

# Grundbegriffe der Theoretischen Informatik

Sommersemester 2018 - Thomas Schwentick

Teil C: Berechenbarkeit und Entscheidbarkeit

16: Varianten, Einschränkungen und Erweiterungen

Version von: 19. Juni 2018 (12:09)

## And the winner is...

- In 2013 fand in Großbritannien eine Online-Umfrage statt unter dem Motto *Great British Innovations*
- Es sollte die wichtigste Britische Erfindung der vergangenen 100 Jahre gewählt werden:
  - ehemals: <http://www.topbritishinnovations.org>
- Zur Wahl standen unter anderem:
  - 3D-Displays ohne Glas
  - das WWW
  - Flüssigkristall (-Anzeigen)
  - Die Doppelhelix
  - DNA Sequenzierung
  - der Mini Cooper
- Gewonnen hat mit 24% der Stimmen:  
**die universelle Turingmaschine**
- Was das ist, schauen wir uns jetzt an

# Inhalt


## ▷ 16.1 Universelle Turingmaschinen

16.2 Semientscheidbarkeit

16.3  $\mu$ -rekursive und primitiv rekursive Funktionen

16.4 Weitere unentscheidbare Probleme

# Universelle Turingmaschinen: Vorüberlegungen

- Ein wichtiges Merkmal der von Neumann-Architektur von Rechnern ist die prinzipielle Gleichbehandlung von Programmen und Daten:
    - Programme sind nicht „fest verdrahtet“
    - Vielmehr kann der selbe Rechner viele verschiedene Programme ausführen, die im Speicher wie „normale“ Daten repräsentiert werden
  - Wir werden jetzt sehen, dass sich dieses Prinzip auch bei Turingmaschinen anwenden lässt
  - Wir konstruieren dazu jetzt eine feste Turingmaschine  $U$ , die als Eingabe eine (Kodierung einer) Turingmaschine  $M$  und einen String  $x$  erhält und dann  $f_M(x)$  berechnet
    - $U$  wird als „Interpretierer“ arbeiten
  - Genau genommen hat von Neumann dieses Prinzip von Turingmaschinen übernommen
-  Im Folgenden betrachten wir Turingmaschinen ausschließlich über dem Ein-/Ausgabealphabet  $\Sigma = \{0, 1\}$
- Die Resultate gelten aber entsprechend auch für jedes andere feste Alphabet
  - Wir gehen außerdem im Folgenden davon aus, dass die Menge der Zustände und das Arbeitsalphabet einer TM geordnet sind

# Kodierung von Turingmaschinen (1/2)

- Bei der Konstruktion einer universellen Turingmaschine  $U$  ergibt sich eine Komplikation

- $U$  wird ein festes Alphabet und eine feste Zustandsmenge haben

- Die Turingmaschinen, die  $U$  als Eingabe bekommt, können aber verschiedene und beliebig große Arbeitsalphabete und Zustandsmengen haben

➔ Damit  $U$  beliebige TMs verarbeiten kann, müssen diese *kodiert* werden

- Wir kodieren Turingmaschinen über dem Alphabet  $\Sigma = \{0, 1\}$ :
  - Für eine gegebene TM  $M$  sei zunächst num eine Funktion 16.1, die jedem Zustand, jedem Zeichen und jeder Richtung eine Nummer zuordnet:
    - \*  $\text{num} : Q \rightarrow \mathbb{N}$
    - \*  $\text{num} : \Gamma \rightarrow \mathbb{N}$
    - \*  $\text{num} : \{\leftarrow, \downarrow, \rightarrow\} \rightarrow \mathbb{N}$

- Dabei soll immer gelten:

$x$	$\text{num}(x)$
$\leftarrow$	1
$\downarrow$	2
$\rightarrow$	3

$x$	$\text{num}(x)$
$\triangleright$	1
$\sqcup$	2
0	3
1	4

$x$	$\text{num}(x)$
$s$	1
ja	2
nein	3
$h$	4

- Wir kodieren dann Zustände, Zeichen und Richtungen durch 0-1-Strings gemäß:

$$\text{enc}(q) \stackrel{\text{def}}{=} 0^{\text{num}(q)} 1$$

$$\text{enc}(\sigma) \stackrel{\text{def}}{=} 0^{\text{num}(\sigma)} 1$$

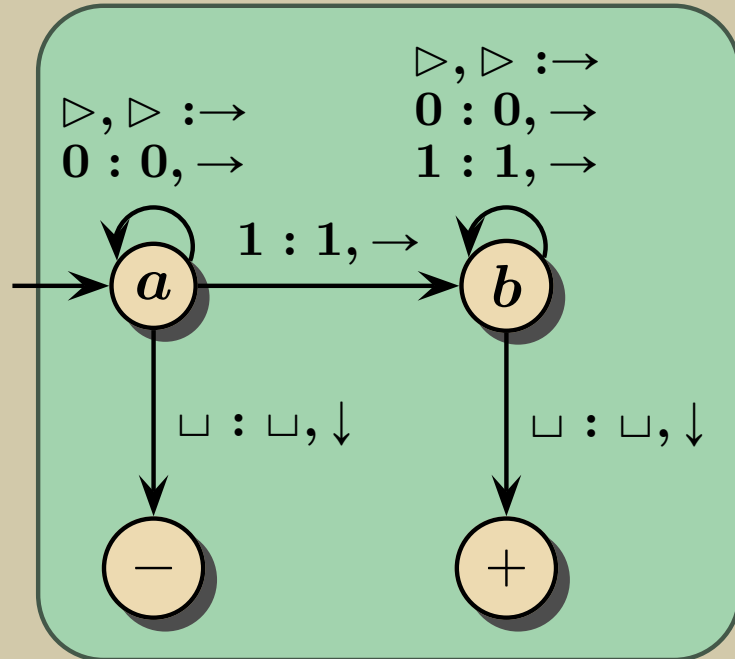
$$\text{enc}(d) \stackrel{\text{def}}{=} 0^{\text{num}(d)} 1$$

- $\delta(q, \sigma) = (q', \sigma', d)$  kodieren wir durch den String  $\text{enc}(q, \sigma) \stackrel{\text{def}}{=} 1\text{enc}(q)\text{enc}(\sigma)\text{enc}(q')\text{enc}(\sigma')\text{enc}(d)$

- Die Kodierung einer TM  $M$ :
  - $\text{enc}(M) \stackrel{\text{def}}{=}$  Konkatenation der Strings  $\text{enc}(q, \sigma)$  in lexikographischer Ordnung nach  $\text{num}(q), \text{num}(\sigma)$

# Kodierung von Turingmaschinen: Beispiel

## Beispiel-Turingmaschine



- Zur Erinnerung:

$x$	$\text{num}(x)$	$x$	$\text{num}(x)$	$x$	$\text{num}(x)$
$\leftarrow$	1	$\triangleright$	1	$s$	1
$\downarrow$	2	$\sqcup$	2	ja	2
$\rightarrow$	3	0	3	nein	3
		1	4	$h$	4

## Beispiel-TM als String

- Da  $s = a$  muss nur noch die Kodierung für  $b$  ergänzt werden:

$x$	$\text{num}(x)$
$a$	1
$b$	5




- Also:

$q, \sigma$	$\text{enc}(q, \sigma)$
$a, \triangleright$	1010101010001
$a, \sqcup$	1010010001001001
$a, 0$	10100010100010001
$a, 1$	10100001000001000010001
$b, \triangleright$	10000101000001010001
$b, \sqcup$	1000001001001001001
$b, 0$	1000001000100000100010001
$b, 1$	100000100001000001000010001

- Die Kodierung der TM ist dann:

10101010100011010010001001001101  
 00010100010001101000010000010000  
 10001100001010000010100011000001  
 00100100100110000010001000001000  
 10001100000100001000001000010001

## Kodierung von Turingmaschinen (2/2)

- Strings, die Turingmaschinen kodieren, müssen (bisher) eine spezielle Form haben
  - Zum Beispiel müssen sie mindestens von der Form  $(10^+10^+10^+10^+10^+1)^*$  sein
- Wir würden aber gerne *jedem* 0-1-String  $w$  eine TM  $M_w$  zuordnen
- Deshalb definieren wir:
  - $\underline{M_w} \stackrel{\text{def}}{=}$ 
    - \* die TM  $M$  mit  $\text{enc}(M) = w$ , falls eine solche TM  $M$  existiert
    - \* die TM  $M_-$ , die bei Lesen des linken Rand-symbols  $\triangleright$  sofort in den ablehnenden Zustand übergeht, andernfalls
-  Ob es zu einem String  $w$  eine TM  $M$  mit  $\text{enc}(M) = w$  gibt, lässt sich von einer TM überprüfen
-   $M$  ist eindeutig  bis auf die Namen der Zustände

# Existenz einer universellen Turingmaschine


## Satz 16.1

- Es gibt eine universelle Turingmaschine, d.h. eine Turingmaschine  $U$ , die für jede TM  $M$  und jeden 0-1-String  $x$  die folgenden Bedingungen erfüllt:
  - Falls  $M$  die Eingabe  $x$  akzeptiert, so akzeptiert  $U$  die Eingabe  $\text{enc}(M)\#x$
  - Falls  $M$  die Eingabe  $x$  ablehnt, so lehnt  $U$  die Eingabe  $\text{enc}(M)\#x$  ab
  - Falls  $M$  bei Eingabe  $x$  nicht terminiert, so terminiert  $U$  bei Eingabe  $\text{enc}(M)\#x$  auch nicht

## Beweisskizze

- Wir konstruieren  $U$  als 4-String TM:
  - String 1:  
Inhalt des Arbeits-Strings von  $M$
  - String 2:  $\text{enc}(M)$
  - String 3: Zustand von  $M$
  - String 4: Hilfsstring für Kopieroperationen

## Beweisskizze (Forts.)

- Bei Eingabe  $\text{enc}(M)\#x$  geht  $U$  wie folgt vor
- Kopiere  $\text{enc}(M)$  auf String 2
- Ersetze  $\#$  durch  $\triangleright$  auf String 1 und ersetze  $x$  durch
$$\text{enc}(x) \stackrel{\text{def}}{=} \text{enc}(x_1) \cdot \dots \cdot \text{enc}(x_{|x|})$$
- Schreibe 01 auf String 3  Startzustand
- Simuliere  $M$  Schritt für Schritt mit Hilfe von  $\text{enc}(M)$  auf String 2

## Bemerkungen

- $U$  kann auch überprüfen, ob die Eingabe überhaupt von der Form  $\text{enc}(M)\#x$  ist und ablehnen, falls dies nicht der Fall ist
- Die Konstruktion kann auch für Turingmaschinen, die Funktionen berechnen, angepasst werden
- Wie in Satz 13.3 kann  $U$  auch als 1-String-TM konstruiert werden



# Inhalt

16.1 Universelle Turingmaschinen

▷ **16.2 Semientscheidbarkeit**

16.3  $\mu$ -rekursive und primitiv rekursive Funktionen

16.4 Weitere unentscheidbare Probleme

# Semientscheidbare Sprachen

## Definition (semientscheidbar)

- Eine Menge  $L \subseteq \Sigma^*$  heißt se-  
mientscheidbar, falls es eine TM  $M$  mit  $L = L(M)$  gibt
- **Zu beachten:**
  - Bei einer semientscheidbaren Menge ist erlaubt, dass die TM für manche Eingaben  $x \notin L$  nicht terminiert
- Klar: jede entscheidbare Sprache ist auch semientscheidbar
- Ein „SemiEntscheidungsalgorithmus“ für  $A$  ist ein Algorithmus, der
  - für alle „Ja-Eingaben“ anhält und akzeptiert und
  - für alle anderen Eingaben ablehnt oder nicht anhält

## Beispiel

- CFG-SCHNITT ist semientscheidbar
  - Ein Algorithmus für CFG-SCHNITT kann alle Strings  $w$  über dem Terminalalphabet der Länge nach erzeugen und mit dem CYK-Algorithmus jeweils testen, ob  $w \in L(G_1)$  und  $w \in L(G_2)$ 
    - \* Wenn er ein solches  $w$  findet, akzeptiert er
    - \* Wenn er kein solches  $w$  findet, terminiert er nicht
- Weitere semientscheidbare Probleme:
  - TM-DIAG, TM-HALT, PCP,...
- Dass TM-HALT semientscheidbar aber nicht entscheidbar ist lässt sich wie folgt interpretieren:
  - es gibt keine einfachere, allgemeine Methode, etwas über das Verhalten einer TM herauszufinden als sie zu simulieren

# Entscheidbar vs. Semientscheidbar

- Die Begriffe „entscheidbar“ und „semientscheidbar“ sind eng miteinander verknüpft, wie das folgende Lemma zeigt

## Lemma 16.2

- Sei  $L \subseteq \Sigma^*$  eine Sprache
- Dann gilt:  
$$L \text{ entscheidbar} \iff L \text{ und } \overline{L} \text{ semientscheidbar}$$
- Dabei bezeichnet  $\overline{L}$  wieder das Komplement  $\Sigma^* - L$  von  $L$

## Beweisidee

- „ $\Rightarrow$ “:
  - Sei  $M$  eine TM, die  $L$  entscheidet
  - ➡  $L = L(M)$ , also ist  $L$  semientscheidbar
  - Vertauschen von ja und nein in  $M$  ergibt eine immer terminierende TM  $M'$  mit  $L(M') = \overline{L}$
  - ➡  $\overline{L}$  ist auch semientscheidbar
- „ $\Leftarrow$ “:
  - Seien  $M_1, M_2$  Turingmaschinen mit  $L = L(M_1)$  und  $\overline{L} = L(M_2)$
  - Sei  $M$  die TM, die, bei Eingabe  $w$ , beide Turing-Maschinen mit Eingabe  $w$  simultan simuliert
    - \* Falls  $M_1$  akzeptieren würde, so akzeptiert  $M$
    - \* Falls  $M_2$  akzeptieren würde, so lehnt  $M$  ab
  - Da einer der beiden Fälle irgendwann zutreffen muss, terminiert  $M$  nach endlich vielen Schritten
- Zu beachten: es ist wichtig, dass die beiden Simulationen *simultan* und nicht nacheinander stattfinden

# Nicht semientscheidbare Probleme

- Lemma 16.2 liefert eine einfache Methode zum Nachweis, dass eine Sprache nicht semientscheidbar ist

## Definition (CFGDISJUNKT)

**Gegeben:** Kontextfreie Grammatiken  $G_1, G_2$

**Frage:** Gilt  $L(G_1) \cap L(G_2) = \emptyset$ ?

## Lemma 16.3

- CFGDISJUNKT ist nicht semientscheidbar

## Beweis

- CFGDISJUNKT ist das Komplement von CFG-SCHNITT
- CFG-SCHNITT ist semientscheidbar
- Wäre das CFGDISJUNKT auch semientscheidbar, so wäre gemäß Lemma 16.2 CFG-SCHNITT entscheidbar

➡ Widerspruch

# Semientscheidbar vs. rekursiv aufzählbar (1/2)

- Die semientscheidbaren Sprachen werden häufig auch „rekursiv aufzählbar“ genannt
- Warum dies so ist, betrachten wir als nächstes

## Definition (rekursiv aufzählbar)

- Eine Sprache  $L \subseteq \Sigma^*$ , heißt rekursiv aufzählbar, falls es eine (2-String-) TM  $M$  gibt, die nach und nach alle Elemente von  $L$  auf ihren zweiten String schreibt
  - Die Strings aus  $L$  werden dabei durch  $\#$  getrennt
  - Der Zeiger des zweiten Strings bewegt sich niemals nach links
- Wir sagen, dass  $M$  die Strings aus  $L$  nach und nach „ausgibt“
- Zu beachten:  $M$  hält bei unendlichen Sprachen  $L$  nicht an

## Semientscheidbar vs. rekursiv aufzählbar (2/2)

### Lemma 16.4

- Sei  $L$  eine Sprache
- Dann gilt:  
 $L$  rekursiv aufzählbar  $\iff L$  semientscheidbar

### Beweisskizze „ $\Rightarrow$ “

- Sei  $L$  rekursiv aufzählbar
- Sei  $M$  eine 2-String-TM, die die Strings aus  $L$  nach und nach auf ihrem zweiten String erzeugt
- Eine TM  $M'$ , die  $L$  „semientscheidet“ kann wie folgt arbeiten:
  - Bei Eingabe  $x$  simuliert  $M'$  die TM  $M$
  - Falls diese den String  $x$  auf dem zweiten String ausgeben würde, akzeptiert  $M'$
  - Falls dies nicht passiert, hält  $M'$  nicht an

### Beweisskizze „ $\Leftarrow$ “

- Sei  $M$  eine TM mit  $L(M) = L$
- Eine „Aufzählungs-TM“  $M'$  für  $L$  kann wie folgt vorgehen:

- 1: **for**  $n := 1$  **TO**  $\infty$  **do**
- 2:   **for** alle  $w$  mit  $|w| \leq n$  **do**
- 3:     Simuliere  $M$  bei Eingabe  $w$  für  $n$  Schritte
- 4:     **if**  $M$  akzeptiert dabei **then**
- 5:       Füge  $w$  auf String 2 an

## Entscheidbar vs. Berechenbar (1/2)

- Der Zusammenhang zwischen berechenbaren Funktionen und entscheidbaren Mengen lässt sich mit Hilfe des Begriffs der *charakteristischen Funktionen* präzise formulieren

### Definition (charakteristische Funktion)

- Sei  $L \subseteq \Sigma^*$  eine Sprache
- Die **charakteristische Funktion** von  $L$ ,  $\chi_L: \Sigma^* \rightarrow \{0, 1\}$ , ist definiert durch:
$$\chi_L(w) \stackrel{\text{def}}{=} \begin{cases} 1 & w \in L \\ 0 & w \notin L \end{cases}$$
- Die **partielle charakteristische Funktion** von  $L$ ,  $\chi'_L: \Sigma^* \rightarrow \{0, 1\}$ , ist definiert durch:
$$\chi'_L(w) \stackrel{\text{def}}{=} \begin{cases} 1 & w \in L \\ \perp & w \notin L \end{cases}$$

## Entscheidbar vs. Berechenbar (2/2)

- Aus Sicht der zur Berechnung verwendeten Turingmaschine ist der Unterschied zwischen dem Entscheiden einer Sprache  $L$  und der Berechnung ihrer charakteristischen Funktion  $\chi_L$  rein formal:
  - Beim Entscheiden von  $L$  geht sie für Strings  $w \in L$  am Ende in den Zustand ja
  - Beim Berechnen von  $\chi_L$  gibt sie für Strings  $w \in L$  den Wert 1 aus
- Das ist also im Grunde nur eine Frage des *User Interfaces*

### Lemma 16.5

- Sei  $L \subseteq \Sigma^*$  eine Sprache
- Dann gelten:
  - (a)  $L$  entscheidbar  $\iff \chi_L$  berechenbar
  - (b)  $L$  semientscheidbar  $\iff \chi'_L$  berechenbar

### Beweis

- Wir betrachten den Beweis von (a) „ $\Rightarrow$ “
  - Die anderen Beweise sind analog
- Sei  $L$  entscheidbar
  - ➔ Dann gibt es eine TM  $M$ , die  $L$  entscheidet
  - Aus  $M$  lässt sich eine TM  $M'$  konstruieren, die, wenn  $M$  in den Zustand „ja“ gehen würde, im letzten String  $1\sqcup$  neben  $\triangleright$  schreibt, den Zeiger an den linken Rand bewegt und anhält
    - (und analog für Zustand „nein“ mit  $0\sqcup$ )
  - ➔  $f_{M'} = \chi_L$
  - ➔  $\chi_L$  berechenbar



# Inhalt

16.1 Universelle Turingmaschinen

16.2 Semientscheidbarkeit

▷ **16.3  $\mu$ -rekursive und primitiv rekursive Funktionen**

16.4 Weitere unentscheidbare Probleme

# Rekursive Funktionen: Vorüberlegungen

## Beispiel

- Die Addition natürlicher Zahlen lässt sich induktiv mit Hilfe der „+1“-Funktion definieren:

$$- m + 0 \stackrel{\text{def}}{=} m$$

$$- m + (n + 1) \stackrel{\text{def}}{=} (m + n) + 1$$

- Analog lässt sich die Multiplikation induktiv mit Hilfe der Addition definieren:

$$- m \times 0 \stackrel{\text{def}}{=} 0$$

$$- m \times (n + 1) \stackrel{\text{def}}{=} (m \times n) + m$$

- Wir werden jetzt genauer untersuchen, welche Funktionen sich mit solchen induktiven Definitionen beschreiben lassen



Wir verwenden für arithmetische Operationen Funktionsnotation anstelle der Infixnotation

- Mit  $s : \mathbb{N}_0 \rightarrow \mathbb{N}_0$  bezeichnen wir die Funktion  $n \mapsto n + 1$

## Beispiel

- Formal definieren wir die 2-stellige Funktion  $\text{add} : \mathbb{N}_0^2 \rightarrow \mathbb{N}_0$  induktiv wie folgt:

$$\bullet \text{ add}(0, m) \stackrel{\text{def}}{=} m, \text{ für alle } m \in \mathbb{N}_0$$

$$\bullet \text{ add}(n + 1, m) \stackrel{\text{def}}{=} s(\text{add}(n, m)), \\ \text{für alle } n, m \in \mathbb{N}_0$$

- Analog definieren wir  $\text{mult} : \mathbb{N}_0^2 \rightarrow \mathbb{N}_0$  durch:

$$\bullet \text{ mult}(0, m) \stackrel{\text{def}}{=} 0$$

$$\bullet \text{ mult}(n + 1, m) \stackrel{\text{def}}{=} \text{add}(\text{mult}(n, m), m)$$




- Wir verwenden hier also
  - gewisse Basisfunktionen ( $s, 0$ ),
  - Komposition von Funktionen, und
  - rekursive Definitionen

- Dies führt uns zur Definition der *primitiv rekursiven Funktionen*

# Primitiv rekursive Funktionen: Definition (1/2)

- Die primitiv rekursiven Funktionen sind induktiv wie folgt definiert

## Definition (PR: Basisfunktionen)

- Alle Funktionen  $c^k : \mathbb{N}_0^k \rightarrow \mathbb{N}_0$  mit  $c \in \mathbb{N}_0$ , definiert durch
  - $c^k(x_1, \dots, x_k) \stackrel{\text{def}}{=} c$sind primitiv rekursiv  konstante Funktionen
- Alle Funktionen  $\pi_i^k : \mathbb{N}_0^k \rightarrow \mathbb{N}_0$ , definiert durch
  - $\pi_i^k(x_1, \dots, x_k) \stackrel{\text{def}}{=} x_i$sind primitiv rekursiv  Projektionen
- Die Funktion  $s : \mathbb{N}_0 \rightarrow \mathbb{N}_0$ , definiert durch
  - $s(x_1) \stackrel{\text{def}}{=} x_1 + 1$ist primitiv rekursiv  Nachfolgerfunktion

## Primitiv rekursive Funktionen: Definition (2/2)

### Definition (PR: Zusammengesetzte Funktionen)

- Sind die Funktionen

- $h : \mathbb{N}_0^\ell \rightarrow \mathbb{N}_0$  und

- $g_1, \dots, g_\ell : \mathbb{N}_0^k \rightarrow \mathbb{N}_0$

primitiv rekursiv, so auch die durch

- $f(x_1, \dots, x_k) \stackrel{\text{def}}{=} h(g_1(x_1, \dots, x_k), \dots, g_\ell(x_1, \dots, x_k))$

definierte Funktion  $f : \mathbb{N}_0^k \rightarrow \mathbb{N}_0$

 Komposition

- Sind die Funktionen

- $g : \mathbb{N}_0^{k-1} \rightarrow \mathbb{N}_0$  und

- $h : \mathbb{N}_0^{k+1} \rightarrow \mathbb{N}_0$

primitiv rekursiv, so auch die durch

- $f(0, x_2, \dots, x_k) \stackrel{\text{def}}{=} g(x_2, \dots, x_k)$

- $f(x_1 + 1, x_2, \dots, x_k) \stackrel{\text{def}}{=} h(f(x_1, x_2, \dots, x_k), x_1, x_2, \dots, x_k)$

definierte Funktion  $f : \mathbb{N}_0^k \rightarrow \mathbb{N}_0$

 Primitive Rekursion

# Primitiv rekursive Funktionen: Beispiele (1/2)

- Wenn wir die Definition der primitiv rekursiven Funktionen ganz genau befolgen, erhalten wir sehr schlecht lesbare Funktionsdefinitionen
- Deshalb verwenden wir eine leicht abkürzende Notation für die Konstanten und Projektionen und erlauben Variablen wie  $x, y, z$

## Beispiel: Addition

- $\text{add}(0, x_2) \stackrel{\text{def}}{=} \pi_1^1(x_2)$ 
  - Abkürzende Notation:  $\text{add}(0, y) \stackrel{\text{def}}{=} y$
- $\text{add}(x_1 + 1, x_2) \stackrel{\text{def}}{=} s(\pi_1^3(\text{add}(x_1, x_2), x_1, x_2))$ 
  - 🖋 Streng genommen müsste die Komposition  $s \circ \pi_1^3$  sogar zuerst als neue Funktion definiert werden...
  - Abkürzende Notation:  
 $\text{add}(x + 1, y) \stackrel{\text{def}}{=} s(\text{add}(x, y))$

## Beispiel: Multiplikation

- Multiplikation in abkürzender Notation:
  - Abkürzende Notation:  $\text{mult}(0, y) \stackrel{\text{def}}{=} 0$
  - $\text{mult}(x + 1, y) \stackrel{\text{def}}{=} \text{add}(\text{mult}(x, y), y)$

## Beispiel: Signum

- Sei  $\sigma : \mathbb{N}_0 \rightarrow \mathbb{N}_0$  die „Sigma-Funktion“ mit der „üblichen Definition“

$$\sigma(x) \stackrel{\text{def}}{=} \begin{cases} 1, & \text{falls } x > 0 \\ 0, & \text{falls } x = 0 \end{cases}$$

- $\sigma$  ist primitiv rekursiv:
  - $\sigma(0) \stackrel{\text{def}}{=} 0$
  - $\sigma(x + 1) \stackrel{\text{def}}{=} 1$

- Sei  $\tau : \mathbb{N}_0 \rightarrow \mathbb{N}_0$  definiert durch
  - $\tau(0) \stackrel{\text{def}}{=} 1$
  - $\tau(x + 1) \stackrel{\text{def}}{=} 0$

$$\tau \equiv 1 - \sigma$$

## Primitiv rekursive Funktionen: Beispiele (2/2)

- Wir definieren einen „kleinste Nullstelle“-Operator mit „beschränktem Suchraum“

### Beispiel

- Sei  $f(x, y) = (x^2 \div y) + (y \div x^2)$
- Die kleinste Nullstelle von  $f$  bezüglich  $y = 25$  ist 5
- Allgemein ist die „kleinste Nullstelle“ von  $f$  bezüglich eines festen Wertes  $b$  für  $y$  die kleinste Zahl  $a \in \mathbb{N}_0$  mit  $f(a, b) = 0$
- Für eine Funktion  $f : \mathbb{N}_0^k \rightarrow \mathbb{N}_0$  sei  $\text{MIN}_f : \mathbb{N}_0^k \rightarrow \mathbb{N}_0$  definiert durch:  
 $\text{MIN}_f(x, x_2, \dots, x_k) \stackrel{\text{def}}{=} \begin{cases} \min\{n \leq x \mid f(n, x_2, \dots, x_k) = 0\}, & \text{falls so ein } n \text{ existiert} \\ 0, & \text{andernfalls} \end{cases}$

- $\text{MIN}_f(x, \dots) = 0$  kann bedeuten,
  - dass es keine Nullstelle  $\leq x$  gibt, oder
  - dass die kleinste Nullstelle 0 ist

### Beispiel: beschränktes Minimum

- Ist  $f$  primitiv rekursiv, so auch  $\text{MIN}_f$ :

- $\text{MIN}_f(0, x_2, \dots, x_k) \stackrel{\text{def}}{=} 0$
- $\text{MIN}_f(x + 1, x_2, \dots, x_k) \stackrel{\text{def}}{=}$

$$\begin{aligned} & \text{MIN}_f(x, x_2, \dots, x_k) + \\ & \left[ \tau(\text{MIN}_f(x, x_2, \dots, x_k)) \times \right. \\ & \quad \sigma(f(0, x_2, \dots, x_k)) \times \\ & \quad \quad (x + 1) \times \\ & \quad \left. \tau(f(x + 1, x_2, \dots, x_k)) \right] \end{aligned}$$

- Erläuterung:
  - $\text{MIN}_f(x + 1, x_2, \dots, x_k) = \text{MIN}_f(x, x_2, \dots, x_k)$   
falls  $\text{MIN}_f(x, x_2, \dots, x_k) \neq 0$   
oder  $f(0, x_2, \dots, x_k) = 0$
  - Andernfalls ist es  $x + 1$ ,  
falls  $f(x + 1, x_2, \dots, x_k) = 0$
  - Sonst: 0

# $\mu$ -rekursive Funktionen

- Wir betrachten jetzt einen Operator, der „global“ nach der kleinsten Nullstelle sucht

## Definition ( $\mu$ -rekursiven Funktionen)

- Ist  $f : \mathbb{N}_0^{k+1} \rightarrow \mathbb{N}_0$  eine partielle Funktion, so sei die partielle Funktion  $\mu f : \mathbb{N}_0^k \rightarrow \mathbb{N}_0$ , definiert durch:
- $\mu f(x_1, \dots, x_k) \stackrel{\text{def}}{=} \begin{aligned} & - \text{kleinstes } n \in \mathbb{N}_0, \text{ für das gilt:} \\ & * f(n, x_1, \dots, x_k) = 0 \text{ und} \\ & * f(m, x_1, \dots, x_k) \neq \perp, \text{ für alle } m < n \\ & - \perp, \text{ wenn kein solches } n \text{ existiert} \end{aligned}$
- Die  $\mu$ -rekursiven Funktionen sind wie die primitiv rekursiven Funktion definiert mit der zusätzlichen Regel:
  - Ist  $f$   $\mu$ -rekursiv, so auch  $\mu f$

## Beispiel

- Ist  $f(x, y) = (x^2 \dot{-} y) + (y \dot{-} x^2)$ , so ist also  $\mu f(y)$  die nicht negative ganzzahlige Quadratwurzel von  $y$ , falls diese existiert

# $\mu$ -rekursiv vs. WHILE

## Satz 16.6

- Eine partielle Funktion  $f : \mathbb{N}_0^k \rightarrow \mathbb{N}_0$  ist genau dann  $\mu$ -rekursiv, wenn sie durch ein WHILE-Programm berechnet werden kann

## Beweisidee

- „WHILE  $\rightarrow \mu$ -rekursiv“: kompliziert
- „ $\mu$ -rekursiv  $\rightarrow$  WHILE“: sehr einfach
  - Sei  $f : \mathbb{N}_0^{k+1} \rightarrow \mathbb{N}_0$  WHILE-berechenbar
  - Das folgende WHILE-Programm berechnet  $\mu f$ :
    - ☞ Wesentlicher Induktionsschritt

```

$$\begin{aligned} x_{k+1} &:= 0; \\ x_{k+2} &:= f(0, x_1, \dots, x_k); \\ \text{WHILE } x_{k+2} \neq 0 \text{ DO} \\ &\quad x_{k+1} := x_{k+1} + 1; \\ &\quad x_{k+2} := f(x_{k+1}, x_1, \dots, x_k); \\ \text{END;} \\ x_1 &:= x_{k+1} \end{aligned}$$

```

- Dabei ist  $x_{k+2} := f(x_{k+1}, x_1, \dots, x_k)$  eine abkürzende Schreibweise für den Einschub eines WHILE-Programmes zur Berechnung von  $f$
- Zu beachten: es ist möglich, dass das Programm für  $f$  nicht terminiert
  - \* In diesem Fall ist der Wert für  $\mu f$  undefiniert



# LOOP-Programme (1/2)

- LOOP-Programme sind eine Einschränkung von WHILE-Programmen
- Ihre Syntax ergibt sich aus der Syntax von WHILE-Programmen wie folgt:
- Keine bedingte Wiederholung
- Weiteres Schlüsselwort: LOOP
- **(Unbedingte Wiederholung)**  
Falls  $P$  ein LOOP-Programm ist, so auch  
LOOP  $x_i$  DO  $P$  END


## LOOP-Programm 1

```
 $x_1 := x_2;$   
LOOP  $x_3$  DO  $x_1 := x_1 + 1$   
END
```

- Effekt:  $x_1 := x_2 + x_3$

## LOOP-Programm 2

```
 $x_2 := 1;$   
LOOP  $x_1$  DO  $x_2 := 0$  END;  
LOOP  $x_2$  DO  $P$  END
```

 Dabei ist  $P$  ein beliebiges LOOP-Programm

- Effekt: IF  $x_1 = 0$  THEN  $P$  END

## LOOP-Programme (2/2)

### Definition (Semantik von LOOP-Programmen)

- Semantik von LOOP-Schleifen:

Ist  $P$  von der Form

LOOP  $x_i$  DO  $P_1$  END,

so ist  $P(X) \stackrel{\text{def}}{=} P_1^{X(i)}(X)$


- Dabei bezeichnet  $P_1^{X(i)}$  die  $X(i)$ -malige Wiederholung von  $P_1$

- Die Funktion  $f_P$  ist dann wie bei WHILE-Programmen definiert
- $f : \mathbb{N}_0 \rightarrow \mathbb{N}_0$  heißt LOOP-berechenbar, falls  $f = f_P$  für ein LOOP-Programm  $P$

- LOOP-Programme berechnen nur totale berechenbare Funktionen

### Proposition 16.7

- $f$  LOOP-berechenbar  $\Rightarrow$   
 $f$  WHILE-berechenbar
- Die Umkehrung gilt jedoch nicht

 Wir verwenden die gleiche Konvention im Hinblick auf syntaktischen Zucker wie bei LOOP-Programmen

# LOOP-Programme vs. primitive Rekursion

- Erinnerung: LOOP-Programme können beliebig-stellige Funktionen berechnen

## Satz 16.8

- Eine Funktion  $f : \mathbb{N}_0^k \rightarrow \mathbb{N}_0$  ist genau dann primitiv rekursiv, wenn sie durch ein LOOP-Programm berechnet werden kann

## Beweisskizze

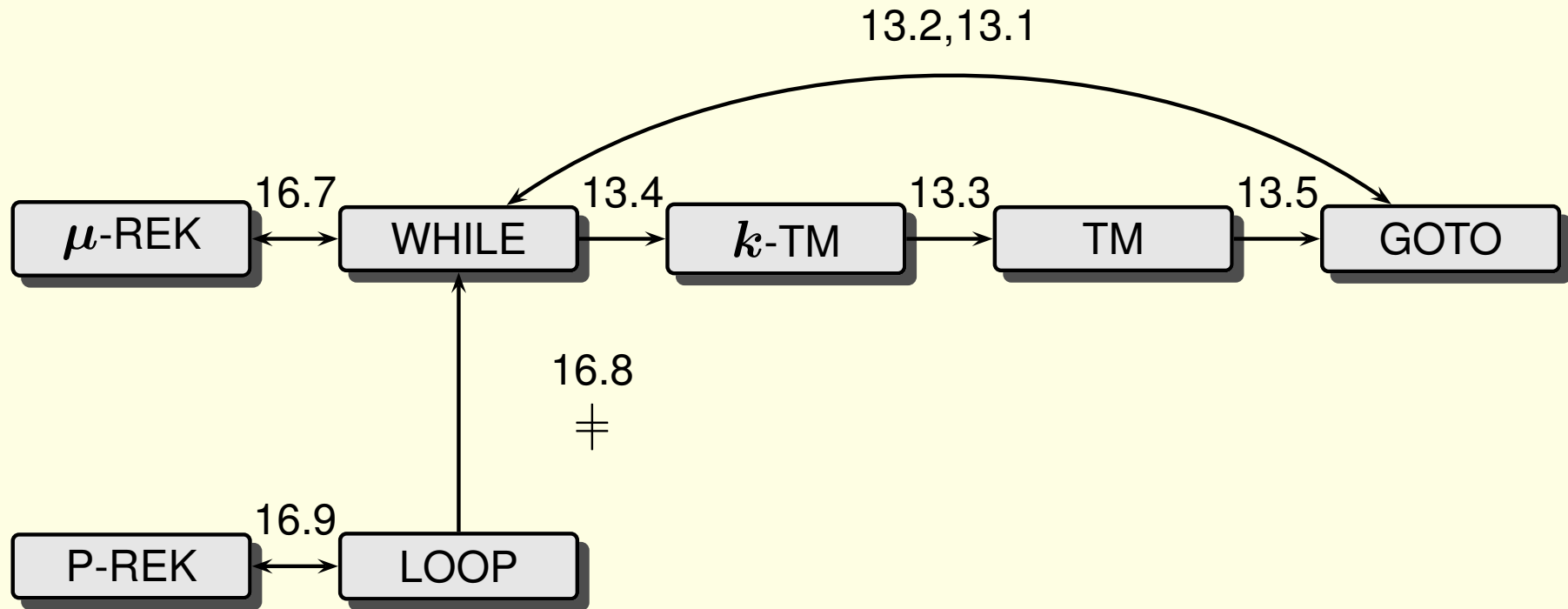
- „**LOOP**  $\rightarrow$  **primitiv rekursiv**“:  
etwas technischer Beweis, findet sich im Buch von Schöning
- „**primitiv rekursiv**  $\rightarrow$  **LOOP**“:  
– Wesentlicher Schritt: Ist  $f$  definiert durch:

$$\begin{aligned} f(0, x_2, \dots, x_k) &\stackrel{\text{def}}{=} g(x_2, \dots, x_k) \\ f(x+1, x_2, \dots, x_k) &\stackrel{\text{def}}{=} h(f(x, x_2, \dots, x_k), x, x_2, \dots, x_k), \end{aligned}$$

so ist  $f(x_1, \dots, x_k)$  berechenbar durch:

```
 $x_{k+1} := 0; x_{k+2} := g(x_2, \dots, x_k);$ 
LOOP  $x_1$  DO
     $x_{k+2} := h(x_{k+2}, x_{k+1}, x_2, \dots, x_k);$ 
     $x_{k+1} := x_{k+1} + 1$ 
END;
 $x_1 := x_{k+2}$ 
```

# Verhältnis der Berechnungsmodelle



# Charakterisierungen der semientscheidbaren Sprachen

- Der folgende Satz fasst verschiedene Charakterisierungen der semientscheidbaren Sprachen zusammen

## Satz 16.9

- Für jede Sprache  $L \subseteq \Sigma^*$  sind die folgenden Aussagen äquivalent:
  - (a)  $L$  ist semientscheidbar
  - (b)  $L$  ist rekursiv aufzählbar
  - (c)  $\chi'_L$  ist (Turing-, GOTO-, WHILE-) berechenbar
  - (d)  $\chi'_L$  ist  $\mu$ -rekursiv
  - (e)  $L = D(f)$  für ein berechenbares  $f$
  - (f)  $L = W(f)$  für ein berechenbares  $f$

# Inhalt

16.1 Universelle Turingmaschinen

16.2 Semientscheidbarkeit

16.3  $\mu$ -rekursive und primitiv rekursive Funktionen

▷ **16.4 Weitere unentscheidbare Probleme**

## Logik und Berechenbarkeit (1/3)

- Wie in Kapitel 12 erwähnt, wurden Turingmaschinen zunächst mit dem Ziel entwickelt, zu zeigen, dass das „Entscheidungsproblem“ keine Lösung hat
- Wir betrachten jetzt kurz einige der erzielten Ergebnisse
- Für die Definition prädikatenlogischer Formeln wird auf die Logik-Vorlesung verwiesen

### Definition (FO-SAT)

**Gegeben:** prädikatenlogische Formel  $\varphi$

**Frage:** Ist  $\varphi$  erfüllbar?

### Definition (FO-FINSAT)

**Gegeben:** prädikatenlogische Formel  $\varphi$

**Frage:** Hat  $\varphi$  ein endliches Modell?

### Definition (FOTaut)

**Gegeben:** prädikatenlogische Formel  $\varphi$

**Frage:** Ist  $\varphi$  allgemein gültig?

## Logik und Berechenbarkeit (2/3)

### Satz 16.10

- (a)  $\overline{\text{FO-SAT}}$ ,  $\text{FO-TAUT}$ , und  $\text{FO-FINSAT}$  sind semientscheidbar, aber nicht entscheidbar
- (b)  $\text{FO-SAT}$  ist nicht semientscheidbar

- Zur Erinnerung: für jede prädikatenlogische Formel  $\varphi$  gilt:
  - $\varphi$  ist Tautologie  $\iff \neg\varphi$  ist unerfüllbar
- Also gelten:
  - $\text{FO-TAUT} \leq \overline{\text{FO-SAT}}$  und
  - $\overline{\text{FO-SAT}} \leq \text{FO-TAUT}$

### Beweisidee

- Dass  $\text{FO-TAUT}$  (und damit auch  $\overline{\text{FO-SAT}}$ ) semientscheidbar ist, folgt aus dem Resolutionssatz:
  - Ist eine Formel eine Tautologie, so gibt es dafür einen (endlichen) Resolutionsbeweis, der in endlich vielen Schritten gefunden werden kann
- Die Semientscheidbarkeit von  $\text{FO-FINSAT}$  folgt, da die endlichen Strukturen systematisch aufgezählt werden können, bis ein endliches Modell gefunden ist
- Die Unentscheidbarkeit von  $\overline{\text{FO-SAT}}$ ,  $\text{FO-TAUT}$ , und  $\text{FO-FINSAT}$  ergibt sich durch Reduktion von  $\text{TM-E-HALT}$ :
  - Zu jeder TM  $M$  lässt sich eine Formel  $\varphi_M$  konstruieren, die genau dann ein Modell hat, wenn  $M(\epsilon)$  terminiert
    - \* (und das Modell enthält dann eine Kodierung dieser Berechnung)
- Dass  $\text{FO-SAT}$  nicht semientscheidbar ist, folgt dann mit Lemma 16.2



# Logik und Berechenbarkeit (3/3)

- Auch der automatische Beweisbarkeit von Aussagen über die Arithmetik sind enge Grenzen gesetzt:

## Definition (FO-NAT-MC)

**Gegeben:** prädikatenlogische Formel  $\varphi$

**Frage:** Gilt  
 $(\mathbb{N}_0, 0, 1, +, \times) \models \varphi$ ?

## Satz 16.11

- FO-NAT-MC ist nicht semientscheidbar

## Beweisidee

- Der Beweis ist durch Reduktion von  $\overline{\text{TM-E-HALT}}$
- Die Idee ist, zu jeder TM  $M$  eine arithmetische Formel  $\psi_M$  zu konstruieren, so dass gilt:  
 $(\mathbb{N}_0, 0, 1, +, \times) \models \psi_M \iff M(\epsilon) \text{ terminiert nicht}$
- Dabei wird das Konzept der *Gödelisierung* verwendet:
  - Wie wir bei der Simulation von TMs durch GOTO-Programme gesehen haben, lassen sich Konfigurationen einer TM in vier Zahlen kodieren
  - Es lässt sich zeigen, dass beliebig lange (endliche) Konfigurationsfolgen auch durch einzelne Zahlen kodiert werden können
  - Außerdem lässt sich eine Formel  $\varphi_M(x)$  konstruieren, die ausdrückt, dass eine gegebene Zahl  $x$  eine terminierende Berechnung von  $M$  bei leerer Eingabe kodiert
  - Dann sei  $\psi_M \stackrel{\text{def}}{=} \neg \exists x \varphi_M(x)$

# Hilberts zehntes Problem

- David Hilbert hat in seiner Rede beim internationalen Mathematikerkongress im Jahre 1900 für das neue Jahrhundert 23 zu lösende Probleme formuliert
- Das zehnte Problem war, ein Verfahren zu finden, das für eine beliebige diophantische Gleichung entscheidet, ob sie lösbar ist
- Etwas umformuliert geht es um das folgende algorithmische Problem:

## Definition (HILBERT10)

**Gegeben:** Ganzzahliges Polynom  $f(x_1, \dots, x_k)$

**Frage:** Gibt es  $n_1, \dots, n_k \in \mathbb{Z}$  mit  
$$f(n_1, \dots, n_k) = 0?$$

- Matyasevich hat 1970 bewiesen, dass dieses Problem nicht entscheidbar ist

# Abstufungen unentscheidbarer Probleme (1/2)

- Die nicht entscheidbaren Probleme lassen sich unterteilen in:
  - semientscheidbare Probleme und
  - nicht semientscheidbare Probleme
- Es gibt noch erheblich weitergehende Unterteilungen

- Es lässt sich zeigen, dass eine Sprache  $L$  genau dann semientscheidbar ist, wenn es eine entscheidbare Sprache  $L'$  (von Tupeln von Strings) gibt, so dass für alle Strings  $w \in \Sigma^*$  gilt:  
 $w \in L \iff$   
 $\exists x_1, \dots, x_\ell \in \Sigma^* :$   
 $(w, x_1, \dots, x_\ell) \in L'$

## Arithmetische Hierarchie

- $\Sigma_1^0 \stackrel{\text{def}}{=} \text{semientscheidbare Sprachen}$
- $\Sigma_2^0 \stackrel{\text{def}}{=} \text{Sprachen } L, \text{ für es eine entscheidbare Sprache } L' \text{ gibt, so dass gilt:}$   
 $w \in L \iff$   
 $\exists x_1, \dots, x_\ell \in \Sigma^* \forall y_1, \dots, \forall y_m \in \Sigma^* :$   
 $(w, x_1, \dots, x_\ell, y_1, \dots, y_m) \in L'$
- $\Sigma_k^0 \stackrel{\text{def}}{=} \text{analog mit } k \text{ Quantorenblöcken, beginnend mit einem Block von Existenzquantoren}$
- $\Pi_k^0 \stackrel{\text{def}}{=} \text{analog zu } \Sigma_k^0, \text{ beginnend mit einem Block von Allquantoren}$
- Wir betrachten im Folgenden wieder algorithmische Probleme statt Sprachen
  - Die Quantifizierung kann dann auch über andere abzählbare Mengen erfolgen (z.B.:  $\mathbb{N}$ )

## Abstufungen unentscheidbarer Probleme (2/2)

- Eine Sprache  $L$  heißt *vollständig* für eine Klasse  $\mathcal{C}$ , wenn
  - $L \in \mathcal{C}$  und
  - für jede Sprache  $L' \in \mathcal{C}$  gilt:  
 $L' \leq L$

### Beispiel

- TM-DIAG, TM-HALT, und TM-E-HALT sind vollständig für  $\Sigma_1^0$

- Hält eine gegebene TM  $M$  nur für endlich viele Strings?
  - $\exists m \in \mathbb{N} \forall z \in \Sigma^* \forall n \in \mathbb{N}$ :
    - \* falls  $|z| > m$  läuft  $M(z)$  mindestens  $n$  Schritte ohne zu akzeptieren
  - Dieses Problem ist vollständig für  $\Sigma_2^0$

- Ist die von  $M$  berechnete Funktion  $f_M$  total?
  - $\forall y \in \Sigma^* \exists n \in \mathbb{N}$ :
    - \*  $M(y)$  hält nach spätestens  $n$  Schritten an und erzeugt eine Ausgabe
  - Dieses Problem ist vollständig für  $\Pi_2^0$

- Ist  $W(f_M)$  entscheidbar?
  - $\exists \text{TM } M' \forall y_1, y_2 \in \Sigma^* \forall m \in \mathbb{N}$   
 $\exists z \in \Sigma^* \exists n_1, n_2 \in \mathbb{N}$ :
    - \*  $M'(y_1)$  terminiert nach spätestens  $n_1$  Schritten
    - \* falls  $M(y_2)$  nach  $m$  Schritten die Ausgabe  $y_1$  hat, akzeptiert  $M'(y_1)$ , und
    - \* falls  $M'(y_1)$  akzeptiert, hat  $M(z)$  nach  $n_2$  Schritten die Ausgabe  $y_1$
  - Dieses Problem ist vollständig für  $\Sigma_3^0$

- Es gibt noch die *analytische Hierarchie*...

# Zusammenfassung

- Es gibt universelle Turingmaschinen
- Die  $\mu$ -rekursiven Funktionen sind genau die berechenbaren Funktionen
- Die primitiv rekursiven Funktionen sind genau die LOOP-berechenbaren Funktionen
- Aus der Unentscheidbarkeit des Halteproblems lässt sich folgern, dass die Erfüllbarkeit prädikatenlogischer Formeln und einige verwandte Probleme unentscheidbar sind
- Die unentscheidbaren Probleme lassen sich weiter klassifizieren, z.B., durch die Klassen der Arithmetischen Hierarchie

# Literatur

- Arithmetische Hierarchie:
  - Heribert Vollmer: Berechenbarkeit und Logik:  
Vorlesungsskript, Uni Hannover, 2005

# Erläuterungen

## Bemerkung 16.1

- Wir verwenden den Namen nun hier für formal verschiedene Definitionsbereiche:  $Q, \Gamma, \{\leftarrow, \downarrow, \rightarrow\}$