3. Berechenbarkeit

Wann ist eine Funktion (über den natürlichen Zahlen) berechenbar?

Intuitiv: Wenn es einen Algorithmus gibt, der sie berechnet!

Was heißt, eine Elementaroperation ist maschinell ausführbar?
Was verstehen wir unter einer Rechenmaschine?

Verschiedene Ansätze zur Präzisierung des Berechenbarkeitsbegriffs:

- **Turing-Berechenbarkeit**
- **While-Berechenbarkeit**
- **Goto-Berechenbarkeit**

All diese Präzisierungen (und weitere) beschreiben exakt dieselbe Klasse von Funktionen

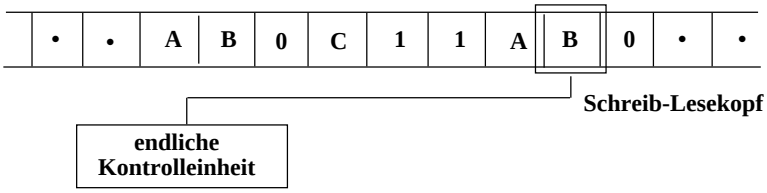
==> **Churchsche These: die so erfaßte Klasse von Funktionen ist identisch mit der Klasse der *intuitiv berechenbaren* Funktionen**

Turing-Maschinen: Ein abstrakes Maschinenmodell

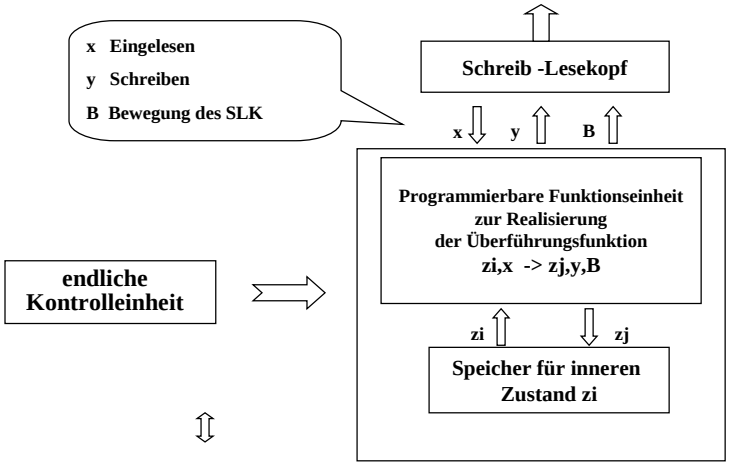
Eine Turing-Maschine M = <Z, Σ, Γ, δ, z0, E> besteht aus

Z: endliche Zustandsmenge
Σ: Eingabealphabet
Γ:Arbeitsalphabet, enthält Σ
δ: Überführungsfunktion Z × Γ ---> Z × Γ × {L, R, N}
z0: Startzustand
E: Menge der Endzustände in Z

Das Arbeitsalphabet enthält das sog. blank Symbol •
Das Band ist als unendlich lang zu denken.



Die Kontrolleinheit



Berechnungen einer TM

Eingabe steht auf Band, pro Feld ein Symbol.
SL-Kopf auf dem am weitesten links stehenden Eingabesymbol S.
Zustand ist Anfangszustand z0.
In Abhängigkeit von Zustand und gelesenem Symbol gibt δ an

- **Nachfolgezustand,**
- **auf Band geschriebenes neues Symbol,**
- **Bewegungsrichtung (Links, Rechts, Nicht bewegen)**

Maschine führt so lange Aktionen aus, bis Zustand aus E erreicht wird
Ausgabe steht nun auf Band.

> **Eine Funktion heißt Turing-berechenbar, wenn es eine Turing-Maschine gibt, die sie (im genannten Sinn) berechnet. Jede Turingmaschine ist durch die Menge ihrer inneren Zustände und durch ihre Überführungsfunktion eindeutig festgelegt.**

Beispiel

Eingabealphabet: {0, 1}, Arbeitsalphabet: {0, 1, •}
Z = {z0, z1, z2, z3}, E = {z3}
Überführungsfunktion [(x, y) -> (z, v, w) statt δ(x,y) = (z, v, w)]:

(z0, •) -> (z3, 1, N)	**(z1, •) -> (z3, 1, N)**	**(z2, •) -> (z2, •, R)**
(z0, 0) -> (z0, •, R)	**(z1, 0) -> (z2, •, R)**	**(z2, 0) -> (z2, •, R)**
(z0, 1) -> (z1, •, R)	**(z1, 1) -> (z1, •, R)**	**(z2, 1) -> (z2, •, R)**

Maschine liefert 1 gdw auf Band eine Folge 0...01...1 steht.
Anzahl jeweils beliebig (auch null)

While-Berechenbarkeit

Ein while-Programm besteht aus folgenden Komponenten:

Variablen: x0, x1, x2, ... **Operationszeichen: +, -, =, !=**
Konstanten: 0, 1, 2, ...
Sonderzeichen: ;, {, }, (,) **Schlüsselwörter: while**

Syntax von While-Programmen, induktive Definition:

1. Eine Wertzuweisung der Form xi = xj + c; oder xi = xj - c; (c Konstante) ist ein while-Programm
2. Falls P1 und P2 while-Programme sind, so auch P1 P2
3. Falls P while-Programm ist, so ist auch
while (xi !=0) {P}
ein while-Programm
4. Nur die durch 1-3 beschiebenen Konstrukte sind while-Programme

Semantik:
Wertzuweisung: xi erhält Wert von xj ± c
Sequenz: erst wird P1 ausgeführt, dann P2
Schleife: P wird so lange ausgeführt, bis xi = 0 gilt. Test *vor* P-Ausführung
Eingabewerte sind Werte der Variablen x1, ..., xn, alle anderen zunächst 0

Beispiel

Multiplikation: Eingabe: x1, x2, Ausgabe: x0 (Vorbelegung x0 = 0)

```
while (x1 !=0)
   {    x3 = x2;
        while (x3 != 0)
          {      x0 = x0 + 1;
                 x3 = x3 – 1;
          }
        x1 = x1 – 1;
   }
```

andere Konstrukte können als Abkürzung eingeführt werden, etwa:

if (x !=0) P1 else P2

statt

```
x1 = 1; x2 = x;
while (x2 !=0) {P1; x2 = 0; x1 = 0; }
while (x1 != 0) {P2; x1 = 0 ;}
```

Funktion ist while-berechenbar: es gibt while-Programm, das sie berechnet.

Goto-Berechenbarkeit

Ein Goto-Programm ist eine Sequenz von Anweisungen Ai mit Marken Mi:
M1: A1; M2: A2; ...; Mk: Ak;
(Marken, zu denen nie gesprungen wird, dürfen entfallen.)

Anweisungen sind
- **Wertzuweisungen: xi = xj ± c;**
- **unbedingter Sprung: goto Mi; (Ai wird als nächstes ausgeführt)**
- **bedingter Sprung: if (xi == c) goto Mi;**
- **Stopanweisung: STOP.**

While-Schleifen lassen sich mit Goto's simulieren: Aus

while (xi !=0) P

wird

```
M1:  if (xi == 0) goto M2;
     P
     goto  M1;
M2:  ...
```

Funktion Goto-berechenbar: Es gibt entsprechendes Goto-Programm

Ein Goto-Programm für die Multiplikation

Annahmen:
Eingabewerte x1 und x2, Ausgabe x0, x0 mit 0 vorbelegt

```
M1: if (x1 == 0) goto M4;
    x3 = x2;
M2: if  (x3 == 0) goto M3;
    x0 = x0 + 1;
    x3 = x3 - 1;
    goto  M2;
M3: x1 = x1 - 1;
    goto  M1;
M4: return x0;
```

Programme mit GOTO schwer verstehbar und kaum verifizierbar.
Konstrukt sollte deshalb beim Programmieren nicht verwendet werden.

Äquivalenz der Präzisierungen von Berechenbarkeit

Theorem:
Die Begriffe
Turing-berechenbar, While-berechenbar und Goto-berechenbar
sind äquivalent.

Deshalb geht man davon aus, daß der intuitive Berechenbarkeitsbegriff in diesen Präzisierungen adäquat erfaßt wurde.

Frage: Gibt es Funktionen (über den natürlichen Zahlen), die

- **mathematisch präzise beschrieben werden können, aber**
- **nicht berechenbar sind?**

Nicht berechenbare Funktionen: Das Halteproblem

Jede Turing-Maschine und jede Eingabe läßt sich eindeutig als natürliche Zahl codieren.

Unter dem speziellen Halteproblem versteht man folgendes Problem:

- **gegeben: natürliche Zahl m**
- **gefragt: terminiert Turing-Maschine mit Code m bei Eingabe m?**

Problem repräsentierbar als Funktion f: Nat --> {0, 1},
f(m) = 1 gdw TM mit Code m hält bei Eingabe m, f(m) = 0 sonst.

Funktion f ist nicht berechenbar!!

Beweisskizze:
Angenommen es gäbe TM M, die f berechnet. Konstruiere daraus TM M' wie folgt: M' führt erst M aus, danach wird getestet, ob das Band den Wert 0 hat. Falls ja terminiert M', falls nein geht M' in Endlosschleife. Sei n der Code von M'.

Falls gilt: M' hält bei Eingabe n
<=> M bei Eingabe n liefert 0
<=> TM mit Code n hält nicht bei Eingabe n
<=> M' hält nicht bei Eingabe n

Widerspruch!!