VL-11: Primitiv rekursive Funktionen

(Berechenbarkeit und Komplexität, WS 2018)

Gerhard Woeginger

WS 2018, RWTH

Organisatorisches

- Nächste Vorlesung:
  Donnerstag, Dezember 13, 12:30–14:00 Uhr, Aula
- Freitag, Dezember 7 (morgen): Tag der Informatik
- Webseite:
  `http://algo.rwth-aachen.de/Lehre/WS1819/BuK.php`

Wiederholung

Wdh.: Ackermann Funktion: Definition

Definition

Die Ackermann Funktion $A : \mathbb{N}^2 \to \mathbb{N}$ ist folgendermassen definiert:

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

Wilhelm Ackermann:
"Zum Hilbertschen Aufbau der reellen Zahlen"
Mathematische Annalen 99 (1928), pp. 118–133

Rózsa Péter:
"Konstruktion nichtrekursiver Funktionen"
Mathematische Annalen 111 (1935), pp. 42–60

Wdh.: LOOP versus WHILE

Lemma

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

Satz

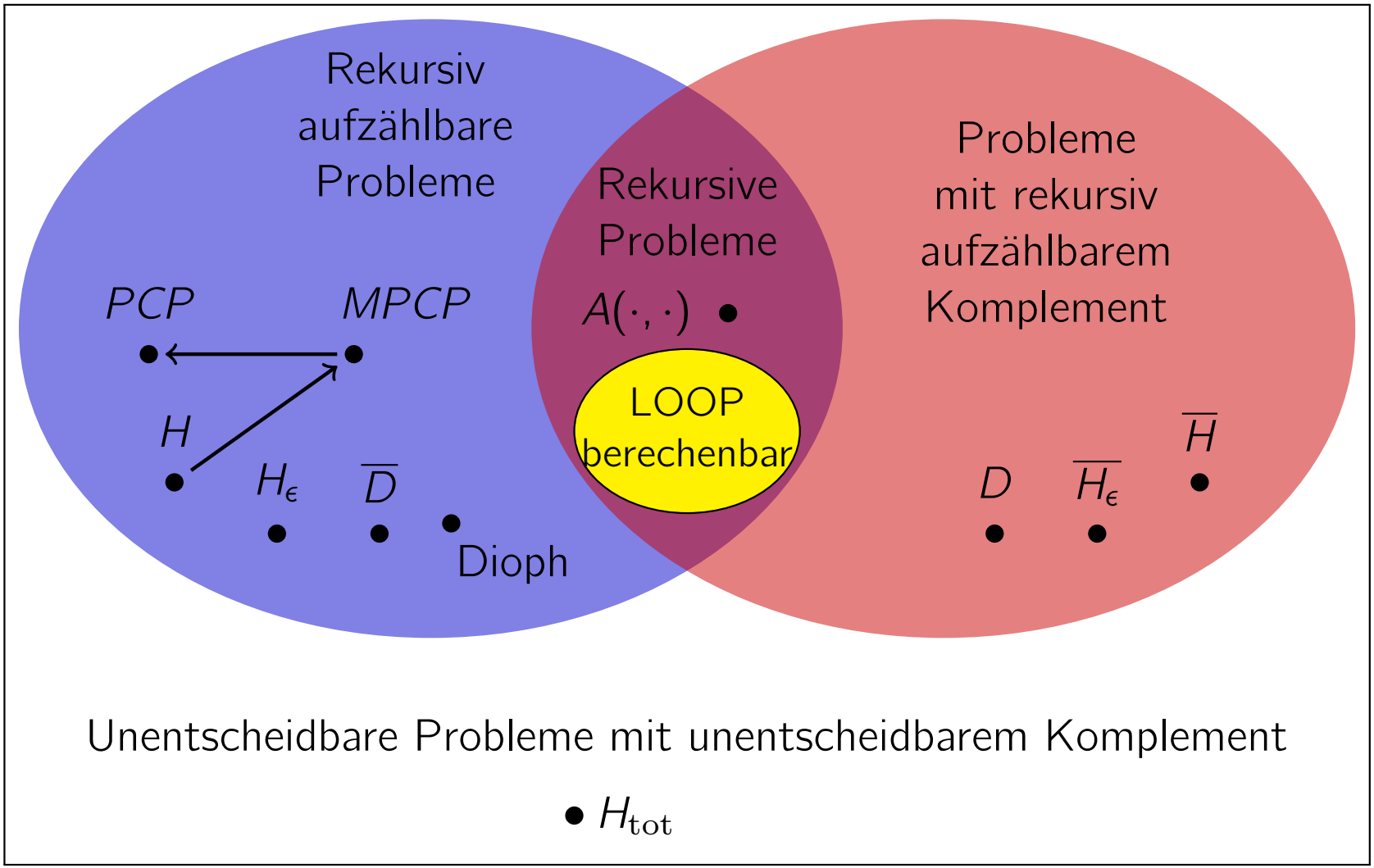
Die Ackermann Funktion ist nicht LOOP-berechenbar.

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

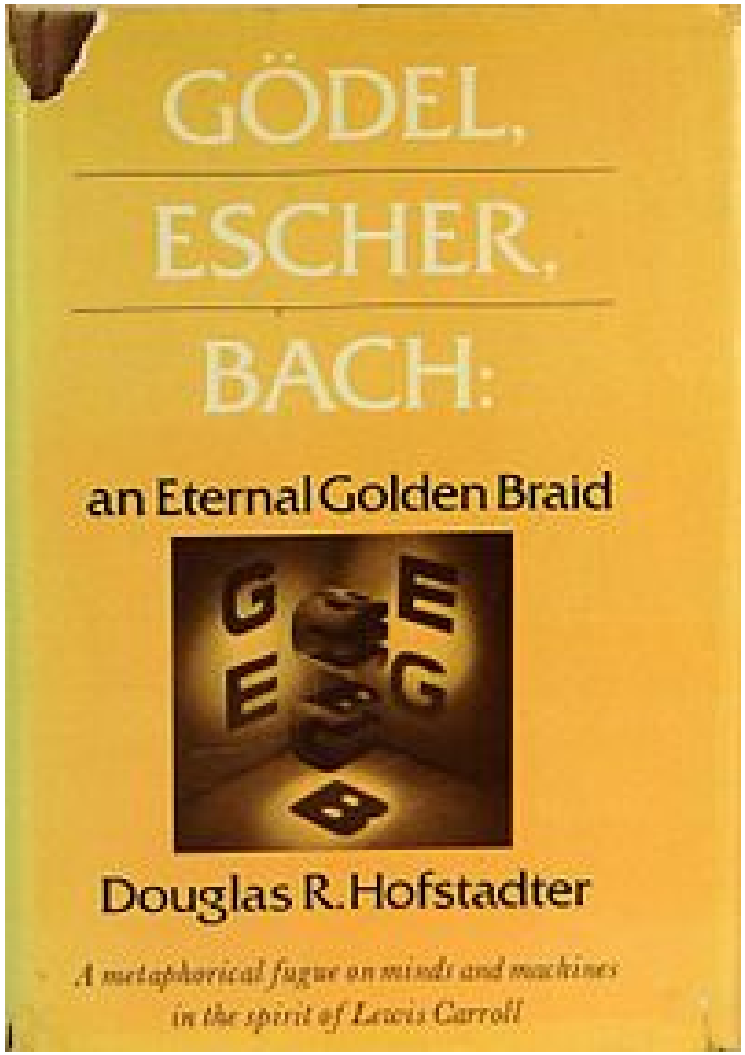
Satz

Die Menge der LOOP-berechenbaren Funktionen
bildet eine echte Teilmenge der berechenbaren Funktionen.

Wdh.: Die Berechenbarkeitslandschaft



Douglas Hofstadter: *"Gödel, Escher, Bach"* (1979)



This book looks at the surprising points of contact between the music of Bach, the artwork of Escher, and the mathematics of Gödel. It also looks at the prospects for computers and artificial intelligence for mimicking human thought.

`BlooP` is a non-Turing-complete programming language in Hofstadter's book whose flow structure is a bounded loop. All programs in the language must terminate, and this language can only express primitive recursive functions.

Vorlesung VL-11
Primitiv rekursive Funktionen

- Primitiv rekursive Funktionen
- Äquivalenz zu LOOP

- μ-rekursive Funktionen
- Äquivalenz zu WHILE

Primitiv rekursive Funktionen (1)

Die primitiv rekursiven Funktionen bilden eine Unterfamilie der Funktionen $\mathbb{N}^k \to \mathbb{N}$ mit $k \geq 1$.

Sie sind induktiv definiert, und werden durch zwei Operationen aus den sogenannten Basisfunktionen zusammengebaut.

Thoralf Skolem (1923):
"Begründung der elementaren Arithmetik durch die rekurrierende Denkweise ohne Anwendung scheinbarer Veränderlichen mit unendlichem Ausdehnungsbereich",
Skrifter utgit av Videnskapsselskapet i Kristiania 6, pp 1–38.

Primitiv rekursive Funktionen (2)

Definition (Basisfunktionen)

Die folgenden drei Funktionsmengen bilden die Basisfunktionen unter den primitiv rekursiven Funktionen:

- Alle konstanten Funktionen sind primitiv rekursiv.
- Alle identischen Abbildungen (Projektionen auf eine der Komponenten) sind primitiv rekursiv.
- Die Nachfolgerfunktion $\mathrm{succ}(n) = n + 1$ ist primitiv rekursiv.

Beispiel: $\pi_{6,4}(a, b, c, d, e, f) = d$

Primitiv rekursive Funktionen (3)

Definition (Operationen)

Weiters sind die folgenden Funktionen primitiv rekursiv:

- Jede Komposition von primitiv rekursiven Funktionen ist primitiv rekursiv.
- Jede Funktion, die durch primitive Rekursion aus primitiv rekursiven Funktionen entsteht ist primitiv rekursiv.
  Wenn die beiden Funktionen $g : \mathbb{N}^k \to \mathbb{N}$ und $h : \mathbb{N}^{k+2} \to \mathbb{N}$ primitiv rekursiv sind, so ist auch die wie folgt definierte Funktion $f : \mathbb{N}^{k+1} \to \mathbb{N}$ primitiv rekursiv:

$$\begin{aligned} f(0, x_1, \ldots, x_k) &= g(x_1, \ldots, x_k) \\ f(n+1, x_1, \ldots, x_k) &= h(n,\ f(n, x_1, \ldots, x_k),\ x_1, \ldots, x_k) \end{aligned}$$

(Anmerkung: Die primitive Rekursion wird über die erste Komponente durchgeführt.)

Intuition zu primitiv rekursive Funktionen

Beobachtung: Jede primitiv rekursive Funktion ist berechenbar und total.

- Klar für Basisfunktionen; und die Komposition von berechenbaren und totalen Funktionen ist ebenfalls berechenbar und total.
- Angenommen, die beiden Funktionen $g : \mathbb{N}^k \to \mathbb{N}$ und $h : \mathbb{N}^{k+2} \to \mathbb{N}$ sind berechenbar. Dann berechnen wir der Reihe nach:

$$\begin{aligned}
y_0 &:= g(x_1, \ldots, x_k) \\
y_1 &:= h(0, y_0, x_1, \ldots, x_k) \\
y_2 &:= h(1, y_1, x_1, \ldots, x_k) \\
y_3 &:= h(2, y_2, x_1, \ldots, x_k) \\
&\vdots \quad \vdots \\
y_n &:= h(n-1, y_{n-1}, x_1, \ldots, x_k)
\end{aligned}$$

Damit haben wir dann auch $f(n, x_1, \ldots, x_k) = y_n$ berechnet.

Beispiel: Addition

Die Additionsfunktion $\text{add} : \mathbb{N}^2 \to \mathbb{N}$ mit $\text{add}(x, y) = x + y$ ist primitiv rekursiv.

$$\begin{aligned} \text{add}(0, x) &= x \\ \text{add}(n+1, x) &= \text{succ}(\text{add}(n, x)) \end{aligned}$$

Genauer: Die Additionsfunktion $\text{add} : \mathbb{N}^2 \to \mathbb{N}$ entsteht durch primitive Rekursion aus der identischen Abbildung $g(x) = x$ und aus der primitiv rekursiven Funktion $h(a, b, c) = \text{succ}(b)$:

$$\begin{aligned} \text{add}(0, x) &= g(x) \\ \text{add}(n+1, x) &= h(n, \text{add}(n, x), x) \end{aligned}$$

Beispiel: Multiplikation

Die Multiplikationsfunktion $\text{mult} : \mathbb{N}^2 \to \mathbb{N}$ mit $\text{mult}(x, y) = x \cdot y$ ist primitiv rekursiv.

$$\begin{aligned} \text{mult}(0, x) &= 0 \\ \text{mult}(n+1, x) &= \text{add}(\text{mult}(n, x), x) \end{aligned}$$

Genauer: Die Multiplikationsfunktion $\text{mult} : \mathbb{N}^2 \to \mathbb{N}$ entsteht durch primitive Rekursion aus der konstanten Abbildung $g(x) = 0$ und aus der primitiv rekursiven Funktion $h(a, b, c) = \text{add}(b, c)$:

$$\begin{aligned} \text{mult}(0, x) &= g(x) \\ \text{mult}(n+1, x) &= h(n, \text{mult}(n, x), x) \end{aligned}$$

Beispiel: Vorgänger und Subtraktion

Die Vorgängerfunktion $\text{pred} : \mathbb{N} \to \mathbb{N}$ mit $\text{pred}(x) = \max\{x - 1, 0\}$ ist primitiv rekursiv.

$$\begin{aligned} \text{pred}(0) &= 0 \\ \text{pred}(n+1) &= n \end{aligned}$$

Die (modifizierte) Subtraktionsfunktion $\text{sub} : \mathbb{N}^2 \to \mathbb{N}$ mit $\text{sub}(x, y) = x \dot{-} y = \max\{x - y, 0\}$ ist primitiv rekursiv.

Dazu definieren wir zunächst die Hilfsfunktion $\text{aux}(y, x) = x \dot{-} y$.

$$\begin{aligned} \text{aux}(0, x) &= x \\ \text{aux}(n+1, x) &= \text{pred}(\text{aux}(n, x)) \end{aligned}$$

Dann definieren wir $\text{sub}(x, y) = \text{aux}(y, x)$. (Anmerkung: Die zugrunde liegende primitive Rekursion wird über die erste Komponente durchgeführt.)

Weitere primitiv rekursive Funktionen

Übung

Die folgenden Funktionen sind primitiv rekursiv:

- Signum: $f : \mathbb{N} \to \{0, 1\}$ mit $f(x) = [x \geq 1]$
- Gleichheit: $f : \mathbb{N}^2 \to \{0, 1\}$ mit $f(x, y) = [x = y]$
- Kleiner: $f : \mathbb{N}^2 \to \{0, 1\}$ mit $f(x, y) = [x < y]$
- Maximum: $f : \mathbb{N}^2 \to \mathbb{N}$ mit $f(x, y) = \max\{x, y\}$
- Minimum: $f : \mathbb{N}^2 \to \mathbb{N}$ mit $f(x, y) = \min\{x, y\}$
- Division: $f : \mathbb{N}^2 \to \mathbb{N}$ mit $f(x, y) = \lfloor x/y \rfloor$
- Parität: $f : \mathbb{N} \to \{0, 1\}$ mit $f(x) = [x \text{ ungerade}]$
- Teilbarkeit: $f : \mathbb{N}^2 \to \{0, 1\}$ mit $f(x, y) = [x|y]$
- Primzahl: $f : \mathbb{N} \to \{0, 1\}$ mit $f(x) = [x \text{ prim}]$

Anmerkung: Für ein logisches Prädikat Prdkt verwenden wir die Schreibweise [Prdkt], wobei $[\text{Prdkt}] = 0$ falls Prdkt falsch, und $[\text{Prdkt}] = 1$ falls Prdkt wahr.

Ein wichtiges Werkzeug: Primitiv rekursive Bijektionen

Eine Bijektion (1)

Die Funktion $\text{binom}_2 : \mathbb{N} \to \mathbb{N}$ mit $\text{binom}_2(x) = \binom{x}{2} = \frac{1}{2}x(x-1)$ ist primitiv rekursiv.

$$\begin{aligned} \text{binom}_2(0) &= 0 \\ \text{binom}_2(n+1) &= \text{add}(n, \text{binom}_2(n)) \end{aligned}$$

Die Funktion $\beta : \mathbb{N}^2 \to \mathbb{N}$ mit

$$\beta(x, y) = \binom{x + y + 1}{2} + x$$

ist primitiv rekursiv.

Eine Bijektion (2)

Beobachtung

Die Funktion $\beta(x, y) = \binom{x+y+1}{2} + x$ ist eine Bijektion zwischen $\mathbb{N}^2$ und $\mathbb{N}$.

x/y	0	1	2	3	4	5	6	$\cdots$
0	0	1	3	6	10	15	21	$\cdots$
1	2	4	7	11	16	22	29	$\cdots$
2	5	8	12	17	23	30	38	$\cdots$
3	9	13	18	24	31	39	48	$\cdots$
4	14	19	25	32	40	49	59	$\cdots$
5	20	26	33	41	50	60	71	$\cdots$
6	27	34	42	51	61	72	84	$\cdots$
7	35	43	52	62	73	85	98	$\cdots$

Die Umkehrfunktion (1)

- Die Funktion $f : \mathbb{N}^3 \to \{0,1\}$ mit $f(x,y,n) = [\beta(x,y) = n]$ ist primitiv rekursiv.
- Die Funktion $g : \mathbb{N}^3 \to \{0,1\}$ mit $g(x,k,n) = [\exists y \leq k : \beta(x,y) = n]$ ist primitiv rekursiv:

$$\begin{aligned} g(x,0,n) &:= f(x,0,n) \\ g(x,k,n) &:= f(x,k,n) + g(x,k-1,n) \cdot (1 - f(x,k,n)) \end{aligned}$$

- Die Funktion $h : \mathbb{N}^3 \to \mathbb{N}$ mit $h(m,k,n) = \max\{x \leq m : \exists y \leq k : \beta(x,y) = n\}$ ist primitiv rekursiv (wobei wir $\max \emptyset = 0$ definieren):

$$\begin{aligned} h(0,k,n) &:= 0 \\ h(m,k,n) &:= \max\{h(m-1,k,n),\ m \cdot g(m,k,n)\} \end{aligned}$$

- Die Funktion $\gamma : \mathbb{N} \to \mathbb{N}$ mit $\gamma(n) = h(n,n,n)$ ist primitiv rekursiv

Die Umkehrfunktion (2)

Die Funktion $\gamma : \mathbb{N} \to \mathbb{N}$ mit $\gamma(n) = h(n, n, n)$ ist primitiv rekursiv

- Also: $\gamma(n) = h(n, n, n)$ ist die grösste Zahl $x \leq n$, für die eine Zahl $y \leq n$ existiert, sodass $\beta(x, y) = n$ gilt.
- Für jedes $n \in \mathbb{N}$ gibt es ein einziges Zahlenpaar $(x, y) \in \mathbb{N}^2$ mit $\beta(x, y) = n$. Die Funktion $\gamma(n)$ gibt uns daher die (eindeutig bestimmte) Zahl x in dieser Gleichung an.
- Analog: Es gibt eine primitiv rekursive Funktion $\delta(n)$, die die (eindeutig bestimmte) Zahl y in der Gleichung $\beta(x, y) = n$ angibt.

Zusammenfassend:

Die Umkehrfunktionen γ und δ der Bijektion β sind primitiv rekursiv. Für alle $n \in \mathbb{N}$ gilt $\beta(\gamma(n), \delta(n)) = n$.

Weitere Bijektionen

Aus der Bijektion $\beta : \mathbb{N}^2 \to \mathbb{N}$ kann man weitere Bijektionen zwischen $\mathbb{N}^{k+1}$ und $\mathbb{N}$ bauen: Die primitiv rekursive Funktion

$$\beta(n_0, \beta(n_1, \beta(n_2, \ldots, \beta(n_{k-1}, n_k)\ldots))$$

weist jedem $(k+1)$-Tupel $(n_0, n_1, \ldots, n_k)$ eine entsprechende Zahl in $\mathbb{N}$ zu, die wir im Folgenden mit $\langle n_0, n_1, \ldots, n_k \rangle$ bezeichnen.

Mit Hilfe der beiden Funktionen $\gamma(n)$ und $\delta(n)$ kann man dann weitere primitiv rekursive Funktionen $u_0, n_1, \ldots, u_k : \mathbb{N} \to \mathbb{N}$ konstruieren, sodass für alle $n \in \mathbb{N}$ die folgende Beziehung gilt:

$$n \;=\; \langle u_0(n),\, u_1(n),\, \ldots,\, u_k(n) \rangle$$

Äquivalenz zu LOOP Programmen

Äquivalenz zu LOOP Programmen

Satz

Die Menge der primitiv rekursiven Funktionen
fällt mit der Menge der LOOP-berechenbaren zusammen.

Beweis: LOOP $\rightarrow$ Primitiv rekursiv (1)

- Wir betrachten eine LOOP-berechenbare Funktion f, die von einem LOOP Programm P berechnet wird.
- Die im Programm P verwendeten Variablen seien $x_0, x_1, \ldots, x_k$.

Wir beweisen mit Induktion über den Aufbau von P, dass eine primitiv rekursive Funktion $g_P : \mathbb{N} \rightarrow \mathbb{N}$ existiert, die die Arbeitsweise von P simuliert:

Wenn das Programm P die Anfangswerte $a_0, a_1, \ldots, a_k$ der Variablen in die Endwerte $b_0, b_1, \ldots, b_k$ übersetzt, dann gilt entsprechend

$$g_P(\langle a_0, a_1, \ldots, a_k \rangle) = \langle b_0, b_1, \ldots, b_k \rangle$$

Beweis: LOOP $\rightarrow$ Primitiv rekursiv (2)

Zuweisungen

Falls P aus der Zuweisung $x_i := x_j + c$ besteht, so ist

$$g_P(x) = \langle u_0(x), \ldots, u_{i-1}(x), u_j(x) + c, u_{i+1}(x), \ldots, u_k(x) \rangle$$

Hintereinanderausführung

Falls P die Form $Q; R$ hat, so gilt

$$g_P(x) = g_R(\, g_Q(x)\,)$$

Beweis: LOOP $\rightarrow$ Primitiv rekursiv (3)

LOOP-Konstrukt

Angenommen, P ist ein LOOP Programm von der Form

LOOP x_i DO Q ENDLOOP

- Wir definieren zunächst die (primitiv rekursive) Hilfsfunktion $h : \mathbb{N}^2 \rightarrow \mathbb{N}$ mit

$$\begin{aligned} h(0, x) &= x \\ h(n+1, x) &= g_Q(h(n, x)). \end{aligned}$$

- Der Aufruf $h(n, x)$ wendet das Programm Q genau n-mal hintereinander auf die Eingabe x an.
- Da der Wert x_i am Anfang der Schleife durch $u_i(x)$ gegeben ist, gilt

$$g_P(x) = h(u_i(x), x)$$

Beweis: LOOP $\rightarrow$ Primitiv rekursiv (4)

Schlussendlich geben wir Anfang und Ende der Simulation an:

Anfang von LOOP

Die Eingabe ist in den Variablen $x_1, \ldots, x_m$ enthalten.
Alle anderen Variablen werden mit 0 initialisiert.

Wenn die m Eingabevariablen die Werte $x'_1, \ldots, x'_m$ haben,
so ist der Eingabewert für die primitiv rekursive Simulation

$$x \;=\; \langle 0,\; x'_1, x'_2, \ldots, x'_m,\; \underbrace{0, 0, \ldots, 0}_{k-m} \rangle$$

Ende von LOOP

Das Resultat ist die Zahl, die sich am Ende in der Variablen x_0 ergibt.

Das Resultat des LOOP Programs P ergibt sich als

$$u_0\big(\; g_P(\langle 0,\; x'_1, x'_2, \ldots, x'_m,\; 0, 0, \ldots, 0 \rangle)\;\big).$$

Rückrichtung: Primitiv rekursiv $\rightarrow$ LOOP

Beweis: Primitiv rekursiv $\rightarrow$ LOOP (1)

- Wir betrachten eine primitiv rekursive Funktion $f : \mathbb{N}^k \rightarrow \mathbb{N}$
- Wir beweisen mit Induktion über die Definition von f, dass ein LOOP Programm P_f existiert, das die Auswertung von f simuliert: Das Programm P_f nimmt die Variablenwerte $x_1, \ldots, x_k$ und terminiert mit $x_0 = f(x_1, \ldots, x_k)$.

Basisfunktionen

- Die konstante Funktion $f(x_1, \ldots, x_k) = c$ wird durch das LOOP Program $x_0 := c$ simuliert.
- Die Projektion $f(x_1, \ldots, x_k) = x_j$ wird durch das LOOP Program $x_0 := x_j$ simuliert.
- Die Nachfolgerfunktion $\mathrm{succ}(x_j) = x_j + 1$ wird durch das LOOP Program $x_0 := x_j + 1$ simuliert.

Beweis: Primitiv rekursiv $\rightarrow$ LOOP (2)

Komposition

Falls die primitiv rekursive Funktion f durch Komposition aus anderen primitiv rekursiven Funktionen $f_1, \ldots, f_s$ entsteht,
so wird f simuliert, indem man die LOOP Programme für $f_1, \ldots, f_s$ geeignet hintereinander ausführt.

(Zwischenergebnisse kann sich die Simulation für später merken, indem sie in separaten Variablen bis zur Wiederverwendung zwischengespeichert werden.)

Beweis: Primitiv rekursiv → LOOP (3)

Primitive Rekursion

Angenommen, die Funktion $f : \mathbb{N}^{k+1} \to \mathbb{N}$ entsteht durch primitive Rekursion aus den beiden primitiv rekursiven Funktionen $g : \mathbb{N}^k \to \mathbb{N}$ und $h : \mathbb{N}^{k+2} \to \mathbb{N}$, wobei

$$\begin{aligned} f(0, x_1, \ldots, x_k) &= g(x_1, \ldots, x_k) \\ f(n+1, x_1, \ldots, x_k) &= h(n,\ f(n, x_1, \ldots, x_k),\ x_1, \ldots, x_k) \end{aligned}$$

Das simulierende LOOP Programm P_f sieht dann wie folgt aus:

$x_0 := g(x_1, \ldots, x_k);$
$s := 0;$
LOOP n DO
    $x_0 := h(s, x_0,\ x_1, \ldots, x_k);$
    $s := s + 1;$
ENDLOOP;

μ-rekursive Funktionen

Der Kleene'sche μ-Operator (1)

In der folgenden Definition gilt $\min \emptyset = \perp$.

Definition

Es sei $g : \mathbb{N}^{k+1} \to \mathbb{N} \cup \{\perp\}$ eine (partielle oder totale) Funktion mit $k+1$ Argumenten ($k \geq 1$). Durch Anwendung des μ-Operators auf g entsteht eine neue Funktion $f : \mathbb{N}^k \to \mathbb{N} \cup \{\perp\}$ mit k Argumenten, wobei

$$f(x_1, \ldots, x_k) = \min\{n \mid g(n, x_1, \ldots, x_k) = 0 \text{ und } \text{für alle } m < n \text{ gilt } g(m, x_1, \ldots, x_k) \neq \perp\}.$$

Die entstehende Funktion f wird mit $\mu g : \mathbb{N}^k \to \mathbb{N} \cup \{\perp\}$ bezeichnet.

(Anmerkung: Der μ–Operator arbeitet mit der ersten Komponente.)

Der Kleene'sche μ-Operator (2)

Beispiel

Es sei $g : \mathbb{N}^2 \to \mathbb{N}$ mit $g(x, y) = 5x^2 - 4xy - y^2 - 19$.
Dann ist $\mu g(y)$ die kleinste Zahl $x \in \mathbb{N}$ mit $5x^2 - 4xy - y^2 = 19$.

Beispiel

Es sei $g : \mathbb{N}^3 \to \mathbb{N}$ mit $g(x, y, z) \equiv 1$.
Dann gilt $\mu g(y, z) \equiv \bot$.

μ-rekursive Funktionen

Definition

Die Klasse der μ-rekursiven Funktionen ist die kleinste Klasse von (partiellen und totalen) Funktionen,

- die die Basisfunktionen (konstant Funktionen; identischen Abbildungen; Nachfolgerfunktion) enthält und
- die abgeschlossen ist unter Komposition, primitiver Rekursion und Anwendung des μ-Operators.

Stephen Cole Kleene (1909–1994)

Wikipedia: Stephen Kleene was a student of Alonzo Church. Kleene is best known as the founder of recursion theory, which subsequently helped to provide the foundations of theoretical computer science.

Kleene's work grounds the study of computable functions. A number of mathematical concepts are named after him: Kleene hierarchy, Kleene algebra, the Kleene star (Kleene closure), Kleene's recursion theorem, and the Kleene fixpoint theorem. He also invented regular expressions, and made significant contributions to the foundations of mathematical intuitionism.

Äquivalenz zu WHILE Programmen

Äquivalenz zu WHILE Programmen

Satz

Die Menge der μ-rekursiven Funktionen
fällt mit der Menge der WHILE-berechenbaren (Turing-berechenbaren; RAM-berechenbaren) Funktionen zusammen.

- Der Beweis dieses Satz basiert auf dem Beweis unseres letzten Satzes (primitiv rekursiv $\Leftrightarrow$ LOOP-berechenbar), und bildet eine einfache Erweiterung
- In der einen Richtung zeigen wir, dass WHILE Schleifen mit Hilfe des μ-Operators simuliert werden können
- In der anderen Richtung zeigen wir, dass der μ-Operator durch eine WHILE Schleife simuliert werden kann

Beweis: WHILE $\rightarrow$ μ-rekursiv

Simulation eines WHILE Programms P

WHILE $x_i \neq 0$ DO Q ENDWHILE

- Wir recyclen die Hilfsfunktion $h : \mathbb{N}^2 \rightarrow \mathbb{N}$, die durch den Aufruf $h(n, x)$ das Programm Q genau n-mal auf x anwendet:

$$\begin{aligned} h(0, x) &= x \\ h(n+1, x) &= g_Q(h(n, x)). \end{aligned}$$

- $u_i(h(n, x))$ gibt den Wert der Variablen x_i nach n-maliger Anwendung von Q an.
- $\mu u_i(h(n, x))$ gibt dann die kleinste Zahl n an, für die die Variable x_i nach n-maliger Anwendung von Q den Wert 0 hat.
- Daher wird das WHILE Programm P simuliert durch

$$g_P(x) = h(\mu u_i(h(n, x)),\ x)$$

Beweis: μ-rekursiv $\rightarrow$ WHILE

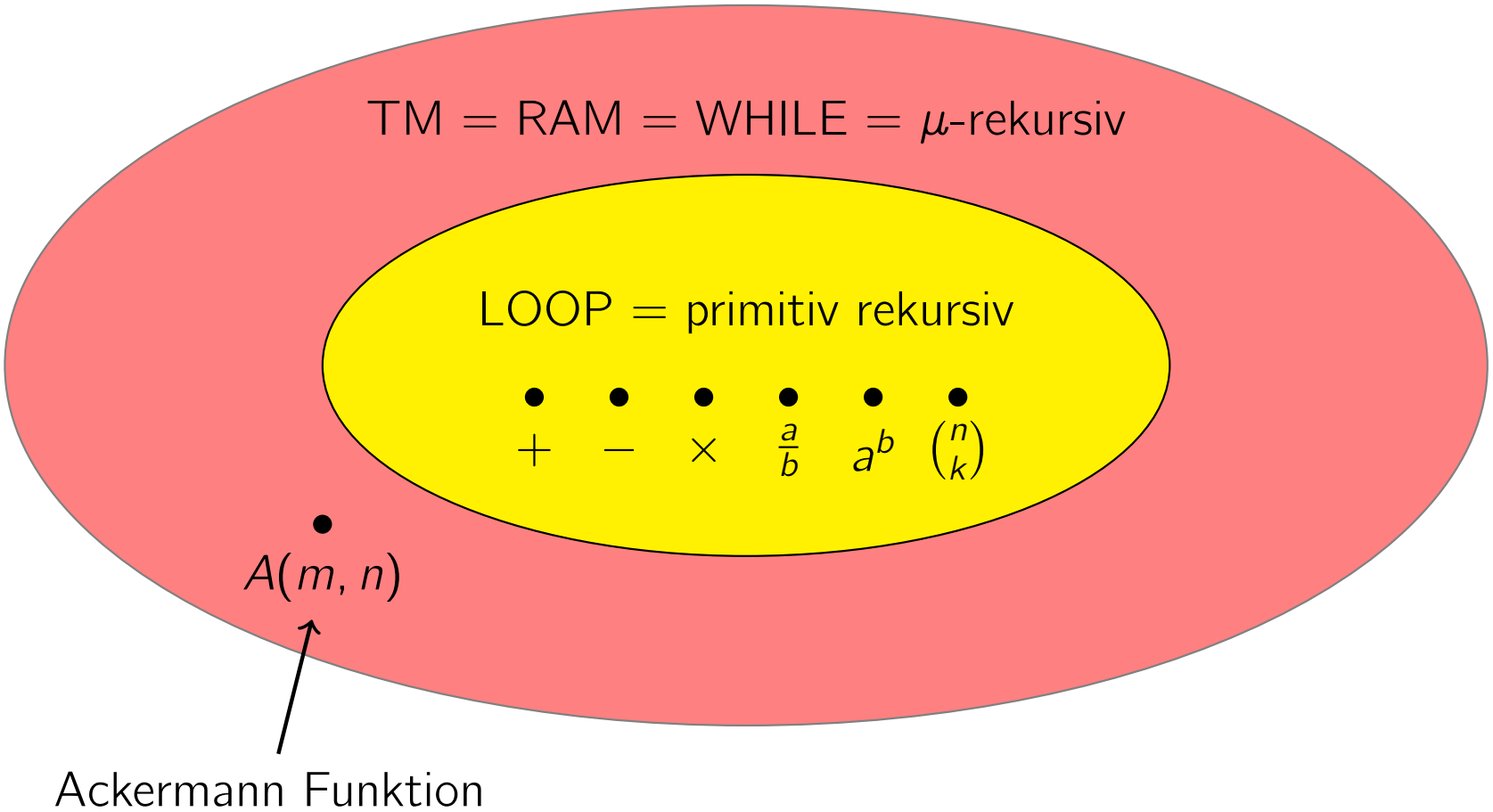
Simulation des μ-Operators

Angenommen, die Funktion $f : \mathbb{N}^k \rightarrow \mathbb{N}$ entsteht durch den μ-Operator aus der μ-rekursiven Funktion $g : \mathbb{N}^{k+1} \rightarrow \mathbb{N}$, also: $f = \mu g$

Das simulierende WHILE Programm P_f sieht dann wie folgt aus:

$x_0 := 0;$
$y := g(0, x_1, \ldots, x_k);$
WHILE $y \neq 0$ DO
    $x_0 := x_0 + 1;$
    $y := g(x_0,\ x_1, \ldots, x_k);$
ENDLOOP;

Landschaftsbild



Zusammenfassung: Berechenbarkeit

Zusammenfassung / Berechenbarkeit (1)

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

- Turingmaschine (TM)
- k-Band-TM
- Registermaschine (RAM)
- WHILE-Programme (und somit C, Java, Pascal, Postscript, etc.)
- μ-rekursive Funktionen (und somit LISP, Haskell, OCaml, etc)

Die folgenden Rechenmodelle und Programmiersprachen sind nicht Turing-mächtig:

- LOOP-Programme
- Primitiv rekursive Funktionen

Zusammenfassung / Berechenbarkeit (2)

Church-Turing These

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

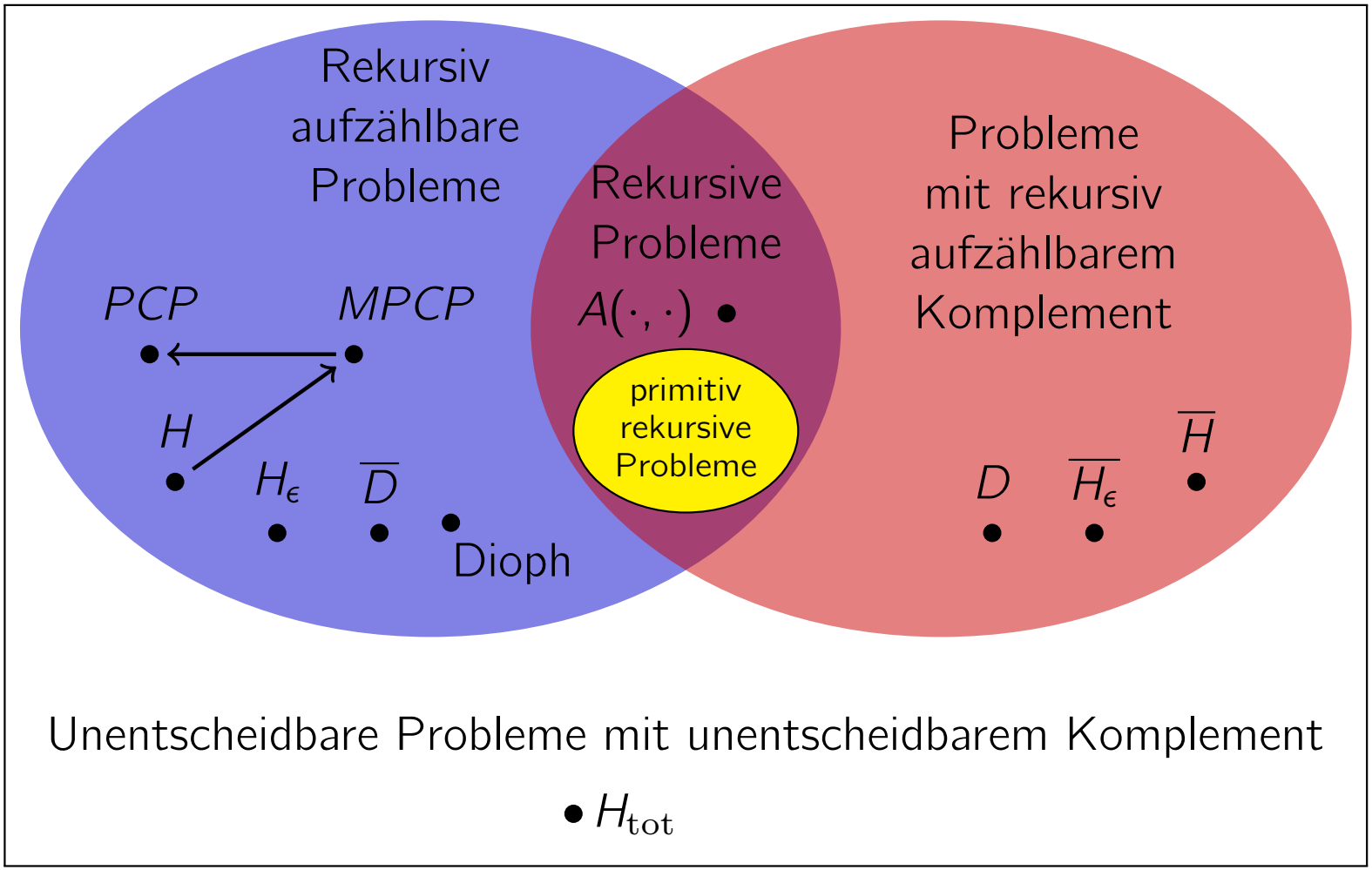
In anderen Worten:

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

An Stelle von "TM" können wir auch jedes andere Turing-mächtige Rechenmodell in die obigen Sätze einfügen.

Zusammenfassung / Berechenbarkeit (3)

Berechenbarkeitslandschaft:



Zusammenfassung / Berechenbarkeit (4)

Methoden zum Nachweis von Nicht-Berechenbarkeit:

- Diagonalisierung
- Unterprogrammtechnik
- Reduktionen (spezielle Variante der Unterprogrammtechnik)

- *Satz von Rice:* Aussagen über Eigenschaften von Funktionen, die durch eine gegebene TM berechnet werden, sind nicht entscheidbar

Zusammenfassung / Berechenbarkeit (5)

Wichtige nicht berechenbare Probleme:

- Halteproblem, in verschiedenen Varianten
- Postsches Korrespondenzproblem
- Verschiedene Entscheidungsprobleme für CFGs
- Erkennung von Funktionen mit elementaren Stammfunktionen
- Hilberts zehntes Problem

- *Schlussfolgerung aus Rice:* Die automatische Verifikation von Programmen in TM-mächtigen Programmiersprachen ist unmöglich