Vorlesung 12
LOOP-Programme

Wdh.: Turing-mächtige Programmiersprachen

Definition

Eine Programmiersprache wird als Turing-mächtig bezeichnet, wenn jede Funktion, die durch eine TM berechnet werden kann, auch durch ein Programm in dieser Programmiersprache berechnet werden kann.

Satz

Die Programmiersprache WHILE ist Turing-mächtig.

Wdh.: Beispiel eines WHILE-Programms

Was berechnet dieses WHILE-Programm?

WHILE $x_2 \neq 0$ DO
$x_1 := x_1 + 1;$
$x_2 := x_2 - 1$
END;
$x_0 := x_1$

Die Programmiersprache LOOP – Syntax

Elemente eines LOOP-Programms

- Variablen $x_0 \quad x_1 \quad x_2 \ldots$
- Konstanten $-1 \quad 0 \quad 1$
- Symbole $; \quad := \quad + \quad \neq$
- Schlüsselwörter LOOP DO END

Die Programmiersprache LOOP – Syntax

Induktive Definition – Induktionsanfang

Zuweisung

Für jedes $c \in \{-1, 0, 1\}$ ist die Zuweisung

$$x_i \ := \ x_j + c$$

ein LOOP-Programm.

Statt $x_i := x_j + 0$ schreiben wir einfach $x_i := x_j$.

Die Programmiersprache LOOP – Syntax

Induktive Definition – Induktionsschritte:

Hintereinanderausführung

Falls P_1 und P_2 LOOP-Programme sind, dann ist auch

$$P_1; P_2$$

ein LOOP-Programm.

LOOP-Konstrukt

Falls P ein LOOP-Programm ist, dann ist auch

$$\text{LOOP } x_i \text{ DO } P \text{ END}$$

ein LOOP-Programm, wobei x_i nicht in P vorkommen darf.

Die Programmiersprache LOOP – Semantik

Ein LOOP-Programm P berechnet eine k-stellige Funktion der Form $f : \mathbb{N}^k \to \mathbb{N}$.

- Die Eingabe ist in den Variablen $x_1, \ldots, x_k$ enthalten.
- Alle anderen Variablen werden mit 0 initialisiert.
- Das Resultat eines LOOP-Programms ist die Zahl, die sich am Ende der Rechnung in der Variable x_0 ergibt.

- Programme der Form $x_i := x_j + c$ sind Zuweisungen des Wertes $x_j + c$ an die Variable x_i.
- In einem LOOP-Programm $P_1; P_2$ wird zunächst P_1 und dann P_2 ausgeführt.
- Das Programm LOOP x_i DO P END hat folgende Bedeutung: P wird x_i mal hintereinander ausgeführt.

LOOP-Programme – Beispiele

Beispiel 1

$x_0 := x_1 + 0;$
LOOP x_2 DO $x_0 := x_0 + 1$ END

Diese Programm berechnet die Addition $x_1 + x_2$.

Beispiel 2

LOOP x_1 DO
    LOOP x_2 DO $x_0 := x_0 + 1$ END
END

Diese Programm berechnet die Multiplikation $x_1 \cdot x_2$.

LOOP-Programme – Beispiele

Beispiel 3

Seien P_1 und P_2 LOOP-Programme, in denen x_1, x_2 und x_3 nicht vorkommen.

$$
\begin{aligned}
&x_2 := x_2 + 1;\\
&\text{LOOP } x_1 \text{ DO } x_2 := x_3;\ x_3 := x_3 + 1 \text{ END};\\
&\text{LOOP } x_2 \text{ DO } P_1 \text{ END};\\
&\text{LOOP } x_3 \text{ DO } P_2 \text{ END}
\end{aligned}
$$

Dieses Programm entspricht:
IF $x_1 = 0$ THEN P_1 ELSE P_2 END

LOOP-berechenbare Funktionen

Beobachtung

Alle LOOP Programme halten bei jeder Eingabe an, weil sie keine Endlosschleifen enthalten können.

LOOP Programme berechnen also immer totale Funktionen.

Die durch LOOP-Programme berechenbaren (totalen) Funktionen bezeichnen wir als LOOP-berechenbar.

Anmerkung

LOOP-berechenbare Funktionen entsprechen genau den sogenannten primitiv rekursiven Funktionen. Das sind (vereinfacht ausgedrückt) Funktionen, deren Wert an der Stelle n rekursiv aus den Werten an Stellen $k < n$ definiert wird.

Die Turmfunktion

Die Turmfunktion $T : \mathbb{N} \to \mathbb{N}$ ist rekursive definiert durch

$$T(0) := 1, \qquad T(n+1) := 2^{T(n)}.$$

Also

$$T(n) = 2^{2^{\cdot^{\cdot^{\cdot^{2}}}}} \Big\} n \text{ mal}$$

Einige Werte:

n	0	1	2	3	4	5
$T(n)$	1	2	4	16	65536	2^{65536}

Lemma

Die Turmfunktion ist LOOP-berechenbar.

Ein Programm für die Turmfunktion

- Ein Programm P_1 für die Funktion $f_1(x) = 2x$:

  LOOP x_1 DO $x_0 := x_0 + 1; x_0 := x_0 + 1$ END

- Ein Programm P_2 für die Funktion $f_2(x) = 2^x$:

  $x_0 := x_0 + 1;$
  LOOP x_1 DO
  $\quad x_0 := 2x_0$
  END
  LOOP x_1 DO
  $\quad x_2 := x_3;$
  $\quad$ LOOP x_0 DO $x_2 := x_2 + 1; x_2 := x_2 + 1$ END;
  $\quad x_0 := x_2$
  END

- Ein Programm P_3 für die Turmfunktion:

  $x_0 := x_0 + 1$
  LOOP x_1 DO
  $\quad x_0 := 2^{x_0}$
  END

Laufzeit von LOOP-Programmen

LOOP-Programme halten immer an, weil sie keine Endlosschleifen enthalten. Schleifen können aber trotzdem s e h r l a n g e laufen.

Beispiel

$x_2 := T(x_1)$;
LOOP x_2 DO P END

Ist LOOP Turing-mächtig?

Vermutung von Hilbert (1926):

Die Klasse der primitiv-rekursiven Funktionen (also LOOP-berechenbaren Funktionen) stimmt mit der Klasse der berechenbaren (totalen) Funktionen überein.

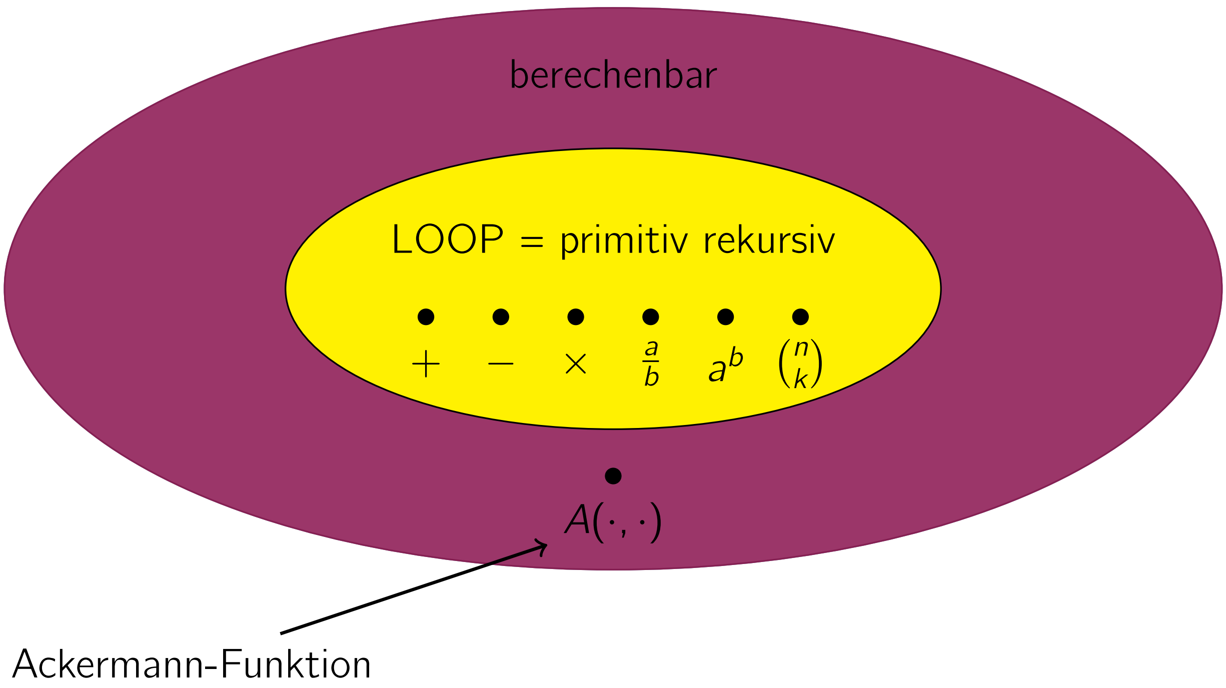
Hilbert hatte allerdings noch keinen formalen Begriff von Berechenbarkeit, sondern bezog sich auf einen intuitiven Begriff von berechenbaren (oder rekursiven) Funktionen.

Ackermann (1929)

Hilberts Vermutung ist falsch!

Ackermann gab eine Funktion an, die intuitiv klar berechenbar ist, und bewies, dass sie nicht primitiv rekursiv ist.

Berechenbare Totale Funktionen



Die Ackermann-Funktion – Definition

Definition

Die Ackermannfunktion $A\colon \mathbb{N}^2 \to \mathbb{N}$ ist folgendermaßen definert:

$$\begin{array}{lcll} A(0,n) & = & n+1 & \text{für } n \geq 0 \\ A(m+1,0) & = & A(m,1) & \text{für } m \geq 0 \\ A(m+1,n+1) & = & A(m,A(m+1,n)) & \text{für } m,n \geq 0 \end{array}$$

Einige Werte der Ackermann-Funktion

$m\backslash n$	0	1	2	3	4	5	6
0	1	2	3	4	5	6	7
1	2	3	4	5	6	7	8
2	3	5	7	9	11	13	15
3	5	13	29	61	125	253	509
4	13	65 533	$2^{65536}-3$				

Es ist klar, dass die Ackermann-Funktion berechenbar ist.

Wachstum für feste m

Lemma

Für alle $n \in \mathbb{N}$ gilt:

- $A(0, n) = n + 1$,
- $A(1, n) = n + 2$,
- $A(2, n) = 2 \cdot (n + 3) - 3$,
- $A(3, n) = 2^{n+3} - 3$,
- $A(4, n) = T(n + 3) - 3$,

(Ohne Beweis)

Bereits $A(4, 2) = 2^{65536} - 3$ ist größer als die (vermutete) Anzahl der Atome im Weltraum.

Monotonieeigenschaften der Ackermannfunktion

Strikte Monotonie in der zweiten Komponente

$A(m, n + 1) > A(m, n)$ für alle $m, n \in \mathbb{N}$ (Übungsaufgabe)

Diagonale Monotonie

$A(m + 1, n) \geq A(m, n + 1)$ für alle $m, n \in \mathbb{N}$

Beweis der diagonalen Monotonie

Wir beweisen zunächst $A(m,n) \geq n+1$ für alle $m, n \in \mathbb{N}$ per Induktion über m.

Induktionsanfang: $m = 0$

$$A(0,n) = n+1.$$

Induktionsschritt: $m \to m+1$

Wir beweisen $A(m+1,n) \geq n+1$ per Induktion über n.

Induktionsanfang: $n = 0$

$$A(m+1,0) = A(m,1) \geq 2$$

nach IA(m) (Induktionsannahme der Induktion über m).

Induktionsschritt: $n \to n+1$

$$\begin{aligned} A(m+1,n+1) &= A(m, A(m+1,n)) && \text{Definition } A \\ &\geq A(m+1,n)+1 && \mathrm{IA}(m) \\ &\geq n+1+1 && \mathrm{IA}(n) \\ &= n+2. \end{aligned}$$

Beweis der diagonalen Monotonie (Forts.)

Wir beweisen $A(m+1,n) \geq A(m,n+1)$ per Induktion über n.

Induktionsanfang: $n = 0$

$A(m+1,0) = A(m,1)$ gilt nach Definition von A.

Induktionsschritt: $n \to n+1$

Nach der Induktionsannahme gilt $A(m+1,n) \geq A(m,n+1)$.

Wir haben bereits $A(m,n+1) \geq n+2$ bewiesen.

Also gilt $A(m+1,n) \geq n+2$ und damit

$$\begin{aligned} A(m+1,n+1) &= A(m, A(m+1,n)) && \text{Def. } A \\ &\geq A(m,n+2) && \text{Mon. in 2. Komp.} \end{aligned}$$

Monotonie

Strikte Monotonie in der zweiten Komponente

$A(m, n+1) > A(m, n)$ für alle $m, n \in \mathbb{N}$ (Übungsaufgabe)

Diagonale Monotonie

$A(m+1, n) \geq A(m, n+1)$ für alle $m, n \in \mathbb{N}$

Strikte Monotonie in der ersten Komponente

$A(m+1, n) > A(m, n)$ für alle $m, n \in \mathbb{N}$

Beweis:

$$A(m+1, n) \geq A(m, n+1) > A(m, n).$$

Wachstum der Variableninhalte in einem LOOP-Programm

Definition der Funktion F_P

- Sei P ein LOOP-Programm
- Seien $x_0, x_1, \ldots, x_k$ die Variablen in P.
- Wenn die Variablen initial die Werte $a = (a_0, \ldots, a_k) \in \mathbb{N}^{k+1}$ haben, dann sei $f_P(a)$ das $(k+1)$-Tupel der Variablenwerte nach Ausführung von P.
- Sei $|f_P(a)|$ die Summe der Einträge im $(k+1)$-Tupel $f_P(a)$.
- Wir definieren nun die Funktion $F_P\colon \mathbb{N} \to \mathbb{N}$ durch

$$F_P(n) = \max \left\{ |f_P(a)| \;\middle|\; a \in \mathbb{N}^{k+1} \text{ mit } \sum_{i=0}^{k} a_i \leq n \right\}.$$

Intuitiv beschreibt die Funktion F_P das maximale Wachstum der Variablenwerte im LOOP-Programm P.

Ackermannfunktion versus F_P

Wir zeigen nun, dass $F_P(n)$ für alle $n \in \mathbb{N}$ echt kleiner ist als $A(m, n)$, wenn der Parameter m genügend groß in Abhängigkeit von P gewählt wird.

Lemma

Für jedes LOOP-Programm P gibt es eine natürliche Zahl m, so dass für alle $n \in \mathbb{N}$ gilt: $F_P(n) < A(m, n)$.

Beachte: Für ein festes Programm P ist der Parameter m eine Konstante.

Beweis durch strukturelle Induktion (Überblick)

Induktionsanfang

- Sei P von der Form $x_i := x_j + c$ für $c \in \{-1, 0, 1\}$.
- Wir werden zeigen: $F_P(n) < A(2, n)$.

Induktionsschritt (1. Art)

- Sei P von der Form $P_1; P_2$.
- Induktionsannahme: $\exists q \in \mathbb{N} : F_{P_1}(\ell) < A(q, \ell)$ und $F_{P_2}(\ell) < A(q, \ell)$.
- Wir werden zeigen: $F_P(n) < A(q + 1, n)$.

Induktionsschritt (2. Art)

- Sei P von der Form LOOP x_i DO Q END.
- Induktionsannahme: $\exists q \in \mathbb{N} : F_Q(\ell) < A(q, \ell)$.
- Wir werden zeigen: $F_P(n) < A(q + 1, n)$.

Beweis des Lemmas

Induktionsanfang

- Sei P von der Form $x_i := x_j + c$ für $c \in \{-1, 0, 1\}$.
- Dann gilt $F_P(n) \leq 2n + 1$.
- Somit folgt $F_P(n) < A(2, n)$.

Erläuterung:

- Vor Ausführung von P könnte gelten $x_j = n$ und alle anderen Variablen haben den Wert 0.
- Ferner könnte c den Wert 1 haben.
- Nach Ausführung von P gilt somit $x_i = n + 1$ und somit ist die Summe der Variableninhalte $x_i + x_j = 2n + 1$.
- Ein größeres Wachstum der Variableninhalte ist nicht möglich.

Formal:

$$\sum_{t \in \{0,\dots,k\}} x'_t = x'_i + \sum_{t \in \{0,i-1,i+1,k\}} x'_t \leq x_j + 1 + \sum_{t \in \{0,i-1,i+1,k\}} x_t \leq n + 1 + n,$$

wobei x'_t der Wert der Variable x_t nach Ausführung des Programms P sei.

Beweis des Lemmas

Induktionsschritt (1. Art)

- Sei P von der Form $P_1; P_2$.
- Induktionsannahme: $\exists q \in \mathbb{N} : F_{P_1}(\ell) < A(q, \ell)$ und $F_{P_2}(\ell) < A(q, \ell)$.
- Somit gilt

$$\begin{aligned} F_P(n) &\leq F_{P_2}(F_{P_1}(n)) \\ &< A(q, A(q, n)) && \text{IA und Monotonie 2. Komp.} \\ &\leq A(q, A(q+1, n-1)) && \text{Diagonalmon. und Mon. 2. Komp.} \\ &= A(q+1, n). \end{aligned}$$

Beweis des Lemmas

Induktionsschritt (2. Art)

- Sei P von der Form LOOP x_i DO Q END.
- Induktionsannahme: $\exists q \in \mathbb{N} : F_Q(\ell) < A(q, \ell)$.
- Sei $\alpha = \alpha(n)$ derjenige Wert aus $\{0, 1, \ldots, n\}$ für x_i, der $F_P(n)$ maximiert.
- Dann gilt

$$F_P(n) \;\le\; F_Q(F_Q(\ldots F_Q(F_Q(n - \alpha))\ldots)) + \alpha,$$

wobei die Funktion $F_Q(\cdot)$ hier α-fach ineinander eingesetzt ist.

Beweis des Lemmas

Induktionsschritt (2. Art) – Fortsetzung

- Bisher haben wir gezeigt, dass

$$F_P(n) \;\le\; \underbrace{F_Q(F_Q(\ldots F_Q(F_Q(n - \alpha))\ldots))}_{\alpha\text{-fach verschachtelt}} + \alpha.$$

- Aus der Induktionsannahme folgt $F_Q(\ell) < A(q, \ell)$.
- Dies wenden wir auf die äußerste Funktion F_Q an und erhalten

$$F_P(n) \;<\; A(q, \underbrace{F_Q(\ldots F_Q(F_Q(n - \alpha))\ldots)}_{(\alpha-1)\text{-fach verschachtelt}}) + \alpha,$$

also

$$F_P(n) \;\le\; A(q, \underbrace{F_Q(\ldots F_Q(F_Q(n - \alpha))\ldots)}_{(\alpha-1)\text{-fach verschachtelt}}) + \alpha - 1,$$

Beweis des Lemmas

Induktionsschritt (2. Art) – Fortsetzung

- Bisher haben wir gezeigt, dass

$$F_P(n) \;\le\; A(q, \underbrace{F_Q(\dots F_Q(F_Q(n-\alpha))\dots)}_{(\alpha-1)\text{-fach verschachtelt}}) + \alpha - 1,$$

- Erneute Anwendung von $F_Q(\ell) < A(q,\ell)$ und Monotonie ergibt

$$F_P(n) \;<\; A(q, A(q, \underbrace{F_Q(\dots F_Q(F_Q(n-\alpha))\dots)}_{(\alpha-2)\text{-fach verschachtelt}})) + \alpha - 1,$$

also $$F_P(n) \;\le\; A(q, A(q, \underbrace{F_Q(\dots F_Q(F_Q(n-\alpha))\dots)}_{(\alpha-2)\text{-fach verschachtelt}}))) + \alpha - 2,$$

- Wir wiederholen das Argument und erhalten für $0 \le i \le \alpha$

$$F_P(n) \;\le\; \underbrace{A(q, A(q, \dots}_{i\text{-mal verschachtelt}}, \underbrace{F_Q(\dots F_Q(F_Q(n-\alpha))\dots)}_{(\alpha-i)\text{-fach verschachtelt}}))) + \alpha - i,$$

Beweis des Lemmas

Induktionsschritt (2. Art) – Fortsetzung

- Für $i = \alpha$ ergibt sich

$$F_P(n) \le \underbrace{A(q, A(q, \dots A(q, A(q, n-\alpha))\dots))}_{\alpha\text{-fach verschachtelt}}$$

Mit Hilfe der Monotonie erhält man

$$F_P(n) \le \underbrace{A(q, A(q, \dots A(q, A(q+1, n-\alpha))\dots))}_{\alpha\text{-fach verschachtelt}}$$

- Der Definition der Ackermannfunktion angewandt auf die innere Verschachtelung ergibt

$$F_P(n) \;\le\; \underbrace{A(q, A(q, \dots A(q+1, n-\alpha+1)\dots))}_{(\alpha-1)\text{-fach verschachtelt}}.$$

- Nach $\alpha - 2$ weiteren Anwendungen folgt $F_P(n) \;\le\; A(q+1, n-1)$.

Beweis des Lemmas

Induktionsschritt (2. Art) – Fortsetzung

- Bisher haben wir gezeigt:

$$F_P(n) \;\leq\; A(q+1, n-1)$$

- Die Monotonie im zweiten Argument ergibt dann

$$F_P(n) \;<\; A(q+1, n)$$

□

Nicht-LOOP-Berechenbarkeit der Ackermannfunktion

Satz

Die Ackermannfunktion ist nicht LOOP-berechenbar.

Beweis:

- Angenommen, es gibt ein LOOP-Programm, das die Ackermannfunktion berechnet.
- Dann gibt es auch ein LOOP-Programm, das die Funktion $B(n) = A(n, n)$ berechnet. Sei P dieses LOOP-Programm.
- Aus dem Lemma über LOOP-Programme folgt: es gibt $m \in \mathbb{N}$, so dass für jedes $n \in \mathbb{N}$ gilt: $F_P(n) < A(m, n)$.
- Wenn P mit Eingabe m aufgerufen wird, so berechnet P den Funktionswert $B(m)$. Somit gilt $B(m) \leq F_P(m)$.
- Es folgt

$$B(m) \;\leq\; F_P(m) \;<\; A(m, m) \stackrel{\text{Def. von } B}{=} B(m).$$

- Widerspruch! Also folgt der Satz. □

Schlussfolgerung

Da die Ackermannfunktion (durch eine TM) berechenbar ist, folgt:

Korollar

Die Klasse der LOOP-berechenbaren Funktionen ist eine echte Teilmenge der berechenbaren (totalen) Funktionen.

Zur Klärung

Technisch beschränken wir uns in diesem Kapitel auf Funktionen $f : \mathbb{N}^k \to \mathbb{N}$, $k \in \mathbb{N}$. Obige Aussage gilt auch für Funktionen der Form $f : \Sigma^* \to \Sigma^*$ über einem beliebigen endlichen Alphabet Σ.

LOOP-entscheidbare Mengen

Eine Menge $L \subseteq \mathbb{N}$ ist LOOP-entscheidbar, wenn ihre charakteristische Funktion $\chi_L : \mathbb{N} \to \{0, 1\}$, definiert durch

$$\chi_L(x) = \begin{cases} 1 & \text{fall } x \in L, \\ 0 & \text{sonst,} \end{cases}$$

LOOP-berechenbar ist.

Theorem

Es gibt eine entscheidbare Menge, die nicht LOOP-entscheidbar ist.

Der Beweis lässt sich mittels eines Diagonalisierungsarguments führen.

Zur Klärung

- Eine Menge $L \subseteq \mathbb{N}$ nennen wir entscheidbar, wenn die Menge $\{\text{bin}(n) \mid n \in L\} \subseteq \{0, 1\}^*$ entscheidbar ist.
- Umgekehrt können wir den Begriff der LOOP-Entscheidbarkeit auch auf Sprachen $L \subseteq \Sigma^*$ über beliebigen endlichen Alphabeten Σ erweitern.

Zusammenfassung – Berechenbarkeit

Wir haben die folgenden Turing-mächtigen Rechenmodelle und Programmiersprachen kennen gelernt.

- Turingmaschine (TM)
- k-Band-TM
- Registermaschine (RAM)
- eingeschränkte RAM
- WHILE-Programme (und somit C, Java, Pascal, Postscript, etc.)

LOOP-Programme sind hingegen nicht Turing-mächtig.

Zusammenfassung – Berechenbarkeit

Church-Turing-These

Die Klasse der TM-berechenbaren Funktionen stimmt mit der Klasse der „intuitiv berechenbaren“ Funktionen überein.

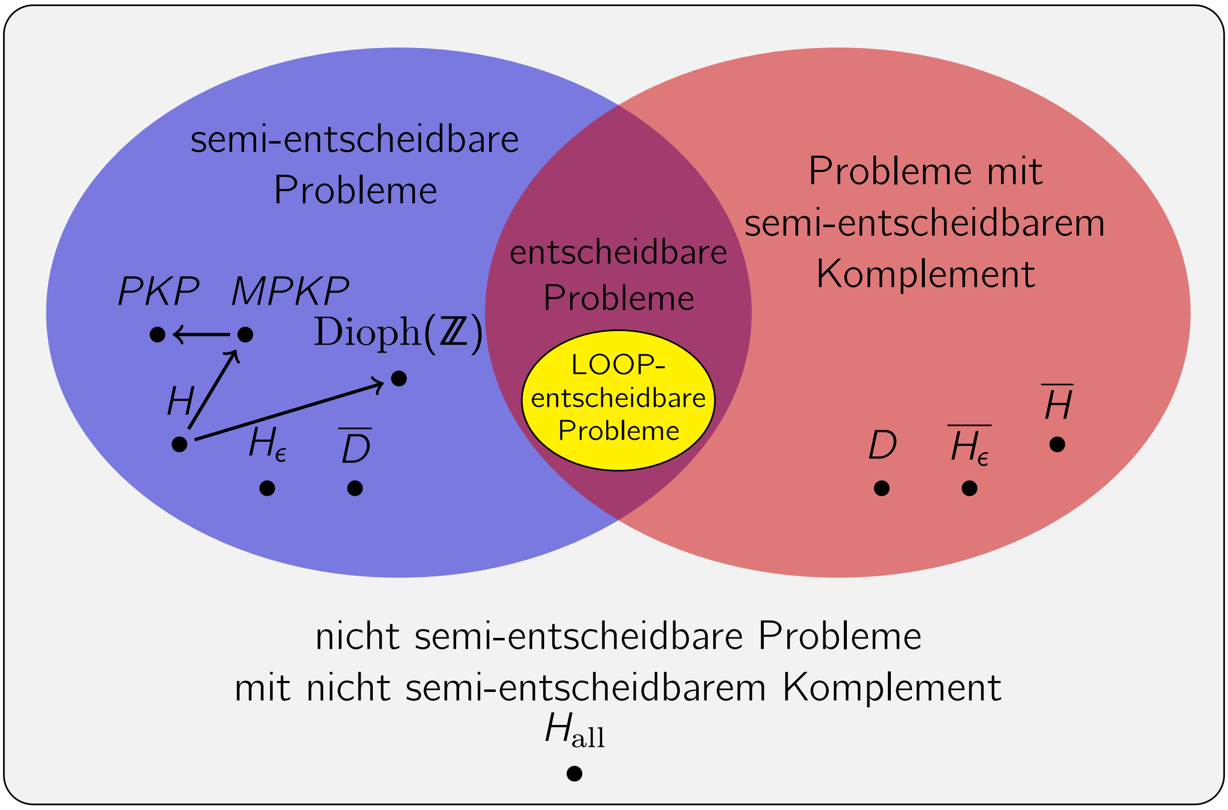
In anderen Worten:

Ein Problem kann genau dann „algorithmisch gelöst werden“, wenn es eine TM für dieses Problem gibt.

An Stelle des Begriffs „TM“ können wir auch jedes andere Turing-mächtige Rechenmodell verwenden.

Zusammenfassung – Berechenbarkeit

Berechenbarkeitslandschaft:



Zusammenfassung – Berechenbarkeit

Bedeutende nicht berechenbare Probleme:

- Halteproblem, in verschiedenen Varianten
- *Satz von Rice:* Aussagen über Eigenschaften von Funktionen, die durch eine gegebene TM berechnet werden, sind nicht entscheidbar
- *Schlussfolgerung:* Die automatische Verifikation von Programmen in einer TM-mächtigen Programmiersprachen ist nicht möglich
- Hilberts 10. Problem
- Postsches Korrespondenzproblem

Zusammenfassung – Berechenbarkeit

Methoden zum Nachweis von Nicht-Berechenbarkeit:

- Diagonalisierung
- Unterprogrammtechnik
- Satz von Rice
- Reduktionen (spezielle Variante der Unterprogrammtechnik)