $\mathbf{pr}[f, g] \colon \mathbb{N}^{n+1} \dashrightarrow \mathbb{N}$

$$\begin{aligned} \mathbf{pr}[f, g](0, a_1, \dots, a_n) &= f(a_1, \dots, a_n) \\ \mathbf{pr}[f, g](a+1, a_1, \dots, a_n) &= g\big(\mathbf{pr}[f, g](a, a_1, \dots, a_n), a, a_1, \dots, a_n\big) \end{aligned}$$

für alle $a, a_1, \dots, a_n \in \mathbb{N}$ (primitive Rekursion)

Keine weitere primitiv rekursiven partiellen Funktionen

Primitiv rekursive Funktionen

n-stelliges Inkrement Komponente i $\quad \mathrm{nf}_i^{(n)} = \mathrm{nf}\langle \pi_i^{(n)}\rangle$

$$\mathrm{nf}_i^{(n)}(a_1, \dots, a_n) = \mathrm{nf}\big(\pi_i^{(n)}(a_1, \dots, a_n)\big) = \mathrm{nf}(a_i) = a_i + 1$$

Addition $\quad \mathrm{add} = \mathbf{pr}[\pi_1^{(1)}, \mathrm{nf}_1^{(3)}]$

$$\begin{aligned} \mathrm{add}(0, b) &= \pi_1^{(1)}(b) = b \\ \mathrm{add}(a+1, b) &= \mathrm{nf}_1^{(3)}\big(\mathrm{add}(a, b), a, b\big) = \mathrm{add}(a, b) + 1 \end{aligned}$$

Multiplikation $\quad \mathrm{mult} = \mathbf{pr}\big[0^{(1)}, \mathrm{add}\langle \pi_1^{(3)}, \pi_3^{(3)}\rangle\big]$

$$\begin{aligned} \mathrm{mult}(0, b) &= 0^{(1)}(b) = 0 \\ \mathrm{mult}(a+1, b) &= \mathrm{add}\Big(\pi_1^{(3)}\big(\mathrm{mult}(a, b), a, b\big), \pi_3^{(3)}\big(\mathrm{mult}(a, b), a, b\big)\Big) \\ &= \mathrm{add}\big(\mathrm{mult}(a, b), b\big) = \mathrm{mult}(a, b) + b \end{aligned}$$

Primitiv rekursive Funktionen

Vereinfachungen

- Direkte Verwendung Projektion (ohne explizite Angabe)
- Freie Verwendung Parameter
- Verwendung Makros & übliche Schreibweisen (für bereits als primitiv rekursiv bekannte Funktionen)
- Schreibweise "+1" statt nf

Primitiv rekursive Funktionen

Addition

wesentliche Rekursion $(a+1)+b=(a+b)+1$

$$\begin{aligned} \mathrm{add}(0,b) &= b \\ \mathrm{add}(a+1,b) &= \mathrm{add}(a,b)+1 \end{aligned}$$

Multiplikation

wesentliche Rekursion $(a+1)\cdot b=(a\cdot b)+b$

$$\begin{aligned} \mathrm{mult}(0,b) &= 0 \\ \mathrm{mult}(a+1,b) &= \mathrm{mult}(a,b)+b \end{aligned}$$

Primitiv rekursive Funktionen

Vorgänger

$\mathrm{vg} = \mathbf{pr}[0^{(0)}, \pi_2^{(2)}]$

$$\begin{aligned} \mathrm{vg}(0) &= 0 \\ \mathrm{vg}(a+1) &= a \end{aligned}$$

Subtraktion

$\mathrm{sub}' = \mathbf{pr}[\pi_1^{(1)}, \mathrm{vg}\langle\pi_1^{(3)}\rangle]$

wesentliche Rekursion $a-(b+1)=(a-b)-1$

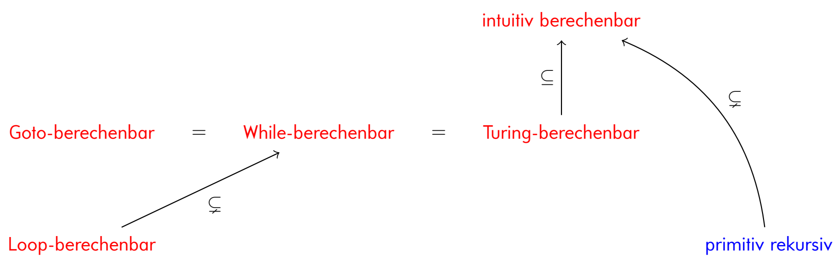
$$\begin{aligned} \mathrm{sub}'(0,a) &= a \\ \mathrm{sub}'(b+1,a) &= \mathrm{vg}(\mathrm{sub}'(b,a)) \end{aligned}$$

$$\mathrm{sub}(a,b) = \mathrm{sub}'(b,a) \qquad \mathrm{sub} = \mathrm{sub}'\langle\pi_2^{(2)}, \pi_1^{(2)}\rangle$$

Primitiv rekursive Funktionen

Notizen

- Primitiv rekursive Funktionen total
- Beschränkte Rekursion über 1 Argument
- Ähnlichkeit zu Loop-Programmen



Primitiv rekursive Funktionen

Falsche Variante der Subtraktion $\quad \mathrm{sub} = \mathbf{pr}\left[0^{(1)}, \mathrm{nf}\langle\pi_1^{(3)}\rangle\right]$

wesentliche Rekursion $(a+1) - b = (a-b) + 1$

$$\mathrm{sub}(0, b) = 0$$
$$\mathrm{sub}(a+1, b) = \mathrm{nf}(\mathrm{sub}(a, b))$$

Warum nicht gewünschte Funktion?

Denn $\mathrm{sub}(a, b) = a$ für alle $a, b \in \mathbb{N}$ (trivialer Induktionsbeweis)

Primitiv rekursive Funktionen

Berechnung für $\mathrm{sub}(2, 1)$

$$\begin{aligned}
\mathrm{sub}(2,1) &= \mathbf{pr}\left[0^{(1)}, \mathrm{nf}\langle\pi_1^{(3)}\rangle\right](2,1) \\
&= \mathrm{nf}\langle\pi_1^{(3)}\rangle\left(\mathbf{pr}\left[0^{(1)}, \mathrm{nf}\langle\pi_1^{(3)}\rangle\right](1,1),1,1\right) \\
&= \mathrm{nf}\left(\pi_1^{(3)}\left(\mathbf{pr}\left[0^{(1)}, \mathrm{nf}\langle\pi_1^{(3)}\rangle\right](1,1),1,1\right)\right) \\
&= \mathbf{pr}\left[0^{(1)}, \mathrm{nf}\langle\pi_1^{(3)}\rangle\right](1,1) + 1 \\
&= \mathrm{nf}\langle\pi_1^{(3)}\rangle\left(\mathbf{pr}\left[0^{(1)}, \mathrm{nf}\langle\pi_1^{(3)}\rangle\right](0,1),0,1\right) + 1 \\
&= \mathrm{nf}\left(\pi_1^{(3)}\left(\mathbf{pr}\left[0^{(1)}, \mathrm{nf}\langle\pi_1^{(3)}\rangle\right](0,1),0,1\right)\right) + 1 \\
&= \mathbf{pr}\left[0^{(1)}, \mathrm{nf}\langle\pi_1^{(3)}\rangle\right](0,1) + 2 \\
&= 0^{(1)}(1) + 2 = 0 + 2 = 2
\end{aligned}$$

Loop-Berechenbarkeit vs. primitive Rekursion

Ansatz Loop-berechenbar impliziert primitiv rekursiv

- Semantik $\|P\|_n \colon \mathbb{N}^n \to \mathbb{N}^n$ Loop-Programm P
- Primitiv rekursive Funktion $f \colon \mathbb{N}^n \to \mathbb{N}$
- Primitiv rekursive Variante $\|P\|_n$ benötigt Kodierung von $\mathbb{N}^n$ in $\mathbb{N}$ (z.B. Kellerspeicher)

Loop-Berechenbarkeit vs. primitive Rekursion

Binomialkoeffizient $\quad \mathrm{bk2} = \mathbf{pr}\left[0^{(0)}, \mathrm{add}\langle\pi_1^{(2)}, \pi_2^{(2)}\rangle\right]$

wesentliche Rekursion $\binom{a+1}{2} = \binom{a}{1} + \binom{a}{2} = a + \binom{a}{2}$

$$\mathrm{bk2}(0) = 0$$
$$\mathrm{bk2}(a+1) = a + \mathrm{bk2}(a)$$

Paarung $\quad c = \mathrm{add}\left\langle\pi_1^{(2)}, \mathrm{bk2}\left\langle\mathrm{nf}\left\langle\mathrm{add}\langle\pi_1^{(2)}, \pi_2^{(2)}\rangle\right\rangle\right\rangle\right\rangle$

$$c(a,b) = a + \mathrm{bk2}(a+b+1) = a + \binom{a+b+1}{2}$$

Loop-Berechenbarkeit vs. primitive Rekursion

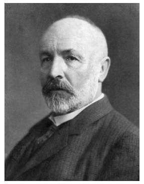
§7.1 Theorem (Cantorsche Paarungsfunktion)

$c\colon \mathbb{N}^2 \to \mathbb{N}$ bijektiv

$a \setminus b$	0	1	2	3	4	5
0	0	1	3	6	10	15
1	2	4	7	11	16	22
2	5	8	12	17	23	30
3	9	13	18	24	31	39
4	14	19	25	32	40	49
5	20	26	33	41	50	60

Georg Cantor (* 1845; † 1918)

- Dtsch. Mathematiker
- Begründer moderner Mengenlehre
- Kardinal- & Ordinalzahlen



Loop-Berechenbarkeit vs. primitive Rekursion

Notizen

- Offenbar $a \leq c(a,b)$ und $b \leq c(a,b)$ für alle $a, b \in \mathbb{N}$
- Kodierung Paare natürlicher Zahlen möglich

$$\begin{pmatrix} a \\ b \end{pmatrix} = c(a,b)$$

- Erweiterbar auf beliebige n-Tupel

$$\begin{pmatrix} a_1 \\ \vdots \\ a_n \end{pmatrix} = c\left(a_1, \begin{pmatrix} a_2 \\ \vdots \\ a_n \end{pmatrix}\right) = c\Big(a_1, c\big(a_2, \cdots, c(a_{n-1}, a_n) \cdots\big)\Big)$$

- Primitiv rekursiv für alle $n \in \mathbb{N}$

Loop-Berechenbarkeit vs. primitive Rekursion

Dekodierung

- Funktionen $\Pi_1, \Pi_2 \colon \mathbb{N} \to \mathbb{N}$

$$\Pi_1\big(c(a,b)\big) = \Pi_1 \begin{pmatrix} a \\ b \end{pmatrix} = a$$

$$\Pi_2\big(c(a,b)\big) = \Pi_2 \begin{pmatrix} a \\ b \end{pmatrix} = b$$

- Längere Tupel $\begin{pmatrix} a_1 \\ a_2 \\ a_3 \end{pmatrix} = \begin{pmatrix} a_1 \\ \begin{pmatrix} a_2 \\ a_3 \end{pmatrix} \end{pmatrix}$ ebenso dekodierbar

$$a_1 = \Pi_1 \begin{pmatrix} a_1 \\ a_2 \\ a_3 \end{pmatrix} \quad a_2 = \Pi_1\left(\Pi_2 \begin{pmatrix} a_1 \\ a_2 \\ a_3 \end{pmatrix}\right) \quad a_3 = \Pi_2\left(\Pi_2 \begin{pmatrix} a_1 \\ a_2 \\ a_3 \end{pmatrix}\right)$$

Loop-Berechenbarkeit vs. primitive Rekursion

§7.2 Definition (beschränkter max-Operator; *bounded maximum*)

Sei $P\colon \mathbb{N}^{n+1} \to \{0,1\}$ Prädikat und $\max \emptyset = 0$.

$$\max_P(a, a_1, \ldots, a_n) = \max\,\{b \leq a \mid P(b, a_1, \ldots, a_n) = 1\}$$

Notizen

- $\max_P(a, a_1, \ldots, a_n)$ maximaler Wert $b \leq a$ mit $P(b, a_1, \ldots, a_n) = 1$
- Liefert 0 falls kein Wert $b \leq a$ Prädikat $P(b, a_1, \ldots, a_n)$ erfüllt

Loop-Berechenbarkeit vs. primitive Rekursion

§7.3 Theorem

$\max_P \colon \mathbb{N}^{n+1} \to \mathbb{N}$ primitiv rek. falls $P \colon \mathbb{N}^{n+1} \to \{0,1\}$ primitiv rek.

Beweis

Offenbar $\max_P(0, a_1, \dots, a_n) = 0$ und

$$\max_P(a+1, a_1, \dots, a_n) = \begin{cases} a+1 & \text{falls } P(a+1, a_1, \dots, a_n) = 1 \\ \max_P(a, a_1, \dots, a_n) & \text{sonst} \end{cases}$$

Fallunterscheidung äquivalent zu

$$\max_P(a, a_1, \dots, a_n) + P(a+1, a_1, \dots, a_n) \cdot (a + 1 - \max_P(a, a_1, \dots, a_n))$$

□

$$\max_P = \mathbf{pr}\big[0^{(n)}, \mathrm{add}\langle\pi_1^{(n+2)}, \mathrm{mult}\langle P\langle \mathrm{nf}\langle\pi_2^{(n+2)}\rangle, \pi_3^{(n+2)}, \dots, \pi_{n+2}^{(n+2)}\rangle, \mathrm{sub}\langle \mathrm{nf}\langle\pi_2^{(n+2)}\rangle, \pi_1^{(n+2)}\rangle\rangle\rangle\big]$$

Loop-Berechenbarkeit vs. primitive Rekursion

§7.4 Definition (beschränkter ∃-Quantor; *bounded ∃-quantifier*)

Sei $P \colon \mathbb{N}^{n+1} \to \{0,1\}$ Prädikat

$$\exists_P(a, a_1, \dots, a_n) = \begin{cases} 1 & \text{falls } \exists b \leq a \colon P(b, a_1, \dots, a_n) = 1 \\ 0 & \text{sonst} \end{cases}$$

Notizen

- $\exists_P(a, a_1, \dots, a_n) = 1$, falls $b \leq a$ mit $P(b, a_1, \dots, a_n) = 1$ existiert
- Liefert 0 falls kein Wert $b \leq a$ Prädikat $P(b, a_1, \dots, a_n)$ erfüllt

Loop-Berechenbarkeit vs. primitive Rekursion

Maximum $\qquad \max = \mathrm{add}\langle \mathrm{sub}\langle\pi_1^{(2)}, \pi_2^{(2)}\rangle, \pi_2^{(2)}\rangle$

$$\max(a, b) = (a - b) + b$$

2 Fälle: Falls $a \geq b$, dann ist $(a - b) + b = a$ und damit $\max(a, b) = a$. Sonst ist $a - b = 0$ und damit $\max(a, b) = b$.

Loop-Berechenbarkeit vs. primitive Rekursion

§7.5 Theorem

$\exists_P \colon \mathbb{N}^{n+1} \to \mathbb{N}$ primitiv rekursiv falls $P \colon \mathbb{N}^{n+1} \to \{0,1\}$ primitiv rekursiv

Beweis

Offenbar $\exists_P(0, a_1, \dots, a_n) = P(0, a_1, \dots, a_n)$ und

$$\exists_P(a+1, a_1, \dots, a_n) = \begin{cases} 1 & \text{falls } P(a+1, a_1, \dots, a_n) = 1 \\ \exists_P(a, a_1, \dots, a_n) & \text{sonst} \end{cases}$$

Fallunterscheidung äquivalent zu

$$\max(P(a+1, a_1, \dots, a_n), \exists_P(a, a_1, \dots, a_n))$$

□

$$\exists_P = \mathbf{pr}\big[P\langle 0^{(n)}, \pi_1^{(n)}, \dots, \pi_n^{(n)}\rangle, \max\langle P\langle \mathrm{nf}\langle\pi_2^{(n+2)}\rangle, \pi_3^{(n+2)}, \dots, \pi_{n+2}^{(n+2)}\rangle, \pi_1^{(n+2)}\rangle\big]$$

Loop-Berechenbarkeit vs. primitive Rekursion

Dekodierung Paare $\binom{a}{b}$

- Definiere Prädikat $C\colon \mathbb{N}^3 \to \{0,1\}$ mit Hilfe von c

$$\boxed{C(a,b,d) = \Big(1 - \big(c(a,b) - d\big)\Big) \cdot \Big(1 - \big(d - c(a,b)\big)\Big)}$$

- C primitiv rekursiv
- $C(a,b,d) = 0$ falls $c(a,b) > d$
- $C(a,b,d) = 0$ falls $c(a,b) < d$
- $C(a,b,d) = 1$ falls $c(a,b) = d$
- Also $C(a,b,d) = 1$ gdw. $c(a,b) = d$

Loop-Berechenbarkeit vs. primitive Rekursion

Dekodierfunktionen

$$C'(b,a,d) = C(a,b,d) \qquad C' = C\langle \pi_2^{(3)}, \pi_1^{(3)}, \pi_3^{(3)} \rangle$$

$$E'(a,b,d) = \exists_{C'}(b,a,d) = \begin{cases} 1 & \text{falls } \exists y \leq b\colon C(a,y,d) = 1 \\ 0 & \text{sonst} \end{cases}$$

$$\begin{aligned} \Pi_1'(a,b,d) &= \max_{E'}(a,b,d) \\ &= \max\{x \leq a \mid E'(x,b,d) = 1\} \\ &= \max\{x \leq a \mid \exists y \leq b\colon C(x,y,d) = 1\} \\ &= \max\{x \leq a \mid \exists y \leq b\colon c(x,y) = d\} \end{aligned}$$

Loop-Berechenbarkeit vs. primitive Rekursion

Dekodierfunktionen

$$E(b,a,d) = \exists_C(a,b,d) = \begin{cases} 1 & \text{falls } \exists x \leq a\colon C(x,b,d) = 1 \\ 0 & \text{sonst} \end{cases}$$

$$\begin{aligned} \Pi_2'(a,b,d) &= \max_E(b,a,d) \\ &= \max\{y \leq b \mid E(y,a,d) = 1\} \\ &= \max\{y \leq b \mid \exists x \leq a\colon C(x,y,d) = 1\} \\ &= \max\{y \leq b \mid \exists x \leq a\colon c(x,y) = d\} \end{aligned}$$

Loop-Berechenbarkeit vs. primitive Rekursion

§7.6 Theorem (Dekodierung)

$\Pi_1\colon \mathbb{N} \to \mathbb{N}$ und $\Pi_2\colon \mathbb{N} \to \mathbb{N}$ primitiv rekursiv

Beweis

Beide Funktionen sind primitiv rekursiv da

$$\Pi_1(d) = \Pi_1'(d,d,d) \qquad \text{und} \qquad \Pi_2(d) = \Pi_2'(d,d,d) \quad \text{für alle } d \in \mathbb{N}$$

Da $a \leq c(a,b)$ und $b \leq c(a,b)$ wird d geeignet dekodiert. □

Notiz

- Verwenden Vektornotation und greifen direkt auf Komponenten zu

Loop-Berechenbarkeit vs. primitive Rekursion

§7.7 Theorem

Jede Loop-berechenbare Funktion ist primitiv rekursiv

Beweis (1/4)

Zeigen $\mathrm{sem}_P \colon \mathbb{N}^n \to \mathbb{N}$ primitiv rekursiv für jedes Loop-Programm P mit $\max \mathrm{var}(P) = n$ per Induktion über P. Für alle $a_1, \dots, a_n \in \mathbb{N}$

$$\begin{aligned} & \mathrm{sem}_P(a_1, \dots, a_n) = c(b_1, \dots, b_n) \\ \iff\ & \|P\|_n(a_1, \dots, a_n) = (b_1, \dots, b_n) \end{aligned}$$

- Sei P Zuweisung $x_i = x_\ell + z$. Dann

$$\mathrm{sem}_P(a_1, \dots, a_n) = c(a_1, \dots, a_{i-1}, a_\ell + z, a_{i+1}, \dots, a_n)$$

$\mathrm{sem}_P = c\big\langle \mathrm{nf}\langle \pi_2^{(2)} \rangle, \pi_2^{(2)} \big\rangle$ für $n = 2$ und $x_1 = x_2 + 1$

Loop-Berechenbarkeit vs. primitive Rekursion

Beweis (2/4)

- Sei $P = P_1 \,;\, P_2$. Dann sem_{P_1} und sem_{P_2} primitiv rekursiv gemäß IH

$$\begin{aligned} \mathrm{sem}_P(a_1, \dots, a_n) &= \mathrm{sem}_{P_2}\Big(\Pi_1\big(\mathrm{sem}_{P_1}(a_1, \dots, a_n)\big), \\ &\qquad \dots \\ &\qquad \Pi_2\big(\cdots \Pi_2\big(\mathrm{sem}_{P_1}(a_1, \dots, a_n)\big) \cdots\big)\Big) \\ &= \mathrm{sem}_{P_2}(b_1, \dots, b_n) \end{aligned}$$

mit $\mathrm{sem}_{P_1}(a_1, \dots, a_n) = c(b_1, \dots, b_n)$

$\mathrm{sem}_P = \big(\mathrm{sem}_{P_2}\langle \Pi_1, \Pi_2 \rangle\big)\langle \mathrm{sem}_{P_1} \rangle$ für $n = 2$

Loop-Berechenbarkeit vs. primitive Rekursion

Beweis (3/4)

- Sei $P = \mathbf{LOOP}(x_i)\,\{P'\}$. Dann $\mathrm{sem}_{P'}$ primitiv rekursiv gemäß IH. Definiere Funktion

$$\begin{aligned} f(0, a_1, \dots, a_n) &= c(a_1, \dots, a_n) \\ f(a + 1, a_1, \dots, a_n) &= \mathrm{sem}_{P'}(b_1, \dots, b_n) \end{aligned}$$

wobei $f(a, a_1, \dots, a_n) = c(b_1, \dots, b_n)$. Dann ist

$$\mathrm{sem}_P(a_1, \dots, a_n) = f(a_i, a_1, \dots, a_n)$$

$\mathrm{sem}_P = \Big(\mathrm{pr}\big[c, \big(\mathrm{sem}_{P'}\langle \Pi_1, \Pi_2 \rangle\big)\langle \pi_1^{(4)} \rangle\big]\Big)\langle \pi_2^{(2)}, \pi_1^{(2)}, \pi_2^{(2)} \rangle$ für $n = i = 2$

Loop-Berechenbarkeit vs. primitive Rekursion

Beweis (4/4)

Sei $f \colon \mathbb{N}^k \to \mathbb{N}$ Loop-berechenbar via P $\qquad (k \leq n)$

Für alle $a_1, \dots, a_k \in \mathbb{N}$

$$\begin{aligned} f(a_1, \dots, a_k) &= |P|_k(a_1, \dots, a_k) \\ &= \pi_1^{(n)}\big(\|P\|(a_1, \dots, a_k, 0, \dots, 0)\big) \\ &= \Pi_1\big(\mathrm{sem}_P(a_1, \dots, a_k, 0, \dots, 0)\big) \end{aligned}$$

Damit f primitiv rekursiv $\square$

Loop-Berechenbarkeit vs. primitive Rekursion

§7.8 Lemma

Für alle $k, k' \in \mathbb{N}$ und Loop-Programme P existiert Loop-Programm P' mit $\max \mathrm{var}(P') = n$ und

- $|P'|_k = |P|_k$ (gleiche berechnete Funktion)
- $\pi_i^{(n)}(\|P'\|_n(a_1, \dots, a_n)) = a_i$ (überschreibt $x_2, \dots, x_{k'}$ nicht) für alle $2 \le i \le k'$ und $a_1, \dots, a_n \in \mathbb{N}$

Loop-Berechenbarkeit vs. primitive Rekursion

§7.9 Theorem

Jede primitiv rekursive Funktion ist Loop-berechenbar

Beweis (1/3)

Induktion über Struktur primitiv rekursiver Funktionen

- Basisfunktionen: Trivial Loop-berechenbar
  (Konstanten, Projektionen, Nachfolger)

Loop-Berechenbarkeit vs. primitive Rekursion

Beweis (2/3)

- Sei $f' = f\langle g_1, \dots, g_m \rangle$ Komposition primitiv rekursiver Funktionen $f, g_1, \dots, g_m$

  $$f'(a_1, \dots, a_n) = f\big(g_1(a_1, \dots, a_n), \dots, g_m(a_1, \dots, a_n)\big)$$

  IH und §7.8 liefern äquivalente Loop-Programme $P, P_1, \dots, P_m$ die Variablen $x_2, \dots, x_{n+m+1}$ nicht überschreiben.
  Folgendes Programm berechnet f'

  $x_{n+m+1} = x_1$ (1. Eingabe sichern)
  P_m ; $x_{n+m} = x_1$; $x_1 = x_{n+m+1}$ (Ergebnis sichern; 1. Eingabe setzen)
  …
  P_1 ; $x_2 = x_{n+2}$; $\cdots$; $x_m = x_{n+m}$ (Eingaben auf Ergebnisse setzen)
  P

Loop-Berechenbarkeit vs. primitive Rekursion

Beweis (3/3)

- Sei $f' = \mathbf{pr}[f, g]$ primitive Rekursion mit primitiv rekursiven Funktionen f und g

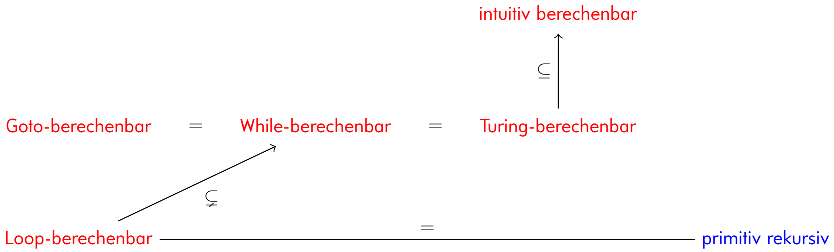
  $$f'(0, a_1, \dots, a_n) = f(a_1, \dots, a_n)$$
  $$f'(a+1, a_1, \dots, a_n) = g\big(f'(a, a_1, \dots, a_n), a, a_1, \dots, a_n\big)$$

  IH und Lemma §7.8 liefern äquivalente Loop-Programme P_f und P_g die Variablen $x_2, \dots, x_{n+4}$ nicht überschreiben.
  Folgendes Programm berechnet f'

  $x_{n+3} = x_1$; $x_{n+4} = x_2$ (1. & 2. Eingabe sichern)
  $x_1 = x_2$; $x_2 = x_3$; $\cdots$; $x_n = x_{n+1}$; P_f (Eingaben für f)
  $x_{n+2} = x_n$; $\cdots$; $x_4 = x_2$; $x_3 = x_{n+3}$; $x_2 = 0$ (Eingaben für g)
  LOOP(x_{n+3}) $\{P_g$; $x_2 = x_2 + 1\}$ (Iterationen zählen)

$\square$

Loop-Berechenbarkeit vs. primitive Rekursion



Notizen

- Primitiv rekursive Funktionen total
- Nicht jede While-berechenbaren Funktion primitiv rekursiv
  (z.B. Ackermann-Funktion nicht primitiv rekursiv)
- Allgemeine Rekursion noch nicht erfasst

Rekursive partielle Funktionen

§7.10 Definition (μ-rekursiv; *μ-recursive*)

Genau folgende partielle Funktionen sind **μ-rekursiv**

- **rekursive Basisfunktionen**
- **Komposition** und **primitive Rekursion** μ-rekursiver Funktionen
- **Minimierung** $\mu f\colon \mathbb{N}^n \dashrightarrow \mathbb{N}$ (μ-Operator) μ-rekursiver Funktion $f\colon \mathbb{N}^{n+1} \dashrightarrow \mathbb{N}$ gegeben für alle $a_1, \dots, a_n \in \mathbb{N}$ durch

$$\mu f(a_1, \dots, a_n) = \min \{a \in \mathbb{N} \mid f(a, a_1, \dots, a_n) = 0 \text{ und } \forall b < a\colon f(b, a_1, \dots, a_n) \text{ definiert}\}$$

mit $\min \emptyset = \text{undef}$

Keine weiteren partiellen Funktionen μ-rekursiv

Rekursive partielle Funktionen

Intuition

Berechnung von $\mu f(a_1, \dots, a_n)$

1. Setze $a \leftarrow 0$
2. Berechne $f(a, a_1, \dots, a_n)$
3. Liefere **undefiniert** falls $f(a, a_1, \dots, a_n)$ undefiniert
   (Endlosschleife)
4. Erhöhe a und zurück zu 2. falls $f(a, a_1, \dots, a_n) \neq 0$
5. Liefere a (falls $f(a, a_1, \dots, a_n) = 0$)

While-Schleife erkennbar

Rekursive partielle Funktionen

Überall undefinierte Funktion $\mu 1^{(2)}\colon \mathbb{N} \to \mathbb{N}$

- $1^{(2)}(a, b) = 1$ für alle $a, b \in \mathbb{N}$
- Also $\mu 1^{(2)}(b) = \text{undef}$ für alle $b \in \mathbb{N}$

Logarithmus $\text{ld}(13) = \lceil 3,7 \rceil = 4$

$\text{ld}\colon \mathbb{N} \to \mathbb{N}$ μ-rekursiv mit $\text{ld}(0) = 0$ und $\text{ld}(a) = \lceil \log_2(a) \rceil$ für alle $a \in \mathbb{N}_+$

- $f(a, b) = b - 2^a$ primitiv rekursiv (Loop-berechenbar)
- $\text{ld}(b) = \mu f(b)$ (kleinstes a mit $2^a \geq b$)

ld primitiv rekursiv

While-Berechenbarkeit vs. Rekursion

§7.11 Theorem

Jede While-berechenbare partielle Funktion ist μ-rekursiv

Beweis

Analog zu Loop-Programm mit neuem dritten Fall

- Sei $P = \mathbf{WHILE}(x_i \neq 0)\,\{P'\}$. Dann existiert μ-rekursive Funktion $\mathrm{sem}_{P'}$ gemäß IH. Sei

$$\begin{aligned} f(0, a_1, \dots, a_n) &= c(a_1, \dots, a_n) \\ f(a+1, a_1, \dots, a_n) &= \mathrm{sem}_{P'}(b_1, \dots, b_n) \\ g(a, a_1, \dots, a_n) &= b_i \end{aligned}$$

wobei $f(a, a_1, \dots, a_n) = c(b_1, \dots, b_n)$

$$\mathrm{sem}_P(a_1, \dots, a_n) = f\big((\mu g)(a_1, \dots, a_n), a_1, \dots, a_n\big) \qquad \square$$

While-Berechenbarkeit vs. Rekursion

§7.12 Theorem

Jede μ-rekursive partielle Funktion ist While-berechenbar

Beweis

Induktion über Struktur μ-rekursiver partieller Funktionen

- Sei $f' = \mu f$ für μ-rekursive Funktion f. Dann existiert äquivalentes While-Programm P ohne Überschreibung Variablen $x_2, \dots, x_{n+2}$. Programm für f'

$$x_{n+2} = 0 \,;\, x_{n+1} = x_n \,;\, \cdots \,;\, x_2 = x_1 \,;\, x_1 = 0 \,;\, P \qquad \text{(Eingaben setzen)}$$

$$\mathbf{WHILE}(x_1 \neq 0)\,\{x_{n+2} = x_{n+2} + 1 \,;\, x_1 = x_{n+2} \,;\, P\}$$

(Iteration erhöhen, Eingaben vorbereiten, weiterer Aufruf)

$$x_1 = x_{n+2} \qquad \text{(Ergebnis ist Anzahl Iterationen)}$$

$\square$

While-Berechenbarkeit vs. Rekursion

$$\begin{tikzcd}
 & & \text{intuitiv berechenbar} & \\
\text{Goto-berechenbar} \arrow[r, phantom, "="] & \text{While-berechenbar} \arrow[r, phantom, "="] & \text{Turing-berechenbar} \arrow[u, "\subseteq"] \arrow[r, phantom, "="] & \mu\text{-rekursiv} \\
\text{Loop-berechenbar} \arrow[ur, "\subsetneq"] \arrow[rrr, dash, "="] & & & \text{primitiv rekursiv}
\end{tikzcd}$$

Notizen

- Verschiedene Berechnungsmodelle & viele weitere existieren
- Alle höchstens Turing-Berechenbarkeit

These von Church

§7.13 Hypothese (These von Church; *Church's conjecture*)

Jede intuitiv berechenbare Funktion ist Turing-berechenbar

Alonzo Church (* 1903; † 1995)

- Amer. Mathematiker und Logiker
- Entwickelte λ-Kalkül (nicht vorgestellt)
- Doktorvater von Stephen Kleene & Alan Turing

Entscheidbarkeit

Grundlegende Fragen

- Was ist Problem?
- Wann entscheidbar?
- Wann semi-entscheidbar? (nur positive Fälle erfolgreich)
- Wann unentscheidbar?

Entscheidbarkeit

§8.1 Definition (Problem; *problem*)

Problem ist Teilmenge $L \subseteq \Sigma^*$ für Alphabet Σ

Notizen

- Entscheidungsprobleme sind ja/nein-Fragen
  (Ist geg. Graph planar? Ist geg. Zahl prim?)
- Identifikation solcher Probleme mit Teilmenge positiver Instanzen
  (z.B. planare Graphen $\subseteq$ Graphen, Primzahlen $\subseteq \mathbb{N}$)
- Kodierung aller Elemente über endlichem Alphabet
  (z.B. $\mathrm{dez}\colon \mathbb{N} \to \{0, \dots, 9\}^*$)
- Probleme sind Sprachen über Σ^*

Entscheidbarkeit

§8.2 Definition (Entscheidbarkeit; *decidability*)

Problem $L \subseteq \Sigma^*$ **entscheidbar** (engl. *decidable*) falls χ_L berechenbar

$$\chi_L\colon \Sigma^* \to \{0,1\} \quad \text{mit} \quad \chi_L(w) = \begin{cases} 1 & \text{falls } w \in L \\ 0 & \text{sonst} \end{cases} \quad \text{für alle } w \in \Sigma^*$$

L **unentscheidbar** (engl. *undecidable*) falls χ_L nicht berechenbar

Notizen

- χ_L = zugeh. Prädikat oder charakteristische Funktion von L
- Entscheidbar = zugeh. Prädikat (total und) berechenbar
  (keine Zeit- und Speicherbegrenzung)
- Teilmengen von $\mathbb{N}$, $\mathbb{N}^k$, etc. auch erlaubt

Entscheidbarkeit

Sei $G = (N, \Sigma, S, P)$ kontextsensitive Grammatik

Wortproblem Sprache $L(G)$

- Frage: Ist geg. $w \in \Sigma^*$ in $L(G)$?
- Problem $L = L(G)$
- Charakteristische Funktion $\chi_L\colon \Sigma^* \to \{0,1\}$ mit

$$\chi_L(w) = \begin{cases} 1 & \text{falls } w \in L(G) \\ 0 & \text{sonst} \end{cases}$$

- Berechenbarkeit von χ_L berechenbar
- Entscheidbarkeit von $L(G)$ entscheidbar

Entscheidbarkeit

§8.3 Theorem

Jede kontextsensitive Sprache $L \subseteq \Sigma^*$ ist entscheidbar

Beweisskizze

Sei $G = (N, \Sigma, S, P)$ kontextsensitive Grammatik mit $L(G) = L$.
Algorithmus für Berechnung χ_L mit Eingabe $w \in \Sigma^*$

1. Setze $\mathcal{F} = \{S\}$ (nur Startsymbol)
2. Setze $\mathcal{F}' = \mathcal{F} \cup \{v \in (N \cup \Sigma)^{\leq |w|} \mid \exists u \in \mathcal{F} \colon u \Rightarrow_G v\}$ (füge Nachfolger der Länge höchstens $|w|$ hinzu)
3. Falls $\mathcal{F} \subsetneq \mathcal{F}'$, dann setze $\mathcal{F} = \mathcal{F}'$ und gehe zu 2.
4. Liefere Wahrheitswert von $w \in \mathcal{F}'$ □

Entscheidbarkeit

Teilstrings von π

- Frage: Kommt w in Dezimalbruchdarstellung von π vor?
- Problem $L = \{w \in \{0, \dots, 9\}^* \mid w \text{ kommt in } \pi \text{ vor}\}$
- Charakteristische Funktion $\chi_L \colon \{0, \dots, 9\}^* \to \{0, 1\}$ mit

$$\chi_L(w) = \begin{cases} 1 & \text{falls } w \text{ in } \pi \text{ vorkommt} \\ 0 & \text{sonst} \end{cases}$$

- Berechenbarkeit von χ_L unklar
- Entscheidbarkeit von L unklar

Entscheidbarkeit

Für alle $n \in \mathbb{N}$ sei $\pi[n]$ Sequenz erste n Stellen in π

Initiale Teilstrings von π

- Frage: Beginnt Dezimalbruchdarstellung von π mit w?
- Problem $L = \{\pi[n] \mid n \in \mathbb{N}\}$
- Charakteristische Funktion $\chi_L \colon \{0, \dots, 9\}^* \to \{0, 1\}$ mit

$$\chi_L(w) = \begin{cases} 1 & \text{falls } w = \pi[n] \text{ für ein } n \in \mathbb{N} \\ 0 & \text{sonst} \end{cases}$$

- Berechenbarkeit von χ_L berechenbar
- Entscheidbarkeit von L entscheidbar

Approximation von π

Ramanujans Algorithmus (mit Genauigkeit n)

1. Setze $k = 0$ und $a = 0$
2. Erhöhe a um $\frac{(4k)! \cdot (1.103 + 26.390 \cdot k)}{(k!)^4 \cdot 396^{4k}}$
3. Erhöhe k um 1
4. Falls $8k \leq n$, dann gehe zu 2.
5. Liefere $\left(\frac{2\sqrt{2}}{9.801} \cdot a\right)^{-1}$

Srinivasa Ramanujan (* 1887; † 1920)

- Ind. Mathematiker
- Autodidakt mit über 3.900 Resultaten
- Analysis, Zahlentheorie, unendliche Reihen, etc.

Entscheidbarkeit

Algorithmus für $\sqrt{2}$

1. Setze $a_0 = 1$
2. Für alle $i \in \mathbb{N}$ sei $a_{i+1} = \frac{a_i}{2} + a_i^{-1}$

Notizen

- Verdoppelt Anzahl korrekter Stellen pro Schritt
  (1 Stelle für a_1; 3 Stellen für a_2; 6 Stellen für a_3; 12 Stellen für a_4)
- 10^{13} Stellen bekannt (ca. 4, 21 TB)
  (64 Bit erlaubt 19 Stellen; 128 Bit (IPv6) erlaubt 38 Stellen)

Entscheidbarkeit

§8.4 Theorem

Jede entscheidbare Sprache ist Typ-0

Beweis

Sei $L \subseteq \Sigma^*$ entscheidbare Sprache. Dann existiert (normierte) det. TM M mit $T(M) = \chi_L$. Wir modifizieren M so dass statt Ausgabe 0 mit Wechsel in akzeptierenden Zustand ablehnender Zustand eingenommen wird. Für erhaltene TM M'

$$w \in L(M') \quad \text{gdw.} \quad (T(M))(w) = \chi_L(w) = 1$$

und damit $L(M') = L$, womit L nach Theorem §4.3 vom Typ-0 □

Entscheidbarkeit

§8.5 Theorem

Für entscheidbare Sprache $L \subseteq \Sigma^*$ existiert det. TM $M = (Q, \Sigma, \Gamma, \Delta, \Box, q_0, q_+, q_-)$, so dass für jedes $w \in \Sigma^*$

- $\varepsilon\, q_0\, w \vdash_M^* u\, q_+\, v$ gdw. $w \in L$
- $\varepsilon\, q_0\, w \vdash_M^* u\, q_-\, v$ gdw. $w \notin L$

Notiz

- Entscheidbare Sprache L erlaubt det. TM, die
  - bei Worten aus L akzeptierenden Zustand erreicht
  - bei Worten außerhalb L ablehnenden Zustand erreicht

Entscheidbarkeit

§8.6 Theorem

Für entscheidbare Sprache $L \subseteq \Sigma^*$ ist auch $\overline{L} = \Sigma^* \setminus L$ entscheidbar

Beweis

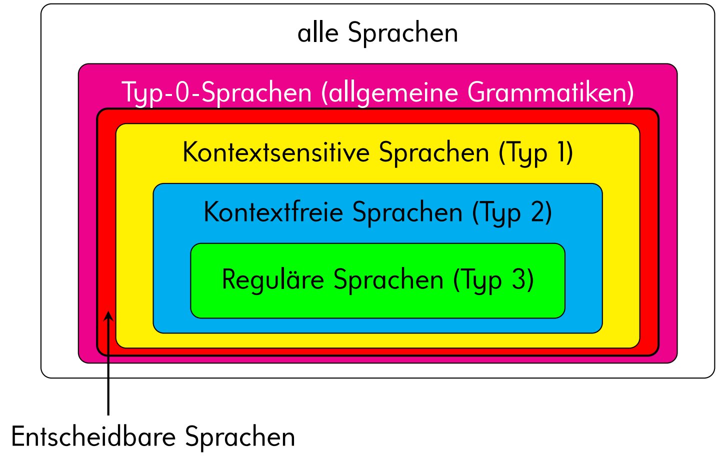
Sei P While-Programm, welches χ_L berechnet. Dann berechnet $P\,;\, x_1 = 1 - x_1$ charakteristische Funktion $\chi_{\overline{L}}$. □

Entscheidbarkeit

Notizen

- Kontextsensitive Sprachen entscheidbar
- Entscheidbare Sprachen sind Typ-0

Entscheidbarkeit



Semi-Entscheidbarkeit

§8.7 Definition (semi-entscheidbar; *semi-decidable*)

Problem $L \subseteq \Sigma^*$ **semi-entscheidbar** falls ρ_L berechenbar

$$\rho_L \colon \Sigma^* \dashrightarrow \{0, 1\} \qquad \text{mit} \qquad \rho_L(w) = \begin{cases} 1 & \text{falls } w \in L \\ \text{undef} & \text{sonst} \end{cases}$$

Notizen

- ρ_L = zugeh. Aufzählung
  (“halbe” (partielle) charakteristische Funktion von L)
- Semi-entscheidbar = zugeh. Aufzählung berechenbar
  (keine Zeit- und Speicherbegrenzung)
- Teilmengen von $\mathbb{N}$, $\mathbb{N}^k$, etc. auch erlaubt

Semi-Entscheidbarkeit

§8.8 Theorem

Für $L \subseteq \Sigma^*$ entscheidbar sind L und $\overline{L} = \Sigma^* \setminus L$ semi-entscheidbar

Beweis

Sei P While-Programm, welches χ_L berechnet. While-Programm

P
IF$(x_1 = 0)$ { ... *Endlosschleife* ... }

berechnet ρ_L und damit L semi-entscheidbar. Da L entscheidbar, ist auch $\overline{L}$ entscheidbar (Theorem §8.6) und damit semi-entscheidbar. □

Semi-Entscheidbarkeit

Sei $G = (N, \Sigma, S, P)$ Grammatik

Wortproblem Sprache $L(G)$

- Frage: Ist geg. $w \in \Sigma^*$ in $L(G)$?
- Problem $L = L(G)$
- Aufzählung $\rho_L \colon \Sigma^* \dashrightarrow \{0, 1\}$ mit

$$\rho_L(w) = \begin{cases} 1 & \text{falls } w \in L \\ \text{undef} & \text{sonst} \end{cases}$$

- Berechenbarkeit von ρ_L berechenbar
- Semi-Entscheidbarkeit von L semi-entscheidbar

Semi-Entscheidbarkeit

§8.9 Theorem

Jede Typ-0-Sprache L ist semi-entscheidbar

Beweis

Sei $G = (N, \Sigma, S, P)$ Grammatik mit $L(G) = L$ und $w \in \Sigma^*$.
Folgender Algorithmus berechnet ρ_L

1. Setze $\mathcal{F} = \{S\}$ (nur Startsymbol)
2. Setze $\mathcal{F}' = \mathcal{F} \cup \{v \in (N \cup \Sigma)^* \mid u \in \mathcal{F},\ u \Rightarrow_G v\}$ (füge alle Nachfolger hinzu)
3. Falls $w \in \mathcal{F}'$, dann liefere Ergebnis 1
4. Setze $\mathcal{F} = \mathcal{F}'$ und gehe zu 2. □

Semi-Entscheidbarkeit

§8.10 Theorem

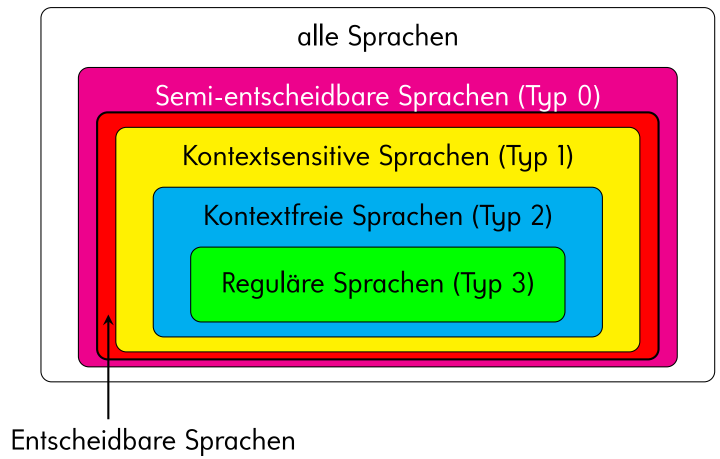
Typ-0-Sprachen = semi-entscheidbare Sprachen

Beweis

($\rightarrow$) Jede Typ-0-Sprache semi-entscheidbar via Theorem §8.9

($\leftarrow$) Sei L semi-entscheidbar. Es existiert det. TM M die ρ_L berechnet. Dann $L(M) = L$ und damit L Typ-0-Sprache via Theorem §4.3 □

Semi-Entscheidbarkeit



Semi-Entscheidbarkeit

Teilstrings von π

- Frage: Ist w Teilstring von π?
- Problem $L = \{w \in \{0, \dots, 9\}^* \mid w \text{ kommt in } \pi \text{ vor}\}$
- Aufzählung $\rho_L \colon \{0, \dots, 9\}^* \dashrightarrow \{0, 1\}$ mit

$$\rho_L(w) = \begin{cases} 1 & \text{falls } w \text{ in } \pi \text{ vorkommt} \\ \text{undef} & \text{sonst} \end{cases}$$

- Berechenbarkeit von ρ_L berechenbar
  (approximiere π und suche nach w in sicheren Stellen)
- Semi-Entscheidbarkeit von L semi-entscheidbar

Semi-Entscheidbarkeit

Nichtteilstrings von π

- Frage: Kommt w nicht in π vor?
- Problem $L = \{w \in \{0, \dots, 9\}^* \mid w \text{ kommt } \underline{\text{nicht}} \text{ in } \pi \text{ vor}\}$
- Aufzählung $\rho_L \colon \{0, \dots, 9\} \dashrightarrow \{0, 1\}$ mit

$$\rho_L(w) = \begin{cases} 1 & \text{falls } w \text{ } \underline{\text{nicht}} \text{ in } \pi \text{ vorkommt} \\ \text{undef} & \text{sonst} \end{cases}$$

- Berechenbarkeit von ρ_L unklar
- Semi-Entscheidbarkeit von L unklar

Semi-Entscheidbarkeit

Längen Nichtteilstrings von π

- Frage: Gibt Sequenz der Länge n die nicht in π vorkommt?
- Problem $L = \{|w| \mid w \in \{0, \dots, 9\}^* \text{ kommt } \underline{\text{nicht}} \text{ in } \pi \text{ vor}\}$
- Aufzählung $\rho_L \colon \mathbb{N} \dashrightarrow \{0, 1\}$ mit

$$\rho_L(n) = \begin{cases} 1 & \text{falls } \exists w \in \{0, \dots, 9\}^n \text{ } \underline{\text{nicht}} \text{ in } \pi \text{ vorkommend} \\ \text{undef} & \text{sonst} \end{cases}$$

- Berechenbarkeit von ρ_L berechenbar
  (falls alle Sequenzen in π vorkommen, dann ρ_L überall undefiniert;
  sonst existiert Sequenz kürzester Länge k und
  $\rho_L(n) = 1$ für alle $n \geq k$ und $\rho_L(n) = \text{undef}$ sonst)
- Semi-Entscheidbarkeit von L semi-entscheidbar
- Entscheidbarkeit von L entscheidbar

Semi-Entscheidbarkeit

§8.14 Theorem

Problem $L \subseteq \Sigma^*$ entscheidbar gdw. L und $\overline{L} = \Sigma^* \setminus L$ semi-entscheidbar

Beweis

Sei L entscheidbar. Dann L und $\overline{L}$ semi-entscheidbar via Theorem §8.8.
Seien L und $\overline{L}$ semi-entscheidbar und M und $\overline{M}$ TM die ρ_L und $\rho_{\overline{L}}$ berechnen. Für $w \in \Sigma^*$ berechnet folgender Algorithmus $\chi_L(w)$

1. $i \leftarrow 1$
2. Lasse TM M und $\overline{M}$ für i Schritte auf w laufen
3. Liefere 1, falls M akzeptiert (d.h. mit Ausgabe 1 terminiert)
4. Liefere 0, falls $\overline{M}$ akzeptiert
5. $i \leftarrow i + 1$ und gehe zu 2. $\square$

Rekursive Aufzählbarkeit

§8.15 Definition (rekursiv aufzählbar; *recursively enumerable*)

Problem L **rekursiv aufzählbar** falls $L = \emptyset$ oder berechenbare surjektive Funktion $a \colon \mathbb{N} \to L$ existiert

Notizen

- a zählt L auf da $L = \{a(i) \mid i \in \mathbb{N}\}$
- L rekursiv aufzählbar impliziert L abzählbar, denn (i) $L = \emptyset$ oder (ii) $a \colon \mathbb{N} \to L$ surjektiv implizieren Existenz injektiver Funktion $b \colon L \to \mathbb{N}$
- Rekursiv aufzählbar $\subsetneq$ abzählbar (keine Berechenbarkeitsforderung bei Abzählbarkeit)

Rekursive Aufzählbarkeit

§8.16 Theorem

Problem $L \subseteq \Sigma^*$ rekursiv aufzählbar gdw. semi-entscheidbar

Beweis (1/2)

Sei $L \neq \emptyset$ rekursiv aufzählbar. Dann existiert While-Programm P mit $\max \mathrm{var}(P) = n$ welches Aufzählung $a \colon \mathbb{N} \to L$ von L berechnet

$x_{n+1} = x_1 \,;\, x_1 = 0 \,;\, x_{n+2} = x_1$ (Eingabe sichern; Aufzählung initialisieren)
P (Element für 0 berechnen)
WHILE$(x_1 \neq x_{n+1})\,\{$ (solange x_{n+1} nicht erreicht)
  $x_1 = x_{n+2} + 1 \,;\, x_{n+2} = x_1 \,;\, P\}$ (nächstes Element vorbereiten)
$x_1 = 1$ (falls Eingabe gefunden, liefere Akzeptanz)

Rekursive Aufzählbarkeit

Beweis (2/2)

Sei $L \neq \emptyset$ semi-entscheidbar via det. TM M die ρ_L berechnet.
Bei Eingabe $n \in \mathbb{N}$ berechnet folgendes Programm $a(n)$

$x_2 = 0 \,;\, x_5 = x_1$ (kein Element gefunden)
WHILE$(x_2 = 0)\,\{$ (solange kein Element gefunden)
  $x_3 = \Pi_1(x_5) \,;\, x_4 = \Pi_2(x_5)$ (dekodiere x_5 als Paar (x_3, x_4))
  . . . Simuliere TM M auf Eingabe x_3 für x_4 Schritte . . .
  IF$(x_1 = 1)\,\{x_2 = 1 \,;\, x_1 = x_3\}$ (Element gefunden; Abbruch)
  ELSE $\{x_5 = x_5 + 1\}$ (nächster Versuch)
$\}$ □

Semi-Entscheidbarkeit

§8.17 Theorem

Für Sprache $L \subseteq \Sigma^*$ folgende Aussagen äquivalent

- L semi-entscheidbar
- L rekursiv aufzählbar
- $L = L(G)$ für (Typ-0-) Grammatik G
- $L = L(M)$ für TM M
- $L = L(M)$ für det. TM M

Unentscheidbarkeit

Kodierung TM

$$M = (\{0, 1, 2, \ldots, n\}, \{0, 1\}, \{0, 1, 2, \ldots, k\}, \Delta, 2, 0, 1, 2)$$

- Kodierung Übergang $(q, \gamma) \to (q', \gamma', d) \in \Delta$

$$\text{code}((q, \gamma) \to (q', \gamma', d)) = 1^q 0 1^\gamma 0 1^{q'} 0 1^{\gamma'} 0 1^{\text{bin}'(d)} 0$$

$$\text{bin}'(d) = \begin{cases} 1 & \text{falls } d = \triangleleft \\ 2 & \text{falls } d = \diamond \\ 3 & \text{sonst} \end{cases}$$

- Kodierung TM

$$\text{code}(M) = \prod_{\delta \in \Delta} \text{code}(\delta)$$

Unentscheidbarkeit

Beispiel

$$M = (\{q_0, q, q_a, q'_a, q_b, q'_b, q_+, q_-\}, \{a, b\}, \{a, b, \square\}, \Delta, \square, q_0, q_+, q_-)$$

mit Übergängen Δ

$$\begin{array}{lll}
(q_0, a) \to (q_a, \square, \triangleright) & (q_0, b) \to (q_b, \square, \triangleright) & (q_0, \square) \to (q_+, \square, \diamond) \\
(q_a, a) \to (q_a, a, \triangleright) & (q_a, b) \to (q_a, b, \triangleright) & (q_a, \square) \to (q'_a, \square, \triangleleft) \\
(q_b, a) \to (q_b, a, \triangleright) & (q_b, b) \to (q_b, b, \triangleright) & (q_b, \square) \to (q'_b, \square, \triangleleft) \\
(q'_a, a) \to (q, \square, \triangleleft) & (q'_b, b) \to (q, \square, \triangleleft) & \\
(q, a) \to (q, a, \triangleleft) & (q, b) \to (q, b, \triangleleft) & (q, \square) \to (q_0, \square, \triangleright)
\end{array}$$

Unentscheidbarkeit

Beispiel

$$M = (\{0, 1, 2, 3, 4, 5, 6, 7\}, \{0, 1\}, \{0, 1, 2\}, \Delta, 2, 0, 1, 2)$$

mit Übergängen Δ

$$\begin{array}{lll}
(0, 0) \to (3, 2, \triangleright) & (0, 1) \to (4, 2, \triangleright) & (0, 2) \to (1, 2, \diamond) \\
(3, 0) \to (3, 0, \triangleright) & (3, 1) \to (3, 1, \triangleright) & (3, 2) \to (5, 2, \triangleleft) \\
(4, 0) \to (4, 0, \triangleright) & (4, 1) \to (4, 1, \triangleright) & (4, 2) \to (6, 2, \triangleleft) \\
(5, 0) \to (7, 2, \triangleleft) & (6, 1) \to (7, 2, \triangleleft) & \\
(7, 0) \to (7, 0, \triangleleft) & (7, 1) \to (7, 1, \triangleleft) & (7, 2) \to (0, 2, \triangleright)
\end{array}$$

$$\text{code}(M) = \underbrace{001^3 01^2 01^3 0}_{(0,0) \to (3,2,\triangleright)} \underbrace{01^1 01^4 01^2 01^3 0}_{(0,1) \to (4,2,\triangleright)} \cdots$$

Unentscheidbarkeit

Konvention

- Zustände nummeriert ab 0
- Initialzustand 0, akzeptierender Zustand 1, ablehnender Zustand 2
- Arbeitssymbole nummeriert ab 0; Eingabesymbole $\mathfrak{B} = \{0, 1\}$
- Blanksymbol 2
- Betrachten bereinigte TM
  (jeder Zustand & jedes Symbol an mind. 1 Übergang beteiligt)
- Ausnahme Zustände & Symbole $\{0, 1, 2\}$ immer vorhanden
- Sequenz $\# = 00000$ kommt in keiner gültigen Kodierung vor

Unentscheidbarkeit

§9.1 Definition (Dekodierung; *decoding*)

Sei $\widehat{M}$ beliebige bereinigte det. TM über $\mathfrak{B}$ und
$\text{decode}\colon \mathfrak{B}^* \to \{M \mid M \text{ bereinigte det. TM über } \mathfrak{B}\}$ mit

$$\text{decode}(w) = \begin{cases} M & \text{falls } \text{code}(M) = w \\ \widehat{M} & \text{sonst} \end{cases} \qquad \text{für alle } w \in \mathfrak{B}^*$$

Notizen

- Invertierung Binärdarstellung
- Liefert Standard-TM für ungültige Binärdarstellungen

Unentscheidbarkeit

§9.2 Definition (Halteproblem; *halting problem*)

Halteproblem ist Sprache $\qquad (\# = 00000)$

$$H = \{c\#w \mid c \in \mathfrak{B}^* \setminus \mathfrak{B}^*\{\#\}\mathfrak{B}^*,\ \text{TM decode}(c) \text{ hält auf } w \in \mathfrak{B}^*\}$$

d.h. hält geg. bereinigte det. TM $\text{decode}(c)$ auf Eingabe $w \in \mathfrak{B}^*$?
(erreicht $\text{decode}(c)$ mit Eingabe w Endzustand)

Charakteristische Funktion $\chi_H\colon \mathfrak{B}^* \to \mathfrak{B}$

$$\chi_H(v) = \begin{cases} 1 & \text{falls } v = c\#w \text{ mit } c \in \mathfrak{B}^* \setminus \mathfrak{B}^*\{\#\}\mathfrak{B}^*,\ w \in \mathfrak{B}^* \text{ und} \\ & \qquad \text{decode}(c) \text{ auf } w \text{ hält} \\ 0 & \text{sonst} \end{cases}$$

Unentscheidbarkeit

§9.3 Definition (spez. Halteproblem; *special halting problem*)

Spezielle Halteproblem ist Sprache

$$\underline{H} = \{w \in \mathfrak{B}^* \mid \text{TM decode}(w) \text{ hält auf Eingabe } w\}$$

d.h. hält geg. TM $\text{decode}(w)$ auf (potentiell) eigener Kodierung w?

Charakteristische Funktion $\chi_{\underline{H}}\colon \mathfrak{B}^* \to \mathfrak{B}$

$$\chi_{\underline{H}}(w) = \begin{cases} 1 & \text{falls TM decode}(w) \text{ auf } w \text{ hält} \\ 0 & \text{sonst} \end{cases}$$

Unentscheidbarkeit

§9.4 Theorem (universelle TM; *universal Turing machine*)

Det. TM U die bei Eingabe $u\#w$ det. TM $\text{decode}(u)$ auf w simuliert

Notizen

- **Universelle Turingmaschine** U
- U hält auf $u\#w$ gdw. $\text{decode}(u)$ auf w hält
- U produziert auf $u\#w$ gleiche Ausgabe wie $\text{decode}(u)$ auf w

$$T(U) = \big\{(u\#w, v) \mid (w, v) \in T(\text{decode}(u))\big\}$$

Unentscheidbarkeit

§9.5 Theorem

Spezielles Halteproblem $\underline{H}$ unentscheidbar

Beweis (1/2)

Sei spezielles Halteproblem $\underline{H}$ entscheidbar. Dann existiert det. TM M für charakteristische Funktion $\chi_{\underline{H}}$. Sei P äquivalentes While-Programm. Betrachte Programm P'

P (berechne $\chi_{\underline{H}}$ von Eingabe)
IF$(x_1 \neq 0)$ $\{\ldots$ *Endlosschleife* $\ldots\}$ (falls decode(x_1) auf x_1 hält)
ELSE $\{x_1 = 1\}$ (liefere 1 falls decode(x_1) auf x_1 nicht hält)

Programm P' berechnet

$$\rho_{\overline{H}}(w) = \begin{cases} 1 & \text{falls } \chi_{\underline{H}}(w) = 0 \\ \text{undef} & \text{sonst} \end{cases}$$

Unentscheidbarkeit

Beweis (2/2)

Sei M' äquivalente det. TM zu P'. Betrachte Eingabe $w' = \text{code}(M')$

$$\begin{aligned} & M' = \text{decode}(w') \text{ hält auf } w' \\ \iff\ & \rho_{\overline{H}}(w') = 1 && \text{(da } M' \ \rho_{\overline{H}} \text{ berechnet)} \\ \iff\ & \chi_{\underline{H}}(w') = 0 && \text{(Def. } \rho_{\overline{H}}\text{)} \\ \iff\ & w' \notin \underline{H} && \text{(Def. } \chi_{\underline{H}}\text{)} \\ \iff\ & \text{decode}(w') \text{ hält auf } w' \text{ nicht} && \text{(Def. } \underline{H}\text{)} \end{aligned}$$

Widerspruch ↯ □

Unentscheidbarkeit

- Beweis nutzt Diagonalisierung
- Illustration Halteverhalten

$M \setminus w$	$f(0)$	$f(1)$	$f(2)$	$f(3)$	$\cdots$	$w' = \text{code}(M')$
decode$(f(0))$	$\boxed{✗}$	✗	✓	✓	$\cdots$	✓
decode$(f(1))$	✗	$\boxed{✓}$	✓	✗	$\cdots$	✓
decode$(f(2))$	✗	✗	$\boxed{✓}$	✗	$\cdots$	✗
decode$(f(3))$	✓	✓	✗	$\boxed{✓}$	$\cdots$	✓
$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$
$M' = \text{decode}(w')$	✓	✗	✗	✗	$\cdots$	$\boxed{?}$

M' hält auf w $\iff$ decode(w) auf w nicht hält

Problem-Reduktionen

Komposition oder Verkettung (§3.4)

- **Komposition** $f\colon \Sigma_1^* \dashrightarrow \Sigma_2^*$ und $g\colon \Sigma_2^* \dashrightarrow \Sigma_3^*$ ist $(f\,;g)\colon \Sigma_1^* \dashrightarrow \Sigma_3^*$

$$(f\,;g)(w) = g(f(w)) = \begin{cases} \text{undef} & \text{falls } f(w) = \text{undef} \\ g(f(w)) & \text{sonst} \end{cases}$$

§9.6 Theorem

$(f\,;g)$ berechenbar falls $f\colon \Sigma_1^* \dashrightarrow \Sigma_2^*$ und $g\colon \Sigma_2^* \dashrightarrow \Sigma_3^*$ berechenbar

Beweis

Verkettung det. TM für f und g (Sequenz While-Programme) □

Problem-Reduktionen

§9.7 Theorem

Sei $f\colon \Sigma^* \to \Gamma^*$ total und berechenbar und $K \subseteq \Gamma^*$.
Falls K entscheidbar, dann $f^{-1}(K)$ entscheidbar

Beweis

Da K entscheidbar, ist $\chi_K\colon \Gamma^* \to \{0,1\}$ berechenbar. Gemäß Theorem §9.6 ist $(f\,;\chi_K)\colon \Sigma^* \to \{0,1\}$ berechenbar.

$$(f\,;\chi_K)(w) = \chi_K(f(w)) = \begin{cases} 1 & \text{falls } f(w) \in K \\ 0 & \text{sonst} \end{cases}$$
$$= \begin{cases} 1 & \text{falls } w \in f^{-1}(K) \\ 0 & \text{sonst} \end{cases} = \chi_{f^{-1}(K)}(w)$$

Also $f^{-1}(K)$ entscheidbar $\square$

Problem-Reduktionen

Notizen

- Kontraposition von Theorem §9.7 ebenso interessant:

  Sei $f\colon \Sigma^* \to \Gamma^*$ total & berechenbar und $K \subseteq \Gamma^*$.
  Falls $f^{-1}(K)$ unentscheidbar, dann K unentscheidbar
- Betrachte berechenbare totale Funktion f
  - Sprache K entscheidbar $\to$ Urbild $f^{-1}(K)$ entscheidbar
  - Urbild $f^{-1}(K)$ unentscheidbar $\to$ Sprache K unentscheidbar

Problem-Reduktionen

§9.8 Definition (Reduktion; *reduction*)

Problem $L \subseteq \Sigma^*$ **reduzierbar** auf $K \subseteq \Gamma^*$, geschrieben $L \preceq K$, falls (totale) berechenbare Funktion $f\colon \Sigma^* \to \Gamma^*$ existiert mit $L = f^{-1}(K)$

Notizen

- $L = f^{-1}(K)$ entspricht Aussage

  $$w \in L \quad \text{gdw.} \quad f(w) \in K \qquad \text{für alle } w \in \Sigma^*$$
- f übersetzt Instanz Problem L in Instanz Problem K
  (Bestimmung "$w \in L$" per Bestimmung "$f(w) \in K$")
- $L \preceq K$ bedeutet "L höchstens so schwer wie K"
  (aktuell 2 Schwierigkeiten: entscheidbar & unentscheidbar)
- Berechenbarkeit & Totalität von f essentiell

Problem-Reduktionen

§9.9 Theorem

Seien $L \subseteq \Sigma^*$ und $K \subseteq \Gamma^*$ mit $L \preceq K$

- Falls K entscheidbar, dann L entscheidbar
  (entscheidbar falls leichter als entscheidbares Problem)
- Falls L unentscheidbar, dann K unentscheidbar
  (unentscheidbar falls schwerer als unentscheidbares Problem)

Problem-Reduktionen

§9.10 Theorem

Allgemeines Halteproblem H unentscheidbar

Beweis

Reduktion spezielles Halteproblem $\underline{H}$ auf H

$$\underline{H} = \{w \mid \mathrm{decode}(w) \text{ hält auf } w\} \qquad H = \{c\#w \mid \mathrm{decode}(c) \text{ hält auf } w\}$$

Benötigen berechenbare Funktion $f\colon \mathfrak{B}^* \to \mathfrak{B}^*$, die Elemente von $\underline{H}$ in Elemente von H übersetzt. Sei $f(w) = c\#w$ für alle $w \in \mathfrak{B}^*$ mit $c = w$ falls $w \in \mathfrak{B}^* \setminus \mathfrak{B}^*\{\#\}\mathfrak{B}^*$ und $c = \mathrm{code}(\widehat{M})$ sonst (klar berechenbar). Für alle $w \in \mathfrak{B}^*$ gelten $\mathrm{decode}(c) = \mathrm{decode}(w)$ und

$$w \in \underline{H} \iff \mathrm{decode}(w) \text{ hält auf } w \iff c\#w = f(w) \in H$$

Damit $\underline{H} = f^{-1}(H)$, $\underline{H} \preceq H$ und H unentscheidbar (Theorem §9.9) $\square$

Problem-Reduktionen

§9.11 Theorem

Leerband-Halteproblem $\{c \mid \mathrm{decode}(c) \text{ hält auf } \varepsilon\}$ unentscheidbar

Beweis

Wir reduzieren allgemeines Halteproblem H auf H_ε.

$$H = \{c\#w \mid \mathrm{decode}(c) \text{ hält auf } w\} \qquad H_\varepsilon = \{c \mid \mathrm{decode}(c) \text{ hält auf } \varepsilon\}$$

Sei $f(c\#w) = \mathrm{code}(M'_{c,w})$ für alle $c \in \mathfrak{B}^* \setminus \mathfrak{B}^*\{\#\}\mathfrak{B}^*$ und $w \in \mathfrak{B}^*$, wobei $M'_{c,w}$ det. TM die w auf Band schreibt, zurückläuft und $\mathrm{decode}(c)$ simuliert. Sonst sei $f(v) = \mathrm{code}(M_\perp)$ mit $M_\perp$ det. TM die nie hält (es gilt $v \notin H$ und $f(v) \notin H_\varepsilon$). Für alle $c \in \mathfrak{B}^* \setminus \mathfrak{B}^*\{\#\}\mathfrak{B}^*$ und $w \in \mathfrak{B}^*$

$$c\#w \in H \iff \mathrm{decode}(c) \text{ hält auf } w \iff \mathrm{code}(M'_{c,w}) \in H_\varepsilon$$

Damit $H = f^{-1}(H_\varepsilon)$, $H \preceq H_\varepsilon$ und H_ε unentscheidbar (Thm. §9.9) $\square$

Problem-Reduktionen

§9.12 Theorem (Satz von Rice)

Sei $\mathcal{R}$ Klasse aller berechenbaren partiellen Funktionen und $\mathcal{F} \subseteq \mathcal{R}$ mit $\emptyset \subsetneq \mathcal{F} \subsetneq \mathcal{R}$. Dann $\mathcal{C}(\mathcal{F})$ unentscheidbar

$$\mathcal{C}(\mathcal{F}) = \{w \in \mathfrak{B}^* \mid T(\mathrm{decode}(w)) \in \mathcal{F}\}$$

(Kodierungen aller det. TM, die Funktionen in $\mathcal{F}$ berechnen)

Henry Gordon Rice (* 1920; † 2003)

- Amer. Logiker & Mathematiker
- Bewies berühmten Satz in Dissertation
- Arbeitete zuletzt bei Computer Science Cooperation

Problem-Reduktionen

Beweis (1/3)

Sei $\perp = \emptyset \in \mathcal{R}$ überall undefinierte partielle Funktion auf $\mathfrak{B}^*$, die berechenbar ist und entweder $\perp \in \mathcal{F}$ oder $\perp \notin \mathcal{F}$.

Sei $\perp \in \mathcal{F}$. Da $\mathcal{F} \subsetneq \mathcal{R}$ existiert berechenbare partielle Funktion $g \in \mathcal{R} \setminus \mathcal{F}$. Sei M det. TM die g berechnet.

Wir reduzieren vom Komplement Halteproblem $\overline{H_\varepsilon}$ auf leerem Band. Sei $f\colon \mathfrak{B}^* \to \mathfrak{B}^*$ mit $f(w) = \mathrm{code}(M_w)$ für alle $w \in \mathfrak{B}^*$ und M_w det. 2-Band-TM die bei Eingabe $v \in \mathfrak{B}^*$

1. $\mathrm{decode}(w)$ auf leerem zweiten Band simuliert
2. Bei Akzeptanz danach TM M auf Eingabe v simuliert

$$T(M_w) = \begin{cases} \perp & \text{falls } \mathrm{decode}(w) \text{ auf } \varepsilon \text{ nicht hält (d.h. } w \notin H_\varepsilon) \\ g & \text{sonst (d.h. } w \in H_\varepsilon) \end{cases}$$

Problem-Reduktionen

Beweis (2/3)

Funktion f berechenbar. Wir zeigen $w \in \overline{H_\varepsilon}$ gdw. $f(w) \in \mathcal{C}(\mathcal{F})$

- Sei $w \notin \overline{H_\varepsilon}$. Dann $T(M_w) = g$ und $T(M_w) \notin \mathcal{F}$.
  Also $\mathrm{code}(M_w) \notin \mathcal{C}(\mathcal{F})$, womit $f(w) \notin \mathcal{C}(\mathcal{F})$.
- Sei $w \in \overline{H_\varepsilon}$. Dann $T(M_w) = \perp$ und $T(M_w) \in \mathcal{F}$.
  Also $\mathrm{code}(M_w) \in \mathcal{C}(\mathcal{F})$, womit $f(w) \in \mathcal{C}(\mathcal{F})$.

Also gilt Hilfsaussage, $\overline{H_\varepsilon} \preceq \mathcal{C}(\mathcal{F})$ und $\mathcal{C}(\mathcal{F})$ unentscheidbar da $\overline{H_\varepsilon}$ unentscheidbar (wäre $\overline{H_\varepsilon}$ entscheidbar, so wäre H_ε entscheidbar per Theorem §8.6; dies widerspricht Theorem §9.11)

Problem-Reduktionen

Beweis (3/3)

Sei $\perp \notin \mathcal{F}$. Da $\emptyset \subsetneq \mathcal{F}$ existiert partielle Funktion $g \in \mathcal{F}$. Sei M det. TM die g berechnet.

Wir reduzieren vom Halteproblem H_ε auf leerem Band und verwenden gleiche Funktion f wie vorher. Wir zeigen $w \in H_\varepsilon$ gdw. $f(w) \in \mathcal{C}(\mathcal{F})$

- Sei $w \in H_\varepsilon$. Dann $T(M_w) = g$ und $T(M_w) \in \mathcal{F}$.
  Also $\mathrm{code}(M_w) \in \mathcal{C}(\mathcal{F})$, womit $f(w) \in \mathcal{C}(\mathcal{F})$.
- Sei $w \notin H_\varepsilon$. Dann $T(M_w) = \perp$ und $T(M_w) \notin \mathcal{F}$.
  Also $\mathrm{code}(M_w) \notin \mathcal{C}(\mathcal{F})$, womit $f(w) \notin \mathcal{C}(\mathcal{F})$.

Damit gilt Hilfsaussage, $H_\varepsilon \preceq \mathcal{C}(\mathcal{F})$ und $\mathcal{C}(\mathcal{F})$ unentscheidbar da H_ε unentscheidbar (nach Theorem §9.11) $\square$

Problem-Reduktionen

Notizen

- $\mathcal{F}$ (nicht-triviale) Eigenschaft partieller Funktionen
  (z.B. total, surjektiv; nicht Eigenschaft der TM)
- Unentscheidbar, ob geg. TM Funktion mit Eigenschaft $\mathcal{F}$ berechnet
- Sehr mächtige Aussage
- Kein Programm kann Korrektheit (Äquivalenz) oder Termination (Reduktion von Akzeptanz) beliebiger Programme entscheiden

Problem-Reduktionen

§9.13 Theorem (Konsequenzen Satz von Rice)

Folgende Probleme unentscheidbar

- Universell akzeptierend $\{w \mid T(\mathrm{decode}(w)) \text{ total}\}$
  (Berechnet $\mathrm{decode}(w)$ Funktion?)
- Singulär akzeptierend $\{w \mid T(\mathrm{decode}(w)) \neq \emptyset\}$
  (Liefert $\mathrm{decode}(w)$ mind. 1 Ausgabe?)
- f-äquivalent $\{w \mid T(\mathrm{decode}(w)) = f\}$ für berechenbares f
  (Berechnet $\mathrm{decode}(w)$ genau partielle Funktion f?)
- Konstant $\{w \mid \exists u\colon T(\mathrm{decode}(w))(\{0,1\}^*) = \{u\}\}$
  (Berechnet $\mathrm{decode}(w)$ konstante partielle Funktion?)
- Nicht verkürzend $\{w \mid \forall v \forall u \in T(\mathrm{decode}(w))(\{v\})\colon |u| \geq |v|\}$
  (Ist Ausgabe immer mind. so lang wie Eingabe?)

Korrespondenzproblem von Post

§9.14 Definition (PCP und Lösung; *Post correspondence pairs*)

PCP sind Folge $P = \langle (u_1, w_1), \dots, (u_k, w_k) \rangle$ von Paaren nichtleerer Wörter $(u_i, w_i) \in \Sigma^+ \times \Sigma^+$.
Folge $(i_1, \dots, i_n)$ mit $n \geq 1$ und $i_1, \dots, i_n \in \{1, \dots, k\}$ ist **Lösung** der PCP P falls $u_{i_1} \cdots u_{i_n} = w_{i_1} \cdots w_{i_n}$

Emil Leon Post (* 1897; † 1954)

- Poln.-amer. Mathematiker & Logiker
- Entwickelte universelles Berechnungsmodell
- Korrespondenzproblem

Korrespondenzproblem von Post

Beispiel

- PCP $P = \langle (0, 101), (11, 00), (01, 1) \rangle$

Paar 1:	0	Paar 2:	11	Paar 3:	01
	101		00		1

- Unlösbar (keine Lösung) da alle Paare verschieden beginnen

Weiteres Beispiel

- PCP $P = \langle (0, 010), (1, 101), (0101, 01) \rangle$

Paar 1:	0	Paar 2:	1	Paar 3:	0101
	010		101		01

- Lösbar — Lösung $(3, 1)$ denn

  01010
  01010

Korrespondenzproblem von Post

Letztes Beispiel

- PCP $P = \langle (001, 0), (01, 011), (01, 101), (10, 001) \rangle$

Paar 1:	001	Paar 2:	01	Paar 3:	01	Paar 4:	10
	0		011		101		001

- Lösbar — minimale Lösung Länge 66
  $(2, 4, 3, 4, 4, 2, 1, 2, 4, 3, 4, 3, 4, 4, 3, 4, 4, 2, 1, 4, 4, 2, 1, 3, 4, 1, 1, 3, 4, 4, 4, 2, 1, 2, 1, 1, 1, 3, 4, 3, 4, 1, 2, 1, 4, 4, 2, 1, 4, 1, 1, 3, 4, 1, 1, 3, 1, 1, 3, 1, 2, 1, 4, 1, 1, 3)$

Korrespondenzproblem von Post

Korrespondenzproblem von Post

- Frage: Sind geg. PCP P lösbar?
- Problem $L = \{P \mid \text{PCP } P \text{ lösbar}\}$
- Aufzählung $\rho_L \colon \Sigma^* \dashrightarrow \{0, 1\}$ mit

$$\rho_L(P) = \begin{cases} 1 & \text{falls } P \text{ lösbar} \\ \text{undef} & \text{sonst} \end{cases}$$

- Berechenbarkeit von ρ_L berechenbar (Alle Indexfolgen probieren)
- Semi-Entscheidbarkeit von L semi-entscheidbar

§9.15 Theorem

Korrespondenzproblem von Post semi-entscheidbar

Modifiziertes Korrespondenzproblem von Post

§10.1 Definition (starke Lösung; *strong solution*)

Seien $P = \langle (u_1, w_1), \dots, (u_k, w_k) \rangle$ PCP.
Lösung $(i_1, \dots, i_n)$ der PCP P **stark** *(strong)* falls $i_1 = 1$

Modifiziertes Korrespondenzproblem von Post

- Frage: Sind geg. PCP P stark lösbar? (d.h. gibt es starke Lösung)
- Problem $L_{\text{MPCP}} = \{P \mid P \text{ stark lösbare PCP}\}$

Reduktion MPCP auf PCP

Illustration

- PCP $P = \langle (0101, 01), (1, 101), (0, 010) \rangle$

Paar 1:	0101	Paar 2:	1	Paar 3:	0
	01		101		010

- Lösbar — (schwache) Lösung $(2, 1)$ denn

$$\underbrace{1}_{2}\,\underbrace{0101}_{1} = \underbrace{101}_{2}\,\underbrace{01}_{1}$$

- Stark lösbar — starke Lösung $(1, 1, 3, 2)$ denn

$$\underbrace{0101}_{1}\,\underbrace{0101}_{1}\,\underbrace{0}_{3}\,\underbrace{1}_{2} = \underbrace{01}_{1}\,\underbrace{01}_{1}\,\underbrace{010}_{3}\,\underbrace{101}_{2}$$

Reduktion MPCP auf PCP

Idee

- Anfangs- & Zwischenmarkierung mit spez. Symbol $\#$
- Endmarkierung mit weiterem Symbol $\$$
- Seien $P = \langle (u_1, w_1), \dots, (u_k, w_k) \rangle$ PCP
  - Wort u_i erste Komponente $\#$ hinter jedes Symbol
  - Wort w_i zweite Komponente $\#$ vor jedes Symbol
- Jede Sequenz $w'_{i_1} \cdots w'_{i_n}$ beginnt mit $\#$, aber kein u_i beginnt mit $\#$
- 1 Kopie von u'_1 mit $\#$ am Anfang, Lösung muss damit beginnen

Reduktion MPCP auf PCP

Illustration

- PCP $P = \langle (0101, 01), (1, 101), (0, 010) \rangle$

Paar 1:	0101	Paar 2:	1	Paar 3:	0
	01		101		010

- Neue PCP

#0#1#0#1#	0#1#0#1#	1#	0#	$
#0#1	#0#1	#1#0#1	#0#1#0	#$

- Neue PCP nur starke Lösungen
- Originale PCP stark lösbar gdw. neue PCP lösbar

Reduktion MPCP auf PCP

§10.2 Theorem

$L_{\text{MPCP}} \preceq L_{\text{PCP}}$

Beweis (1/2)

Seien $P = \langle (u_1, w_1), \dots, (u_k, w_k) \rangle$ PCP und $\#, \$$ neue Symbole. Für jedes Wort $w = (\sigma_1, \dots, \sigma_n) \in \Sigma^*$ seien

$$\begin{aligned} {}^{\#}w &= \#\sigma_1\# \cdots \#\sigma_n && \text{(\# vor jedem Symbol)} \\ w^{\#} &= \sigma_1\# \cdots \#\sigma_n\# && \text{(\# hinter jedem Symbol)} \\ {}^{\#}w^{\#} &= \#\sigma_1\# \cdots \#\sigma_n\# && \text{(\# vor und hinter jedem Symbol)} \end{aligned}$$

Wir definieren Reduktion von MPCP auf PCP vermittels Funktion f

$$f(P) = \langle ({}^{\#}u_1^{\#}, {}^{\#}w_1), (u_1^{\#}, {}^{\#}w_1), \dots, (u_k^{\#}, {}^{\#}w_k), (\$, \#\$) \rangle$$

mit $k+2$ Elementen. f offensichtlich total und berechenbar

Reduktion MPCP auf PCP

Beweis (2/2)

$$f(P) = \langle ({}^{\#}u_1^{\#}, {}^{\#}w_1), (u_1^{\#}, {}^{\#}w_1), \dots, (u_k^{\#}, {}^{\#}w_k), (\$, \#\$) \rangle$$

Zu zeigen P stark lösbar gdw. $f(P)$ lösbar. Seien P stark lösbar und $(1, i_2, \dots, i_m)$ Lösung. Dann $(1, i_2 + 1, \dots, i_m + 1, k + 2)$ Lösung für $f(P)$

$$\begin{array}{cccccccccc} u_1 & u_{i_2} & \dots & u_{i_m} & & = & w_1 & w_{i_2} & \dots & w_{i_m} & \\ {}^{\#}u_1^{\#} & u_{i_2}^{\#} & \dots & u_{i_m}^{\#} & \$ & = & {}^{\#}w_1 & {}^{\#}w_{i_2} & \dots & {}^{\#}w_{i_m} & \#\$ \end{array}$$

Seien $f(P)$ lösbar und $(i_1, \dots, i_m)$ kürzeste Lösung. Dann $i_1 = 1$, $i_2, \dots, i_{m-1} \in \{2, \dots, k+1\}$ und $i_m = k + 2$.
Also $(1, i_2 - 1, \dots, i_{m-1} - 1)$ starke Lösung für P

$$\begin{array}{cccccccccc} {}^{\#}u_1^{\#} & u_{i_2}^{\#} & \dots & u_{i_{m-1}}^{\#} & \$ & = & {}^{\#}w_1 & {}^{\#}w_{i_2} & \dots & {}^{\#}w_{i_{m-1}} & \#\$ \\ u_1 & u_{i_2} & \dots & u_{i_{m-1}} & & = & w_1 & w_{i_2} & \dots & w_{i_{m-1}} & \square \end{array}$$

Reduktion Halteproblem auf MPCP

Idee

- 1. Paar für Initialsituation
- Kopiere Symbole & simuliere Ableitungsschritte
- 2. Komponente (unten) hat 1 Schritt Vorsprung

Illustration

$$\begin{aligned} &\$\square\square qabba\square\# \\ = {} &\$\square\square qabba\square\#\square\square\square q_a bba\square\square\#\square \end{aligned}$$

für Übergang $(q, a) \to (q_a, \square, \triangleright) \in \Delta$

Reduktion Halteproblem auf MPCP

§10.3 Theorem

$H_\varepsilon \preceq L_{\text{MPCP}}$ (Halteproblem auf leerem Band reduzierbar auf L_{MPCP})

Beweis (1/4)

Wir reduzieren vom Halteproblem auf leerem Band vermittels Funktion $f\colon \{0,1\}^* \to (V^+ \times V^+)^+$ mit

$$f(v) = \langle (u_1, w_1), \dots, (u_k, w_k) \rangle \qquad \text{für alle } v \in \{0,1\}^*$$

wobei $\text{decode}(v) = (Q, \Sigma, \Gamma, \Delta, \square, q_0, q_+, q_-)$ geeignet kodierte det. TM mit $Q \cup \Gamma \cup \{\$, \#\} \subseteq V$ und $\{\$, \#\} \cap (Q \cup \Gamma) = \emptyset$

Reduktion Halteproblem auf MPCP

Beweis (2/4)

Wir konstruieren PCP $f(v)$

1. $(u_1, w_1) = (\$, \$\square\square q_0\square\#)$ Initialsituation
2. Für alle $\gamma \in \Gamma$ existiert i mit $(u_i, w_i) = (\gamma, \gamma)$ Kopierpaare
3. Für alle $(q, \gamma) \to (q', \gamma', \diamond) \in \Delta$ existiert i mit $(u_i, w_i) = (q\gamma, q'\gamma')$
   Für alle $(q, \gamma) \to (q', \gamma', \triangleright) \in \Delta$ existiert i mit $(u_i, w_i) = (q\gamma, \gamma' q')$
   Für alle $(q, \gamma) \to (q', \gamma', \triangleleft) \in \Delta$ und $\gamma'' \in \Gamma$ existiert i mit
   $(u_i, w_i) = (\gamma'' q\gamma, q'\gamma''\gamma')$ Transitionspaare
4. Existiert i mit $(u_i, w_i) = (\#, \square\#\square)$ Erweiterung um $\square$
5. Für alle $\gamma \in \Gamma$ und $f \in \{q_+, q_-\}$ existiert i mit $(u_i, w_i) = (\gamma f, f)$
   Für alle $\gamma \in \Gamma$ und $f \in \{q_+, q_-\}$ existiert i mit $(u_i, w_i) = (f\gamma, f)$
   Löschregeln
6. Für alle $f \in \{q_+, q_-\}$ existiert i mit $(u_i, w_i) = (f\#\#, \#)$ Abschluss
7. Keine weiteren Paare in $f(v)$

Reduktion Halteproblem auf MPCP

Beweis (3/4)

Zu zeigen $\mathrm{decode}(v)$ hält auf leerem Band gdw. $f(v)$ stark lösbar
Zunächst halte $M = \mathrm{decode}(v)$ auf leerem Band. Dann existiert Folge Konfigurationen $\xi_1, \dots, \xi_n$ mit

- $\xi_1, \dots, \xi_n \in \Gamma^* Q \Gamma^*$ und $|\xi_i| = 2(i+1) + 1$
- $\square\square q_0\square \vdash_M \xi_1 \vdash_M \cdots \vdash_M \xi_n$
- $\xi_n \in \Gamma^*\{q_+, q_-\}\Gamma^*$

Lösungswort

$$\$\square\square q_0\square\#\xi_1\#\cdots\#\xi_n\#\xi_n^{(1)}\#\xi_n^{(2)}\#\cdots\#f\#\#$$

wobei $f \in \{q_+, q_-\}$, $\xi_n^{(0)} = \xi_n$ und $\xi_n^{(i)}$ aus $\xi_n^{(i-1)}$ entsteht indem Symbol links oder rechts vom Endzustand f gelöscht wird.

Reduktion des Halteproblems auf MPCP

Beweis (4/4)

Zu zeigen $\mathrm{decode}(v)$ hält auf leerem Band gdw. $f(v)$ stark lösbar
Umgekehrt sei $(i_1, \dots, i_n)$ starke Lösung von $f(v)$. Also $i_1 = 1$ und Lösungswort beginnt mit $\#\square\square q_0\square\#$. Damit beiden Sequenzen übereinstimmen müssen folgende Paare verwendet werden

1. Kopierpaare kopieren Bandinhalt bis Zustand (oder bis Zeichen vor Zustand) passend auf "oberen" String
2. Transitionspaar simuliert Übergang
   (Kopie Ausgangskonfiguration oben; Folgekonfiguration unten)
3. Kopierpaare kopieren verbleibenden Bandinhalt

Letztlich muss Endzustand erreichen, denn nur dessen Paare haben längere obere Sequenzen als untere Sequenzen. Damit erreicht also M Endzustand und hält auf leerem Band. □

Unentscheidbarkeit des PCP

§10.4 Theorem

Korrespondenzproblem von Post unentscheidbar

Beweis

Theorem §9.11 zeigt Halteproblem H_ε auf leerem Band unentscheidbar. Weiterhin $H_\varepsilon \preceq L_{\mathrm{MPCP}}$ (Theorem §10.3) und damit L_{MPCP} unentscheidbar nach Theorem §9.9. Außerdem $L_{\mathrm{MPCP}} \preceq L_{\mathrm{PCP}}$ (Theorem §10.2) und damit L_{PCP} unentscheidbar □

Schnittproblem kontextfreier Sprachen

Leerheit Schnitt kontextfreier Sprachen

- Frage: Ist $L(G) \cap L(G') \neq \emptyset$ für geg. kontextfreie Grammatiken G und G'?
- Problem $L_{\mathrm{CFI}} = \{\langle G, G'\rangle \mid G \text{ und } G' \text{ kontextfrei}, L(G) \cap L(G') \neq \emptyset\}$

Reduktion vom PCP

- Schnittsprache enthält Worte der Form $\underline{i}^{\mathrm{R}} u \$ w^{\mathrm{R}} \underline{\ell}$, wobei $\underline{i}$ und $\underline{\ell}$ Indexsequenzen und u und w korrespondierende Zeichenreihen
- 1. Sprache sichert Korrespondenz Indexsequenz & Zeichenreihe
- 2. Sprache sichert Gleichheit Indexsequenzen $\underline{i}$ und $\underline{\ell}$ und Gleichheit Zeichenreihen u und w

Schnittproblem kontextfreier Sprachen

§10.5 Theorem

$L_{\mathrm{PCP}} \preceq L_{\mathrm{CFI}}$

Beweis (1/2)

Seien $P = \langle (u_1, w_1), \dots, (u_k, w_k)\rangle$ PCP über Σ, $\Gamma = \Sigma \cup \{\$, 1, \dots, k\}$
Konstruiere 2 kontextfreie Grammatiken G und G' über Γ mit folgenden Produktionen für G

$$S \to A\$B \qquad A \to 1Au_1 \mid 1u_1 \mid \cdots \mid kAu_k \mid ku_k$$
$$B \to w_1^{\mathrm{R}}B1 \mid w_1^{\mathrm{R}}1 \mid \cdots \mid w_k^{\mathrm{R}}Bk \mid w_k^{\mathrm{R}}k$$

Sprache von G

$$L(G) = \{\underbrace{i_n \cdots i_1 u_{i_1} \cdots u_{i_n}}_{A} \$ \underbrace{(w_{\ell_1} \cdots w_{\ell_m})^{\mathrm{R}} \ell_1 \cdots \ell_m}_{B} \mid \cdots\}$$

Schnittproblem kontextfreier Sprachen

Beweis (2/2)

Grammatik G' verwendet folgende Produktionen

$$S \to 1S1 \mid \cdots \mid kSk \mid T \qquad T \to \$ \mid \sigma T \sigma \qquad \text{für alle } \sigma \in \Sigma$$

Sprache von G'

$$L(G') = \{u \underbrace{w\$w^{\mathrm{R}}}_{T} u^{\mathrm{R}} \mid u \in \{1, \dots, k\}^*, w \in \Sigma^*\}$$

Schnitt $L(G) \cap L(G') = \{\ell^{\mathrm{R}} w \$ w^{\mathrm{R}} \ell \mid \ell \text{ erzeugt beidseitig } w \text{ in } P\}$ womit jedes Element von $L(G) \cap L(G')$ Lösung samt Lösungswort repränsentiert. Damit P lösbar gdw. $L(G) \cap L(G') \neq \emptyset$ und damit $L_{\mathrm{PCP}} \preceq L_{\mathrm{CFI}}$ □

Schnittproblem kontextfreier Sprachen

§10.6 Theorem

Schnittproblem L_{CFI} kontextfreier Sprachen unentscheidbar

Beweis

Theorem §10.5 zeigt $L_{\mathrm{PCP}} \preceq L_{\mathrm{CFI}}$ und Korrespondenzproblem L_{PCP} von Post unentscheidbar nach Theorem §10.4. Also Schnittproblem L_{CFI} unentscheidbar nach Theorem §9.9 □

Schnittproblem kontextfreier Sprachen

Unendlichkeit Schnitts kontextfreier Sprachen

- Frage: Ist $L(G) \cap L(G')$ unendlich
  für geg. kontextfreie Grammatiken G und G'?
- Problem $L'_{\text{CFI}} = \{\langle G, G' \rangle \mid L(G) \cap L(G') \text{ unendlich}\}$
- Reduktion vom PCP wie bisher

  $$\text{PCP } P \text{ lösbar} \iff \text{PCP } P \text{ unendlich viele Lösungen}$$

  In Reduktion repräsentiert $L(G) \cap L(G')$ Lösungen und damit auch Reduktion von L_{PCP} auf L'_{CFI}

§10.7 Theorem

Unendlichkeitsproblem L'_{CFI} Schnitt kontextfreier Sprachen unentscheidbar

Inklusion kontextfreier Sprachen

Inklusion kontextfreier Sprachen

- Frage: Gilt $L(G') \subseteq L(G)$
  für geg. kontextfreie Grammatiken G' und G?
- Problem $L_{\text{CFT}} = \{\langle G', G \rangle \mid L(G') \subseteq L(G)\}$
- Offenbar $L(G') \cap L(G) \neq \emptyset$ gdw. $L(G') \not\subseteq \overline{L(G)}$
- Versuch Reduktion von L_{CFI} auf $\overline{L_{\text{CFT}}}$

  $$f(\langle G', G \rangle) = \langle G', \overline{G} \rangle$$

  mit $\overline{G}$ (Typ-0)-Grammatik für Komplement $\overline{L(G)}$; also $L(\overline{G}) = \overline{L(G)}$
  Funktion f ist total & berechenbar
- Allerdings

  $$f^{-1}(\overline{L_{\text{CFT}}}) = \{\langle G', G \rangle \in L_{\text{CFI}} \mid \overline{L(G)} \text{ kontextfrei}\} \subsetneq L_{\text{CFI}}$$

Inklusion kontextfreier Sprachen

Inklusion kontextfreier Sprachen

- Frage: Gilt $L(G') \subseteq L(G)$
  für geg. kontextfreie Grammatiken G' und G?
- Problem $L_{\text{CFT}} = \{\langle G', G \rangle \mid L(G') \subseteq L(G)\}$
- Reduktion f von L_{PCP} auf L_{CFI}; sei $f(P) = \langle G_1, G_2 \rangle$
  Reduktion g von L_{PCP} auf $\overline{L_{\text{CFT}}}$ per $g(P) = \langle G_1, \overline{G_2} \rangle$
  (Komplement $\overline{L(G_2)}$ ebenso kontextfrei; siehe Übung)

  $$\begin{aligned} P \text{ lösbar} &\iff L(G_1) \cap L(G_2) \neq \emptyset &&\iff f(P) \in L_{\text{CFI}} \\ &\iff L(G_1) \not\subseteq L(\overline{G_2}) &&\iff g(P) \in \overline{L_{\text{CFT}}} \end{aligned}$$

  Also $L_{\text{PCP}} \preceq \overline{L_{\text{CFT}}}$

Inklusion kontextfreier Sprachen

§10.8 Theorem

Inklusionsproblem L_{CFT} kontextfreier Sprachen unentscheidbar

Beweis

Wir wissen $L_{\text{PCP}} \preceq \overline{L_{\text{CFT}}}$ und Korrespondenzproblem L_{PCP} von Post unentscheidbar (Theorem §10.4). Damit auch Komplement $\overline{L_{\text{CFT}}}$ Inklusionsproblem unentscheidbar (Theorem §9.9). Wäre L_{CFT} entscheidbar, dann Komplement $\overline{L_{\text{CFT}}}$ entscheidbar nach Theorem §8.6. Also Inklusionsproblem L_{CFT} unentscheidbar □

Gleichheit kontextfreier Sprachen

Äquivalenz kontextfreier Sprachen

- Frage: Gilt $L(G) = L(G')$
  für geg. kontextfreie Grammatiken G und G'?
- Problem $L_{\mathsf{CFE}} = \{\langle G, G' \rangle \mid L(G) = L(G')\}$
- Reduktion f von L_{PCP} auf L_{CFI}; sei $f(P) = \langle G_1, G_2 \rangle$
  Reduktion g von L_{PCP} auf $\overline{L_{\mathsf{CFE}}}$ per $g(P) = \langle G_1 \cup \overline{G_2}, \overline{G_2} \rangle$
  $(L(\overline{G_2}) = \overline{L(G_2)}$ und $L(G_1 \cup \overline{G_2}) = L(G_1) \cup \overline{L(G_2)})$

$$\begin{aligned} P \text{ lösbar} &\iff L(G_1) \cap L(G_2) \neq \emptyset \\ &\iff L(G_1) \not\subseteq L(\overline{G_2}) \\ &\iff L(G_1) \cup L(\overline{G_2}) \neq L(\overline{G_2}) \iff g(P) \in \overline{L_{\mathsf{CFE}}} \end{aligned}$$

Also $L_{\mathsf{PCP}} \preceq \overline{L_{\mathsf{CFE}}}$

Gleichheit kontextfreier Sprachen

§10.9 Theorem

Äquivalenzproblem L_{CFE} kontextfreier Sprachen unentscheidbar

Für kontextfreie Sprachen unentscheidbar

- Leerheit Schnitt
- Endlichkeit Schnitt
- Inklusion
- Äquivalenz
- Kontextfreiheit Komplements
- Regularität

Leerheit kontextsensitiver Sprachen

Leerheit kontextsensitiver Sprachen

- Frage: Ist $L(G) = \emptyset$ für geg. kontextsensitive Grammatik G?
- Problem $L_{\mathsf{CSE}} = \{G \mid L(G) = \emptyset\}$
- Reduktion von L_{CFI}

$$f(\langle G, G' \rangle) = G'' \qquad \text{mit} \qquad L(G'') = L(G) \cap L(G')$$

  (Kontextsensitive Sprachen sind unter Schnitt abgeschlossen)
- $L(G) \cap L(G') \neq \emptyset$ gdw. $f(\langle G, G' \rangle) \neq \emptyset$
  Also $L_{\mathsf{CFI}} \preceq \overline{L_{\mathsf{CSE}}}$
- Damit L_{CSE} unentscheidbar

Satz von Church

Erinnerung Prädikatenlogik erster Stufe ($\forall x$, $\exists x$, etc.)

§10.10 Theorem (Satz von Church)

Erfüllbarkeit geg. Formel Prädikatenlogik erster Stufe unentscheidbar

Alonzo Church (* 1903; † 1995)

- Amer. Mathematiker & Logiker
- Entwickelte λ-Kalkül (nicht vorgestellt)
- Doktorvater von Stephen Kleene & Alan Turing

© Princeton University

Arithmetik

§10.11 Definition (Arithmetische Terme; *arithmetic terms*)

Folgende Ausdrücke sind **arithmetische Terme**

- Jede natürliche Zahl $n \in \mathbb{N}$ und jede Variable $x \in X$
- $(t + t')$ und $(t \cdot t')$ für arithmetische Terme t, t'
- Keine weiteren arithmetischen Terme

Für Variablenbelegung $\theta\colon X \to \mathbb{N}$ sei

$$x\theta = \theta(x) \qquad n\theta = n \qquad x \in X;\ n \in \mathbb{N}$$
$$(t + t')\theta = t\theta + t'\theta \qquad (t \cdot t')\theta = t\theta \cdot t'\theta$$

Beispiele

- $t_1 = 5$ $\qquad t_1\theta = 5$
- $t_2 = (x_2 \cdot 3) + x_1$ $\qquad t_2\theta = 3\theta(x_2) + \theta(x_1)$
- $t_3 = (3 \cdot 2) + 0$ $\qquad t_3\theta = 6$

Arithmetik

§10.12 Definition (Arithmetische Formeln; *arithmetic formulas*)

Arithmetische Formeln sind

- $t = t'$ für arithmetische Terme t, t'
- $\neg F$ und $F \vee F'$ für arithmetische Formeln F, F'
- $\exists x F$ für $x \in X$ und arithmetische Formel F
- Keine weiteren arithmetischen Formeln

Variablenbelegung $\theta\colon X \to \mathbb{N}$ **erfüllt** Formel F, kurz $\theta \models F$, falls

$$\begin{aligned}
&\theta \models (t = t') && \text{gdw. } t\theta = t'\theta \\
&\theta \models \neg F && \text{gdw. } \theta \not\models F \\
&\theta \models (F \vee F') && \text{gdw. } \theta \models F \text{ oder } \theta \models F' \\
&\theta \models \exists x.F && \text{gdw. } n \in \mathbb{N} \text{ existiert mit } \theta_{[x\mapsto n]} \models F
\end{aligned}$$

Arithmetik

Beispiele (wir nutzen wie üblich auch $\forall x$ und $\wedge$)

- $2 + 3 = 5$ wahr
- $\forall x_1.\forall x_2.(x_1 \cdot x_2) = (x_2 \cdot x_1)$ wahr
- $\exists x_1.\forall x_2.(x_1 + x_2) = x_2$ wahr
- $\exists x_1.\forall x_2.(x_1 \cdot x_2) = (x_2 \cdot x_2)$ falsch

Notizen

- **Satz** = Formel ohne freie Variablenvorkommen
- Für Satz F und Variablenbelegungen θ, θ' gilt $\theta \models F$ gdw. $\theta' \models F$
- Satz F also wahr, kurz $\models F$, oder falsch, kurz $\not\models F$
- $\theta_{[x\mapsto n]}$ ist Variablenbelegung θ außer Zuordnung Wert n zu x

$$\theta_{[x\mapsto n]}(y) = \begin{cases} n & \text{falls } y = x \\ \theta(y) & \text{sonst} \end{cases}$$

Arithmetik

§10.13 Definition (arithm. repräsentierbar; *arithm. representable*)

Partielle Funktion $f\colon \mathbb{N}^k \dashrightarrow \mathbb{N}$ **arithmetisch repräsentierbar** falls arithmetische Formel F mit freien Variablen $x, x_1, \dots, x_k$ existiert, so dass für alle $n, n_1, \dots, n_k \in \mathbb{N}$

$$f(n_1, \dots, n_k) = n \quad \text{gdw. } \mathbf{0}_{[x\mapsto n, x_1\mapsto n_1, \dots, x_k\mapsto n_k]} \models F$$

Notizen

- $\mathbf{0}\colon X \to \mathbb{N}$ ist Variablenbelegung mit $\mathbf{0}(x) = 0$ für alle $x \in X$
- Falls für $n_1, \dots, n_k \in \mathbb{N}$ kein $n \in \mathbb{N}$ mit $\mathbf{0}_{[x\mapsto n, x_1\mapsto n_1, \dots, x_k\mapsto n_k]} \models F$ existiert, dann $f(n_1, \dots, n_k)$ undefiniert

§10.14 Theorem

While-berechenbare partielle Funktionen arithmetisch repräsentierbar

§10.15 Theorem

Wahre arithm. Sätze $\mathrm{WA} = \{F \mid \text{Satz } F, \models F\}$ <u>nicht</u> rekursiv aufzählbar

Beweis (per Widerspruch)

Sei WA rekursiv aufzählbar und $a \colon \mathbb{N} \to \mathrm{WA}$ berechenbare surjektive Funktion. Sei F beliebiger arithmetischer Satz. Entweder $\models F$ oder $\models \neg F$. Da a surjektiv, existiert Index $n \in \mathbb{N}$ mit $a(n) \in \{F, \neg F\}$. Also WA entscheidbar per Suche nach Index n.
Weiterhin L_{PCP} semi-entscheidbar (Theorem §9.15). Nach Theorem §10.14 existiert arithmetische Formel F', die $\rho_{L_{\mathrm{PCP}}}$ repräsentiert

$$\begin{aligned} P \in L_{\mathrm{PCP}} \iff \rho_{L_{\mathrm{PCP}}}(P) = 1 &\iff \mathbf{0}_{[x \mapsto 1, x_1 \mapsto P]} \models F' \\ &\iff F'[x \mapsto 1, x_1 \mapsto P] \in \mathrm{WA} \end{aligned}$$

Damit $L_{\mathrm{PCP}} \preceq \mathrm{WA}$ und WA unentscheidbar. Widerspruch ↯ □

Beweissystem

Notizen

- **Beweissystem** = Formelmenge $\mathcal{F}$ + Inferenzregeln $\mathcal{R}$
- **Beweis** = Folge $(F_1, \dots, F_n)$ von Formeln aus $\mathcal{F}$; für alle $1 \leq i \leq n$ Formel F_i vermittels Regel aus $\mathcal{R}$ aus $\{F_1, \dots, F_{i-1}\}$ herleitbar

Beweissystem

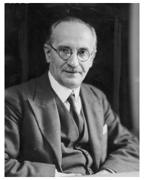
Łukasiewicz-Logik

- Formelmenge $\mathcal{F}$ = aussagenlogische Formeln über $\rightarrow$ und $\neg$
- Inferenzregeln $\mathcal{R}$

$$\begin{aligned} &\vdash F \rightarrow (F' \rightarrow F) \\ &\vdash (F \rightarrow (F' \rightarrow F'')) \rightarrow ((F \rightarrow F') \rightarrow (F' \rightarrow F'')) \\ &\vdash (\neg F \rightarrow \neg F') \rightarrow (F' \rightarrow F) \\ \{F, F \rightarrow F'\} &\vdash F' \qquad \text{(modus ponens)} \end{aligned}$$

Jan Łukasiewicz (* 1878; † 1956)

- Poln. Logiker & Philosoph
- Beiträge Aussagenlogik & mehrwertiger Logik
- Erfinder polnischer Notation



Beweissystem

§11.1 Definition (abstraktes Beweissystem; *abstract proof system*)

Abstraktes Beweissystem über Γ^* ist Paar $(\mathcal{B}, f)$ mit

- $\mathcal{B} \subseteq \Sigma^*$ entscheidbar (Menge gültiger Beweise)
- $f \colon \mathcal{B} \rightarrow \Gamma^*$ berechenbar und total (Zuordnung Beweis zu Aussage)

Łukasiewicz-Logik

- Beweise $\mathcal{B}$ offenbar entscheidbar
- Bewiesene Aussage = letztes Element des Beweises
  Damit f berechenbar

Beweissystem

§11.2 Definition (korrekt, vollständig; *sound, complete*)

Abstraktes Beweissystem $(\mathcal{B}, f)$ über Γ^* ist für Aussagen $\mathcal{T} \subseteq \Gamma^*$

- **korrekt** falls $f(B) \in \mathcal{T}$ für alle $B \in \mathcal{B}$
  (jeder Beweis "beweist" Aussage aus $\mathcal{T}$)
- **vollständig** falls $B \in \mathcal{B}$ mit $f(B) = F$ für alle $F \in \mathcal{T}$ existiert
  (jede Aussage aus $\mathcal{T}$ beweisbar)

Notiz

- Łukasiewicz-Logik für aussagenlog. Tautologien über $\rightarrow$ und $\neg$
  - Korrekt — nur Tautologien beweisbar
  - Vollständig — jede Tautologie beweisbar

Satz von Gödel

§11.3 Theorem (Unvollständigkeitssatz von Gödel)

Jedes abstrakte Beweissystem ist für WA inkorrekt oder unvollständig

Beweis

Sei $(\mathcal{B}, f)$ korrektes und vollständiges abstraktes Beweissystem für WA. Da $\mathcal{B} \neq \emptyset$ und $\mathcal{B}$ entscheidbar, ist $\mathcal{B}$ rekursiv aufzählbar. Also existiert $g \colon \mathbb{N} \to \mathcal{B}$ surjektiv und berechenbar. Dann $(g\,;f) \colon \mathbb{N} \to \Gamma^*$ berechenbar. Aus Korrektheit folgt $f \colon \mathcal{B} \to \mathrm{WA}$ und aus Vollständigkeit folgt Surjektivität von $f \colon \mathcal{B} \to \mathrm{WA}$. Also $(g\,;f) \colon \mathbb{N} \to \mathrm{WA}$ surjektiv. Damit WA rekursiv aufzählbar im Widerspruch zu Theorem §10.15 ↯ $\square$

Satz von Gödel

Konsequenzen

- Jedes vollständige Beweissystem für WA ist inkorrekt
- Jedes korrekte Beweissystem für WA ist unvollständig
  (nicht alle wahren Sätze von WA lassen sich beweisen)

Kurt Gödel (* 1906; † 1978)

- Öster.-amer. Logiker, Mathematiker & Philosoph
- Bedeutendster Logiker; Gödel-Nummern
- Widerlegte Hilbertsche Grundsatzprogramm
  (alle Sätze basierend auf Arithmetik ableitbar)

Komplexitätstheorie

Entscheidbarkeit

- Grundlegende Problem-Lösbarkeit
- Keine Beschränkung der Ressourcen (Zeit, Speicher)
- Entscheidbar $\neq$ praktisch lösbar

Komplexitätstheorie

- Obere & untere Schranken Ressourcen für jedwede Problemlösung
- Genauere Charakterisierung (Unterteilung) der Entscheidbarkeit
  (effizient, ineffizient lösbar, praktisch unlösbar, unentscheidbar)

Komplexitätstheorie

Problem des Handelsreisenden

- Geg. n Orte, Distanzmatrix $D \in \mathbb{N}^{n \times n}$ für Orte & Länge $\ell \in \mathbb{N}$
- Existiert Permutation $\pi = (\pi_1, \dots, \pi_n)$ von $(1, \dots, n)$ mit $D(\pi) \leq \ell$

$$D(\pi) = \left(\sum_{i=1}^{n-1} D_{\pi_i, \pi_{i+1}} \right) + D_{\pi_n, \pi_1} \quad \text{(Summe Distanzen in Rundreise)}$$

- Entscheidbar per Berechnung $D(\pi)$ für alle $n!$ Permutationen π
- Sei $n = 40$ und berechne $D(\pi)$ für 10^{11} Permutationen π pro s
- Laufzeit ca. $2{,}6 \cdot 10^{29}$ Jahre (Alter Universum ca. $1{,}4 \cdot 10^{10}$ Jahre)

Komplexitätstheorie

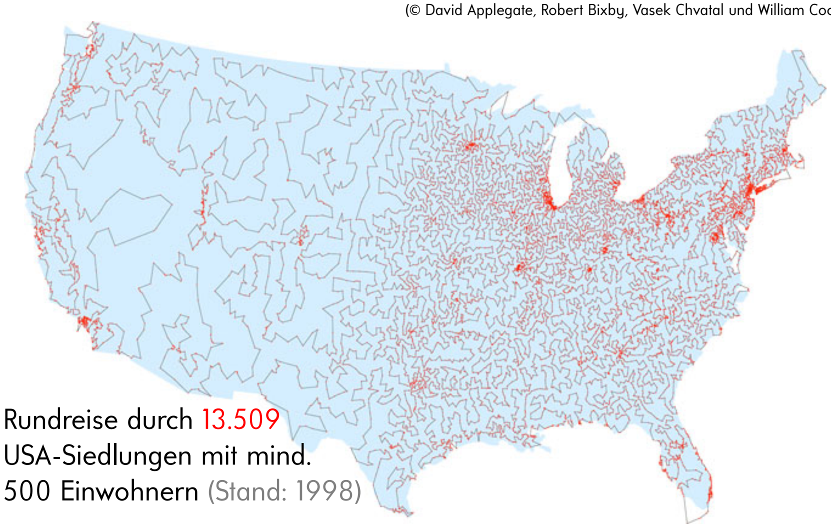
Optimale Rundreise durch 15 größte Städte Deutschlands

$15! \approx 1,3 \cdot 10^{12}$

(© Amirki)

Komplexitätstheorie

(© David Applegate, Robert Bixby, Vasek Chvatal und William Cook)



Rundreise durch 13.509 USA-Siedlungen mit mind. 500 Einwohnern (Stand: 1998)

Komplexitätstheorie

Notizen

- Obere Schranke: Analyse "guter" Lösungsalgorithmus
- Untere Schranke: Problemanalyse & Reduktionen

Beobachtung

- Effizientere Algorithmen für Handelsreisenden-Problem bekannt
- Problem bleibt "schwierig"

Komplexitätstheorie

Kürzester Weg

- Geg. Orte $s, z \in \{1, \ldots, n\}$, Distanzmatrix $D \in \mathbb{N}^{n \times n}$, Länge $\ell \in \mathbb{N}$
- Existiert Pfad π von s nach z mit Länge $D(\pi) \leq \ell$

- Entscheidbar per Berechnung kürzester Pfad von s nach z
- Effizient und selbst für sehr große n lösbar

Notizen

- Entscheidbarkeit unterscheidet beide Probleme nicht
- Unterscheidung effizient lösbar & schwierig (aber lösbar) gesucht

Polynomielle Berechenbarkeit

§11.4 Definition (Notation für obere Schranken; *big-O notation*)

Gegeben Funktion $f\colon \mathbb{N} \to \mathbb{N}$

$$\mathcal{O}(f) = \{g\colon \mathbb{N} \to \mathbb{N} \mid \exists x_0, a, b \in \mathbb{N}, \forall x \geq x_0 \colon g(x) \leq a \cdot f(x) + b\}$$

Notizen

- $4x^2 + 3x + 3 \in \mathcal{O}(x^2)$
- $x \cdot \log x \in \mathcal{O}(x^2)$
- $\sqrt{x} \in \mathcal{O}(x)$

Polynomielle Berechenbarkeit

§11.5 Definition (polyn. berechenbar; *polynomially computable*)

(Totale) Funktion $g\colon \Sigma^* \to \Gamma^*$ **polynomiell berechenbar**
falls det. TM M und Polynom P existieren mit

- $T(M) = g$ und
- M hält auf Eingabe $w \in \Sigma^*$ nach höchstens $P(|w|)$ Schritten

Notizen

- Polynom sichert (weitreichende) Modellunabhängigkeit
- Polynom separiert exponentielles (und schlimmeres) Verhalten
- Polynomiell berechenbar impliziert berechenbar & total

Polynomielle Berechenbarkeit

§11.6 Definition (polyn. entscheidbar; *polynomially decidable*)

Problem $L \subseteq \Sigma^*$ **polynomiell entscheidbar**
falls charakteristische Funktion χ_L polynomiell berechenbar

$$\chi_L\colon \Sigma^* \to \{0,1\} \qquad \text{mit} \qquad \chi_L(w) = \begin{cases} 1 & \text{falls } w \in L \\ 0 & \text{sonst} \end{cases}$$

Notizen

- Gleiche Begriffe für While-Programme, μ-Rekursion, etc.
  (polynomielle Transformationen)
- Berühmte Komplexitätsklasse **P**

$$\mathbf{P} = \{L \mid L \text{ polynomiell entscheidbar}\}$$

Nichtdeterminismus

Ausnahme

- Transformation TM in det. TM exponentiell
- Anderer Begriff polynomieller Entscheidbarkeit für nichtdet. TM
- Definition über Zertifikatverifikation
  (Alternative im Schöning-Buch)

Nichtdeterminismus

§11.7 Definition *(nondeterministically polynomially decidable)*

Problem $L \subseteq \Sigma^*$ **nichtdeterministisch polynomiell entscheidbar** falls Alphabet Γ, Relation $R \subseteq \Sigma^* \times \Gamma^*$ und $k \in \mathbb{N}$ existieren mit

- $\{w\#z \mid (w, z) \in R\} \in \mathrm{P}$ polynomiell entscheidbar und
- $w \in L$ gdw. $z \in \Gamma^*$ existiert mit $(w, z) \in R$ und $|z| \leq |w|^k$ für jedes $w \in \Sigma^*$

Notizen

- Polynomiell entscheidbare Zertifikatrelation R
- Zertifikate polynomieller Länge
- 2 berühmte Klassen

$$\mathbf{P} = \{L \mid L \text{ polynomiell entscheidbar}\}$$
$$\mathbf{NP} = \{L \mid L \text{ nichtdeterministisch polynomiell entscheidbar}\}$$

Nichtdeterminismus

Rucksack-Problem

- Geg. $n_1, \ldots, n_k \in \mathbb{N}$ und $n \in \mathbb{N}$ in Binärkodierung (Gegenstandsgrößen & Rucksackgröße)
- Existiert $I \subseteq \{1, \ldots, k\}$ mit $\sum_{i \in I} n_i = n$? (Kann Rucksack vollständig gefüllt werden?)
- Polynomielle Entscheidbarkeit unklar
- Nichtdet. polynomielle Entscheidbarkeit ja, in **NP**
  - Zertifikatrelation mit $\Gamma = \{0, 1\}$

$$R = \Big\{ \big(\mathrm{bin}(n_1)\# \cdots \# \mathrm{bin}(n_k)\# \mathrm{bin}(n), i_1 \cdots i_k\big) \;\Big|\; \sum_{\substack{\ell=1 \\ i_\ell \neq 0}}^{k} n_\ell = n \Big\}$$

  - R polynomiell entscheidbar (While-Programm überprüft Summe)
  - Instanz w lösbar gdw. $\exists z \in \Gamma^k$ mit $(w, z) \in R$

Determinismus vs. Nichtdeterminismus

§11.8 Theorem

$\mathbf{P} \subseteq \mathbf{NP}$

Beweis

Sei $L \in \mathbf{P}$. Wähle $\Gamma = \Sigma$, $R = \{(w, w) \mid w \in L\}$ und $k = 1$

$$w \in L \iff (w, w) \in R$$
$$\iff \exists z\colon (w, z) \in R \text{ und } |z| \leq |w|$$

$\{w\#w \mid w \in L\}$ polynomiell entscheidbar, da $L \in \mathbf{P}$ □

Determinismus vs. Nichtdeterminismus

Notizen

- Nichtdet. TM rät & überprüft “kurzen” Lösungsnachweis (Zertifikat)
- Nichtdet. TM benötigt keine Suche
  Det. TM benötigt aktuell Suche nach solchen Zertifikaten

- Polynomiell berechenbar ≈ effizient berechenbar
- Nichtdeterminismus vermutlich nicht effizient simulierbar

Polynomielle Berechenbarkeit

Wortproblem kontextsensitiver Sprache

- Ist geg. $w \in \Sigma^*$ in Sprache $L(G)$ kontextsensitiver Grammatik $G = (N, \Sigma, S, P)$?
- Problem $L(G)$

- Entscheidbarkeit von $L(G)$ entscheidbar
- Polynomielle Entscheidbarkeit von $L(G)$ unklar
- Nichtdet. polynomielle Entscheidbarkeit von $L(G)$ unklar

Polynomielle Berechenbarkeit

Sei $G = (N, \Sigma, S, P)$ kontextsensitive Grammatik

1. Setze $\mathcal{F} = \{S\}$ (nur Startsymbol)
2. Setze $\mathcal{F}' = \mathcal{F} \cup \{v \in (N \cup \Sigma)^{\leq |w|} \mid \exists u \in \mathcal{F} \colon u \Rightarrow_G v\}$ (füge Nachfolger der Länge höchstens $|w|$ hinzu)
3. Falls $\mathcal{F} \subsetneq \mathcal{F}'$, dann setze $\mathcal{F} = \mathcal{F}'$ und gehe zu 2.
4. Liefere Wahrheitswert von $w \in \mathcal{F}'$

Komplexität

- Potentiell $\sum_{i=0}^{|w|} (|N| + |\Sigma|)^i$ Elemente; exponentiell
- Ableitung als Zertifikat potentiell zu lang

Polynomielle Berechenbarkeit

Wortproblem kontextfreier Sprache

- Ist geg. $w \in \Sigma^*$ in Sprache $L(G)$ kontextfreier Grammatik $G = (N, \Sigma, S, P)$?
- Problem $L(G)$

- Entscheidbarkeit von $L(G)$ entscheidbar
- Polynomielle Entscheidbarkeit von $L(G)$ in **P**
- CYK-Algorithmus $\mathcal{O}(|w|^3)$

Polynomielle Berechenbarkeit

Problem des Handelsreisenden

- Hat geg. Distanzmatrix $D \in \mathbb{N}^{n \times n}$ Rundreise der Länge höchstens k?
- Problem $\mathrm{TSP} = \{\langle D, k\rangle \mid D \text{ hat Rundreise der Länge höchstens } k\}$

- Entscheidbarkeit von TSP entscheidbar
- Polynomielle Entscheidbarkeit von TSP unklar
- Nichtdet. polynomielle Entscheidbarkeit von TSP ja, in NP
- Zertifikat ist Rundreise der Länge höchstens k

Polynomielle Problemreduktion

§12.1 Definition (polynomielle Reduktion; *polynomial reduction*)

Problem $L \subseteq \Sigma^*$ **polynomiell reduzierbar** auf $L' \subseteq \Gamma^*$, kurz $L \preceq_{\mathbf{P}} L'$, falls polyn. ber. totale Funktion $f \colon \Sigma^* \to \Gamma^*$ mit $L = f^{-1}(L')$ existiert

Konsequenzen

- Seien $L \subseteq \Sigma^*$ und $L' \subseteq \Gamma^*$ mit $L \preceq_{\mathbf{P}} L'$
- L polynomiell entscheidbar falls L' polynomiell entscheidbar
  ($L \in \mathbf{P}$ falls $L' \in \mathbf{P}$)
- L nichtdet. polyn. entscheidbar
  falls L' nichtdet. polyn. entscheidbar
  ($L \in \mathbf{NP}$ falls $L' \in \mathbf{NP}$)

Polynomielle Problemreduktion

Problem

- Keine untere Schranke per Reduktion
- Wie erhalten wir untere Schranken?

Stephen Arthur Cook (* 1939)

- Amer.-kan. Mathematiker & Informatiker
- Polynomielle Reduktion & **NP**-Vollständigkeit
- Turing-Preisträger

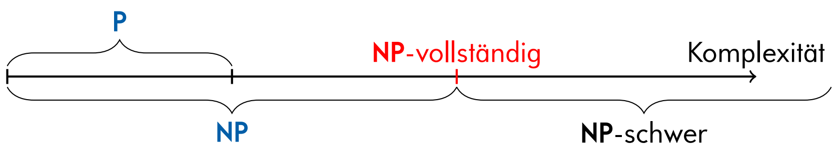
© Jiří Janíček

NP-Schwere & NP-Vollständigkeit

§12.2 Definition (NP-schwer, -vollständig; *NP-hard, -complete*)

Problem L

- **NP**-schwer falls $L' \preceq_{\mathbf{P}} L$ für alle $L' \in \mathbf{NP}$
- **NP**-vollständig falls L **NP**-schwer und $L \in \mathbf{NP}$



Notizen

- **NP**-schwer = mind. so schwer wie alle Probleme in **NP**
  (untere Schranke)
- **NP**-vollständig = passende untere & obere Schranke **NP**

NP-Schwere & NP-Vollständigkeit

§12.3 Theorem

Sei L **NP**-vollständig. Dann $L \in \mathbf{P}$ gdw. $\mathbf{P} = \mathbf{NP}$

Beweis

Falls $\mathbf{P} = \mathbf{NP}$, dann $L \in \mathbf{P} = \mathbf{NP}$, da $L \in \mathbf{NP}$ (da **NP**-vollständig).

Umgekehrt sei $L \in \mathbf{P}$ und $L' \in \mathbf{NP}$ beliebig. Da L **NP**-vollständig und damit **NP**-schwer, gilt $L' \preceq_{\mathbf{P}} L$. Zusammen mit $L \in \mathbf{P}$ folgt $L' \in \mathbf{P}$ und damit $\mathbf{NP} \subseteq \mathbf{P}$. Per Theorem §11.8 $\mathbf{P} \subseteq \mathbf{NP}$ und damit $\mathbf{P} = \mathbf{NP}$. $\square$

NP-Schwere & NP-Vollständigkeit

Notizen

- Nachweis **NP**-Schwere schwierig
  (polynomielle Reduktion von jedem Problem aus **NP**)
- Mitgliedschaft in **NP** per nichtdet. polynomielle Entscheidbarkeit
  (Angabe geeigneter Zertifikatrelation)

NP-Schwere & NP-Vollständigkeit

§12.4 Theorem

Problem L **NP**-schwer, falls **NP**-schweres Problem L' mit $L' \preceq_p L$ existiert

Beweis

Sei L' **NP**-schwer und $L' \preceq_p L$. Dann $L'' \preceq_p L' \preceq_p L$ für alle $L'' \in$ **NP**.
Transitivität $\preceq_p$ liefert $L'' \preceq_p L$ für alle $L'' \in$ **NP**, womit L **NP**-schwer $\square$

Schwierigkeit

- Bisher kein **NP**-schweres Problem

NP-Vollständigkeit

Erfüllbarkeit Aussagenlogik

- Geg. aussagenlogische Formel F
- F erfüllbar? (d.h. existiert Modell der Formel?)
- Problem $\mathrm{SAT} = \{F \mid F \text{ erfüllbare Formel Aussagenlogik}\}$

- Entscheidbarkeit SAT entscheidbar
- Polynomielle Entscheidbarkeit SAT unklar
- Nichtdet. polynomielle Entscheidbarkeit SAT ja, in **NP**

NP-Vollständigkeit

Beispiele

- Erfüllbare Formel

  $F_1 = (x_2 \vee (x_1 \wedge \neg x_3) \vee x_4) \wedge x_1 \qquad F_1^I = (1 \vee (1 \wedge \neg 0) \vee 0) \wedge 1 = 1$

  Modell $I = \{x_1, x_2\}$, kurz 1100 $\qquad x_1 = 1;\ x_2 = 1;\ x_3 = 0;\ x_4 = 0$
- Unerfüllbare Formel

  $$F_2 = ((\neg x_1 \wedge \neg x_3) \vee x_2) \wedge x_1 \wedge \neg x_2$$
- Erfüllbare Formel

  $F_3 = (\neg x_1 \vee x_3 \vee x_2) \wedge x_1 \wedge \neg x_2 \qquad F_3^I = (\neg 1 \vee 1 \vee 0) \wedge 1 \wedge \neg 0 = 1$

  Modell $I = \{x_1, x_3\}$, kurz: 101 $\qquad x_1 = 1;\ x_2 = 0;\ x_3 = 1$

NP-Vollständigkeit

§12.5 Theorem

SAT $\in$ **NP**

Beweis

Sei F aussagenlogische Formel mit k Atomen $\{x_1, \dots, x_k\}$. Wir setzen $R = \{(F, I) \mid I \models F\}$; Zertifikat ist Modell. Repräsentation R polynomiell entscheidbar, denn While-Programm kann F dekodieren, Atome gemäß Interpretation I auswerten und Wahrheitswert von $I \models F$ bestimmen. Zertifikat hat Länge $k \leq |F|$ (Interpretation = Teilmenge repräsentiert als $\{0,1\}^k$) und $F \in$ SAT gdw. $(F, I) \in R$ für $I \subseteq \{x_1, \dots, x_k\}$, wobei $(F, I) \in R$ gdw. $I \models F$. □

NP-Vollständigkeit

§12.6 Theorem

Existiert Formel U polynomieller Größe in k mit Atomen $x_1, \dots, x_k$ und $|I| = 1$ für jedes Modell $I \models U$ mit $I \subseteq \{x_1, \dots, x_k\}$

Beweis

Sei

$$U = \Big(\bigvee_{i=1}^{k} x_i\Big) \wedge \Big(\bigwedge_{1 \leq m < \ell \leq k} \neg(x_m \wedge x_\ell)\Big)$$

Formel hat Größe in $\mathcal{O}(k^2)$ und Teil $\bigvee_{i=1}^{k} x_i$ erzwingt mind. 1 Atom in I. Verbleibender Teil genau dann falsch, wenn $|I| \geq 2$. Also $|I| = 1$ für alle Modelle $I \models F$ □

NP-Vollständigkeit

§12.7 Theorem (Satz von Cook)

SAT **NP**-vollständig

Beweis (1/6)

SAT $\in$ **NP** bekannt; zu zeigen **NP**-Schwere

Sei $L \subseteq \Sigma^*$ Problem aus **NP**. Dann existieren Alphabet Γ, $k \geq 1$ und polynomiell entscheidbare Zertifikatrelation $R \subseteq \Sigma^* \times \Gamma^*$ mit

$$w \in L \text{ gdw. } (w, z) \in R \text{ und } |z| \leq |w|^k \text{ für ein } z \in \Gamma^*$$

Sei $M = (Q, \Sigma', \Gamma', \Delta, \square, q_0, q_+, q_-)$ det. TM, die R polynomiell berechnet und beschränkendes Polynom P. O.B.d.A.

- $Q = \{1, \dots, k'\}$ und $\Gamma' = \{1, \dots, k''\}$
- $(w, z) \in R$ impliziert $|z| = |w|^k$ (Zertifikate mit uniformer Länge)
- $t_{\max}(w) = P(1 + |w| + |w|^k)$ (ob. Schranke Laufzeit auf w)

NP-Vollständigkeit

Beweis (2/6)

Sei $w = \sigma_1 \cdots \sigma_\ell$. Konstruiere aussagenlogische Formel $F(w)$ mit folgenden Atomen für alle $0 \leq t \leq t_{\max}(w)$, $-t_{\max}(w) \leq i \leq t_{\max}(w)$, $q \in Q$ und $\gamma \in \Gamma'$

- $\mathsf{InState}_{t,q}$: Ist M bei Schritt t in Zustand q?
- $\mathsf{AtPos}_{t,i}$: Ist Kopf von M bei Schritt t an Bandposition i?
- $\mathsf{OnTape}_{t,i,\gamma}$: Steht Zeichen γ bei Schritt t an Bandposition i?

$$F(w) = A(w) \wedge I(w) \wedge T(w) \wedge E(w)$$

(wir geben offensichtliche Quantifikation nicht an)

<u>Endbedingung</u> $E(w) = \bigvee_t \mathsf{InState}_{t,q_+}$ (akzeptierender Zustand erreicht)

NP-Vollständigkeit

$F(w) = A(w) \land I(w) \land T(w) \land E(w)$
$Q = \{1, \ldots, k'\}$ und $\Gamma' = \{1, \ldots, k''\}$

Beweis (3/6)

Randbedingungen

$$A(w) = \bigwedge_t U(\mathrm{InState}_{t,1}, \ldots, \mathrm{InState}_{t,k'}) \land \bigwedge_t U(\mathrm{AtPos}_{t,-t_{\max}(w)}, \ldots, \mathrm{AtPos}_{t,t_{\max}(w)}) \land \bigwedge_{t,i} U(\mathrm{OnTape}_{t,i,1}, \ldots, \mathrm{OnTape}_{t,i,k''})$$

- Zu jedem Schritt in genau 1 Zustand
- Zu jedem Schritt an genau 1 Position
- Zu jedem Schritt steht genau 1 Zeichen an Position

NP-Vollständigkeit

$F(w) = A(w) \land I(w) \land T(w) \land E(w)$

Beweis (4/6)

Initialbedingungen

$$I(w) = \mathrm{InState}_{0,q_0} \land \mathrm{AtPos}_{0,0} \land \Big(\bigwedge_{m \notin \{0,\ldots,\ell^k+\ell\}} \mathrm{OnTape}_{0,m,\square} \Big) \land \Big(\bigwedge_{m=1}^{\ell} \mathrm{OnTape}_{0,m-1,\sigma_m} \Big) \land \mathrm{OnTape}_{0,\ell,\#} \land \Big(\bigwedge_{m=\ell+1}^{\ell^k+\ell} \neg \mathrm{OnTape}_{0,m,\square} \Big)$$

- Initial im Zustand q_0 und an Position 0
- Außerhalb Eingabe steht $\square$ auf Band
- Auf Band steht $w\#z$ für beliebiges $z \in \Gamma^{\ell^k}$

NP-Vollständigkeit

$F(w) = A(w) \land I(w) \land T(w) \land E(w)$

Beweis (5/6)

Übergangsbedingungen mit $\diamond = 0$, $\triangleleft = -1$ und $\triangleright = 1$

$$T(w) = \bigwedge_{\substack{t,i,\gamma \\ t \neq t_{\max}(w)}} \Big((\neg \mathrm{AtPos}_{t,i} \land \mathrm{OnTape}_{t,i,\gamma}) \to \mathrm{OnTape}_{t+1,i,\gamma} \Big) \land \bigwedge_{\substack{t,q,i,\gamma \\ t \neq t_{\max}(w), q \notin \{q_+,q_-\}}} \Big((\mathrm{InState}_{t,q} \land \mathrm{AtPos}_{t,i} \land \mathrm{OnTape}_{t,i,\gamma}) \to \bigvee_{((q,\gamma)\to(q',\gamma',d)) \in \Delta'} (\mathrm{InState}_{t+1,q'} \land \mathrm{AtPos}_{t+1,i+d} \land \mathrm{OnTape}_{t+1,i,\gamma'}) \Big)$$

- Band außerhalb aktueller Position erhalten
- Prüfe Vorbedingungen Übergang
- Für jeden Schritt führe passenden Übergang aus

NP-Vollständigkeit

$F(w) = A(w) \land I(w) \land T(w) \land E(w)$

Beweis (6/6)

Teilformeln polynomieller Länge in $|w|$ und $F(w)$ polynomiell berechenbar. Sei $w \in L$ mit Zertifikat $z \in \Gamma^{\ell^k}$. Dann $F(w)$ für Band $w\#z$ erfüllbar, da Berechnung von M simuliert und M akzeptiert. Umgekehrt sei $F(w)$ erfüllbar. Dann liefert Modell Zertifikat z und akzeptierende Berechnung det. TM M. Folglich $w \in L$. Also $w \in L$ gdw. $F(w)$ erfüllbar gdw. $F(w) \in \mathrm{SAT}$. Damit $L \preceq_P \mathrm{SAT}$, womit SAT **NP**-schwer und **NP**-vollständig. $\square$

NP-vollständige Probleme

§12.8 Definition (konjunktive Normalform mit 3 Literalen)

Aussagenlogische Formel F in **konjunktiver Normalform mit 3 Literalen** (3KNF) falls $F = F_1 \wedge \cdots \wedge F_k$ für Formeln $F_1, \dots, F_k$ mit $F_i = L_{i1} \vee L_{i2} \vee L_{i3}$ für alle $1 \leq i \leq k$ und Literale L_{i1}, L_{i2}, L_{i3}
(Literal = Atom oder negiertes Atom)

Beispiele

- $x_1 \vee x_2 \vee x_3$ in 3KNF
- $(x_1 \wedge x_2) \vee x_4$ nicht in 3KNF
- $x_1 \wedge (x_2 \vee x_1 \vee \neg x_3)$ nicht in 3KNF
- $(x_1 \vee x_1 \vee x_1) \wedge (x_2 \vee \neg x_1 \vee \neg x_3)$ in 3KNF
  (erlauben auch ≤ 3 Literale; dann 3. Beispiel in 3KNF)

NP-vollständige Probleme

Erfüllbarkeit 3KNF-Formel

- Geg. aussagenlogische Formel F in 3KNF
- Ist F erfüllbar? (Existiert Modell?)
- Problem 3-SAT $= \{F \mid F$ erfüllbare Formel in 3KNF$\}$

- Entscheidbarkeit 3-SAT entscheidbar
- Polynomielle Entscheidbarkeit 3-SAT unklar
- Nichtdet. polynomielle Entscheidbarkeit 3-SAT ja, in **NP**
  (denn 3-SAT $\preceq_{\mathrm{P}}$ SAT vermittels Identität; also 3-SAT $\in$ **NP**)

NP-vollständige Probleme

§12.9 Theorem

3-SAT **NP**-vollständig

Beweis (1/2)

Da 3-SAT $\in$ **NP** nur **NP**-Schwere per SAT $\preceq_{\mathrm{P}}$ 3-SAT zu zeigen. Sei F aussagenlogische Formel. Transformiere F in Polynomialzeit in Negationsnormalform (Negationen nur vor Atomen). Sei T Syntaxbaum der erhaltenen Formel F' und für jeden Knoten w dieses Baumes sei

- $v(w) = T(w)$ falls Knotenbeschriftung $T(w)$ Literal
- sonst $v(w) = y$ für neues Atom y.

Konstruiere Formel $f(F) = v(\varepsilon) \wedge \bigwedge_{w \text{ innere Position in } T} F'_w$, in der für jedes w Formel F'_w durch Formel $v(w) \leftrightarrow (v(w_1)\ T(w)\ v(w_2))$ in 3KNF gegeben ist, wobei $T(w)$ Symbol ($\vee$ oder $\wedge$) an Position w und w_1/w_2 erste/zweite Kindposition von w (Tseitin-Transformation)

NP-vollständige Probleme

Beweis (2/2)

3KNF Teilformel F'_w gegeben durch

$$(L_1 \leftrightarrow (L_2 \vee L_3)) \quad \text{äq. zu} \quad (L_1 \vee \neg L_2) \wedge (\neg L_1 \vee L_2 \vee L_3) \wedge (L_1 \vee \neg L_3)$$

$$(L_1 \leftrightarrow (L_2 \wedge L_3)) \quad \text{äq. zu} \quad (\neg L_1 \vee L_2) \wedge (L_1 \vee \neg L_2 \vee \neg L_3) \wedge (\neg L_1 \vee L_3)$$

Offenbar $f(F)$ in 3KNF und

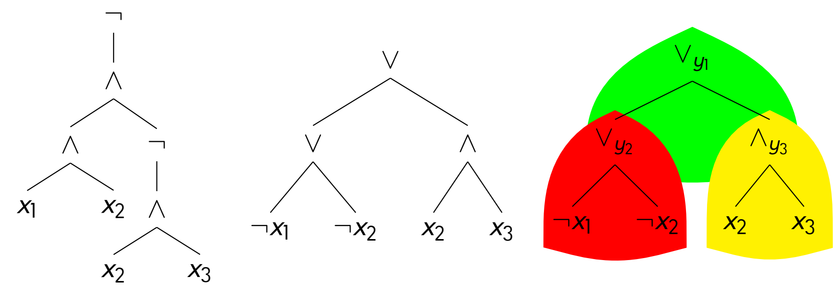
$$F \text{ erfüllbar gdw. } f(F) \text{ erfüllbar}$$

Damit SAT $\preceq_{\mathrm{P}}$ 3-SAT, womit 3-SAT **NP**-vollständig. $\square$

NP-vollständige Probleme

Beispiel $F = \neg((x_1 \wedge x_2) \wedge \neg(x_2 \wedge x_3))$

Syntaxbaum von F — Syntaxbaum NNF F' — Zuweisung



Konstruierte Formel $f(F)$

$$y_1 \wedge (y_1 \leftrightarrow (y_2 \vee y_3)) \wedge (y_2 \leftrightarrow (\neg x_1 \vee \neg x_2)) \wedge (y_3 \leftrightarrow (x_2 \wedge x_3))$$

NP-vollständige Probleme

Konjunktive Normalform mit 2 Literalen

- Geg. aussagenlogische Formel F in 2KNF (max. 2 Literale)
- Ist F erfüllbar? (Existiert Modell?)
- Problem 2-SAT $= \{F \mid F \text{ erfüllbare Formel in 2KNF}\}$

- Entscheidbarkeit 2-SAT entscheidbar
- Polynomielle Entscheidbarkeit 2-SAT ja, in **P**
- Nichtdet. polynomielle Entscheidbarkeit 2-SAT ja, in **NP**

NP-vollständige Probleme

Algorithmus für polynomielle Entscheidbarkeit

- Resolution für Unerfüllbarkeit (Resolution korrekt & vollständig)
- Resolventen haben höchstens 2 Literale
- Höchstens $(2|F|)^2$ Resolventen bildbar