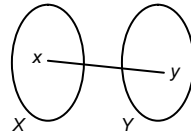


Relationen

Eine Relation bestimmt eindeutig die Menge der geordneten Paare (x, y) , bei denen x zu y in der betreffenden Relation steht.

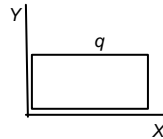


Definition:

Eine *Relation* ist eine Menge geordneter Paare (x, y) , $x \in X$, $y \in Y$.

Die Menge aller geordneten Paare (x, y) ist das Kartesische Produkt $X \times Y$.

Folglich ist eine *Relation* q eine Teilmenge des Kartesischen Produktes $X \times Y$.



Eigenschaften von Relationen

Es sei X die Menge der Menschen in einer Stadt.

Zwischen gewissen Elementen vom X besteht die Vater-Kind-Relation q :

$(x, y) \in q$ bedeutet: x ist Vater von y

Zwischen gewissen Elementen vom X besteht die Geschwister-Relation r :

$(x, y) \in r$ bedeutet: x und y sind Geschwister.

Lexikographische Ordnungsrelation \angle (Ordnung nach Namen):

$(x, y) \in \angle$ bedeutet: x steht alphabetisch vor y .

Eigenschaften von Relationen

Identitäts- oder Gleichheitsrelation:

In jeder Menge X ist jedes Element $x \in X$ mit sich selbst identisch:

$(x, x) \in "="$ bedeutet: x ist gleich x .

Kleinste Relation: \emptyset (leere Menge, Teilmenge des Kartesischen Produkts)

Größte Relation: $X \times Y$ (ganzes Kartesisches Produkt)

Eigenschaften von Relationen

q ist transitiv, wenn aus $(x, y) \in q$ und $(y, z) \in q$ folgt: $(x, z) \in q$

q ist reflexiv, wenn $(x, x) \in q$ für alle x gilt.

q ist antireflexiv, wenn für kein Element gilt: $(x, x) \in q$

q ist symmetrisch, wenn aus $(x, y) \in q$ folgt: $(y, x) \in q$

q ist asymmetrisch, wenn $(x, y) \in q$ stets ausschließt: $(y, x) \in q$

q ist antisymmetrisch, wenn aus $(x, y) \in q$ und $(y, x) \in q$ stets folgt: $x = y$

Komposition von Relationen

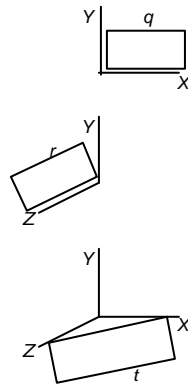
Es seien X, Y (gewöhnliche) Mengen.

Eine *Relation* q von X in Y ist eine Teilmenge des Kartesischen Produktes $X \times Y$ und

Eine *Relation* r von Y in Z ist eine Teilmenge des Kartesischen Produktes $Y \times Z$:

$t := q \circ r$ ist eine Relation von X in Z ,

$t := q \circ r = \{(x, z) \mid \exists y : (x, y) \in q \wedge (y, z) \in r\}$



Fuzzy-Relationen

Seien X, Y (gewöhnliche) Mengen und sei $X \times Y$ deren Kartesisches Produkt.

- $L(X)$: die Menge aller Fuzzy sets in X .
- $L(Y)$: die Menge aller Fuzzy sets in Y .
- $L(X \times Y)$: die Menge aller Fuzzy sets in $X \times Y$.

Eine *Relation* der beiden Mengen X und Y ist Teilmenge von $X \times Y$. Eine *Fuzzy Relation* R von X und Y ist Fuzzy-Teilmenge von $L(X \times Y)$.

Seien drei Mengen X, Y, Z und zwei Fuzzy-Relationen *gegeben* :

- Q in $L(X \times Y)$,
- R in $L(Y \times Z)$.

Wie lassen sich die Fuzzy-Relationen Q und R zu einer neuen Fuzzy-Relation $T \in L(X \times Z)$ kombinieren?

Lotfi Zadeh, 1973: Komposition von Fuzzy-Relationen

\wedge ("und") \rightarrow \min
 \vee ("oder") \rightarrow \max

- Q ist Fuzzy-Relation von X und Y , d. h. Q ist Fuzzy-Teilmenge von $L(X \times Y)$,
- R ist Fuzzy-Relation von Y und Z , d. h. R ist Fuzzy-Teilmenge von $L(Y \times Z)$.
- $T = Q \circ R$ ist Fuzzy-Relation von X und Z ,

d.h.: T ist Fuzzy-Teilmenge von $L(X \times Z)$ mit Zugehörigkeitsfunktion

$$m_T(x, z) = \max_{y \in Y} \min \{m_Q(x, y); m_R(y, z)\}, y \in Y$$

*Outline of a New Approach to the Analysis
of Complex Systems and Decision Processes*

Lotfi A. Zadeh, 1968: Linguistische Variablen

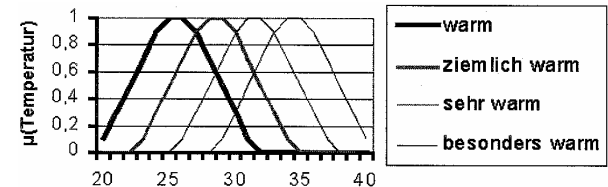
Linguistische Ausdrücke bestehen aus

- Linguistischen Variablen, die durch
- Linguistische Operatoren verknüpft sind.

Werte einer Linguistischen Variablen: Fuzzy sets auf numerischer Skala

Linguistische Operatoren werden durch Fuzzy-Operatoren beschrieben

Beispiel: Linguistische Variable *Temperatur*



Lotfi A. Zadeh, 1968: Linguistische Variable

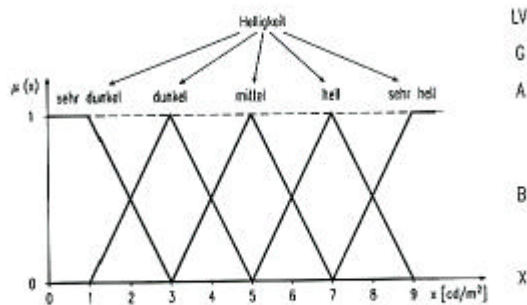


Abb. 4.1. Darstellung der Terme der LV "Helligkeit" über der numerischen Werteskala in $[cd/m^2]$.

Lotfi A. Zadeh, 1968: Linguistische Operatoren

Definition

Ein linguistischer Modifikationsoperator ist ein einstelliger Operator auf einer linguistischen Werteskala. Die Anwendung auf einen Term α führt auf einen neuen Term α' .

Wird der Term α durch ein fuzzy set $A = \{(x, \mu_A(x))\}$ dargestellt, so kann die Modifikation mit Hilfe folgender Mengenoperationen beschrieben werden:

Konzentration $\text{CON}(A)$

Dehnung $\text{DIL}(A)$

Komplementbildung $\neg A$

Kontrastverstärkung $\text{INT}(A)$ $\mu_{\text{INT}}(x) = \begin{cases} 2\mu_A(x)^2 & \text{für } \mu_A(x) \in [0,0.5] \\ 1 - 2[1 - \mu_A(x)]^2 & \text{sonst} \end{cases}$

Lotfi A. Zadeh, 1968: Linguistische Operatoren

Beispielsweise kann die Anwendung eines Modifikators auf einen Term α nach den folgenden Regeln erfolgen:

sehr $\alpha \rightarrow \text{CON}(A)$,

sehr sehr $\alpha \rightarrow \text{CON}[\text{CON}(A)]$,

ziemlich $\alpha \rightarrow \text{CON}(A)$,

Mehr als $\alpha \rightarrow \text{MA} = \{x; \mu_{\text{MA}}(x) \mid \mu_{\text{MA}}(x) = \mu_A(x)^{1,25}\}$,

recht $\alpha \rightarrow \text{INT}[\text{MA} \cap \text{CON}(\neg A)]$

nicht $\alpha \rightarrow \neg A$.

Lotfi A. Zadeh, 1968: Linguistische Operatoren

Beispiel (Baldwin, 1979):

Die Definition der linguistischen Variable „Wahrheit“ nach [Baldwin 1979] geht von folgender Menge A_{Wahrheit} der möglichen Terme aus:

$A_{\text{Wahrheit}} = \{\text{absolut falsch, sehr falsch, falsch, ziemlich falsch, unentschieden, ziemlich wahr, wahr, sehr wahr, absolut wahr}\}$

Lotfi A. Zadeh, 1968: Linguistische Operatoren

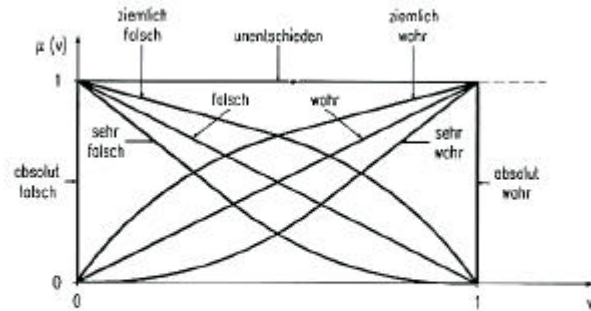


Abb. 4.2. Zugehörigkeitsverläufe für die Terme der LV "Wahrheit" nach [Baldwin, 1979].

Lotfi A. Zadeh, 1968: Linguistische Operatoren

Beispiel (Baldwin, 1979):

Die Zugehörigkeitsfunktionen der Terme berechnen sich folgendermaßen:

$$\begin{aligned} \mu_{\text{absolut wahr}}(v) &= \begin{cases} 1 & \text{für } v = 1 \\ 0 & \text{sonst,} \end{cases} \\ \mu_{\text{absolut falsch}}(v) &= \begin{cases} 1 & \text{für } v = 0 \\ 0 & \text{sonst,} \end{cases} \\ \mu_{\text{wahr}}(v) &= v, \\ \mu_{\text{falsch}}(v) &= 1 - v, \\ \mu_{\text{sehr wahr}}(v) &= v^2, \\ \mu_{\text{sehr falsch}}(v) &= (1 - v)^2, \\ \mu_{\text{ziemlich wahr}}(v) &= v^{1/2}, \\ \mu_{\text{ziemlich falsch}}(v) &= (1 - v)^{1/2}, \\ \mu_{\text{uneingeschieden}}(v) &= 1. \end{aligned}$$



L. A. Zadeh, 1965:
A New View of System Theory

S is a fuzzy system if
 $u(t)$ or
 $y(t)$ or
 $s(t)$ or
any combination are fuzzy sets.

$$s_{t+1} = f(s_t, u_t), \quad t = 0, 1, 2, \dots \quad y_t = g(s_t, u_t)$$

Fuzzy-Automaten

Ein (endlicher) *Fuzzy-Automat* A ist ein 5-tupel (U, V, S, f, g) mit

- U als endlicher, nichtleerer Inputmenge,
- V als endlicher, nichtleerer Outputmenge,
- S als endlicher, nichtleerer Zustandsmenge,
- f als Zugehörigkeitsfunktion eines Fuzzy Sets in $S \times U \times S$ nach $[0,1]$,
- g als Zugehörigkeitsfunktion eines Fuzzy Sets in $V \times U \times S$ nach $[0,1]$.

Die Funktionen f ist die *Fuzzy-Übergangsfunktion*,
die Funktion g ist die *Fuzzy-Output-Funktion*.

Fuzzy-Automaten

Für jedes Fuzzy-Input gibt es eine Zustandsübergangsrelation, die als Übergangsmatrix geschrieben hatte.
Die Matrix für eine Sequenz aus n Inputs ist somit eine n -fache Fuzzy-Relation im Produktraum der n Übergangsmatrizen.

Eine n -wertige Fuzzy-Relation ist ein Fuzzy Set A im Produktraum $X \times X \times \dots \times X$, mit der Zugehörigkeitsfunktion $f_A(x_1, \dots, x_n)$, wobei $x_i \in X$, $x_i = 1, \dots, n$.

$A \circ B$ ist diejenige Fuzzy-Relation, deren Zugehörigkeitsfunktion sich aus denen von A und B folgendermaßen ergibt:

$$f_{A \circ B}(x, y) = \sup_v \min[f_A(x, v), f_B(v, y)]$$

Nur endliche Automaten: Zadehs *Sup-min-Kompositionsregel* wird zur *Max-min-Kompositionsregel* mit Zugehörigkeitsfunktion für den n -Übergang des Fuzzy-Automaten.

Pseudo-Automaten

Ein *Pseudo-Automat* A ist ein System (U, S, f, F, h) mit

U als endlicher, nichtleerer Inputmenge,

S als endlicher, nichtleerer Zustandsmenge,

f als eine Funktion von $S \times U \times S \times T$ nach $[0,1]$, wobei T eine Teilmenge der reellen Geraden ist, d. h. $f(s, u, s', t) \in [0,1]$, für alle $s, s' \in S, u \in U$ und $t \in T$.

f heißt die *Zustandsübergangsfunktion* von A zur Zeit t

F als eine Teilmenge von S ist die Menge der *endlichen Zustände*,

h als eine Funktion von $S \times T$ nach $[0,1]$ heißt die *Anfangsverteilung* von A .

Automaten

$T = \mathbf{N}$, die Menge aller natürlichen Zahlen: A heißt diskret,

f und h von T unabhängig: A heißt stationär.

Elemente aus U heißen *Input-Alphabet* und deren endliche Folgen werden *Input-Bänder (tapes)* genannt.

U^* sei die Menge dieser Inputbänder,

$lg(x)$ sei die Länge eines Bandes-

Ein *Automat* A^* ist ein System (U, S, f^*, F, h) , wobei U, S, F und h wie oben definiert sind.

f^* ist wie f definiert, nur dass U durch U^* ersetzt wurde.

Zu jedem Automaten A^* lässt sich auf natürliche Weise ein Automat A finden, indem f auf f^* eingeschränkt wird.

Probabilistische Automaten

„Probabilistic automata $C(P, RP)$ “

The constraints in CP : for every $s, s' \in S, u \in U$ and $n \in \mathbf{N}$, it is true that

$$\sum_{s' \in S} f(s, u, s', n) = 1 \quad \text{und} \quad \sum_{s' \in S} h(s', n) = 1$$

The rule of extension RP : f^* is defined by induction on $lg(u^*)$, $u^* \in U^*$.

$$f^*(s, e, s', n) = \begin{cases} 1 & \text{if } s = s' \\ 0 & \text{if } s \neq s' \end{cases}$$

$$f^*(s, u^*, u, s', n) = \sum_{s'' \in S} f^*(s, u^*, s'', n) f(s'', u, s', n + lg(u^*))$$

e ist hier das leere Band mit der Eigenschaft $xe = x = ex$ für alle Bänder x .

Probabilistische Automaten

Die Definition deterministischer Automaten ist nun sehr einfach:

Deterministische Automaten $C(CD, RD)$

CD enthält die gleichen Bedingungen wie CP , zusätzlich aber noch die Einschränkung, dass die Funktionen f und h nur die beiden Werte 0 und 1 annehmen dürfen.

Die Erweiterungsregel RD ist gleich der Erweiterungsregel RA .

Probabilistische Automaten

Maximin Automaten $C(CA, RA)$

CA ist die leere Menge, es gibt also keinerlei Einschränkungen an die Funktionen f und h . Die Erweiterungsregel RA für Maximin-Automaten lautet:

$$f^*(s, e, s', n) = \begin{cases} 1 & \text{if } s = s' \\ 0 & \text{if } s \neq s' \end{cases}$$

$$f^*(s, u^*, u, s', n) = \max_{s' \in S} \min \{ f^*(s, u^*, s', n), f(s'', u, s', n + \lg(u^*)) \}$$

e ist hier das leere Band mit der Eigenschaft $xe = x = ex$ für alle Bänder x .

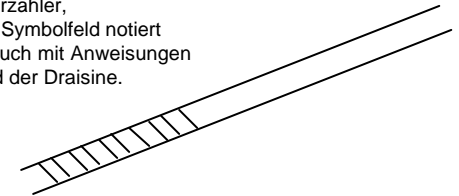
Zwischen Eifa und Alsfeld liegen wieder Schienen ...



Zwar ist die Strecke eigentlich nicht mehr befahrbar, dennoch findet sich hier dieses interessante Gefährt.

Turingmaschine

- Ein nach beiden Seiten unbegrenztes Gleis, auf der die Draisine hin- und herfahren kann,
- zwischen den Schwellen seien Symbole notiert (z. B. 0, 1).
- In der Draisine befindet sich:
 - eine Art Kilometerzähler, der das jeweilige Symbolfeld notiert
 - ein Anweisungsbuch mit Anweisungen für jeden Zustand der Draisine.



Turingmaschine

- Schaue auf den Zählerstand (Wo befinde ich mich?)
- Schlage im Anweisungsbuch nach, ob für diesen Zustand eine Anweisung vorliegt.
- Liegt eine Anweisung für diesen Zustand vor, so führe sie aus:

Anweisungen (Was soll geschehen?):

- Soll das gelesene Symbol stehen bleiben oder durch ein anderes überschrieben werden?
- Wohin soll die Draisine fahren (eine Schwelle weiter nach rechts oder nach links)?
- Wie soll der neue Zählerstand aussehen?

Turingmaschine

Beispiel:

Befindet sich die Maschine im Zustand 1, so besagt die Anweisung im Anweisungsbuch:
Wenn das Feld, über dem der Lesekopf steht, leer ist, dann drucke eine „1“ und bleibe im Zustand 1. Steht aber eine „1“ auf dem Feld, dann gehe ein Feld nach rechts und gehe in den Zustand 2 über.

Befindet sich die Maschine im Zustand 2, so besagt die Anweisung im Anweisungsbuch:
Wenn das Feld, über dem der Lesekopf steht, leer ist, dann stoppe und gehe in Zustand 1 über. Steht aber eine „1“ auf dem Feld, dann lösche sie und bleibe in Zustand 2.

Die Turingmaschine mit diesem Anweisungsbuch ist eine so genannte Nachfolgemaschine, d. h. sie berechnet den Nachfolger $(n+1)$ jeder Zahl n .

Turingmaschine

Die Turingmaschine beginnt im Zustand 1 auf dem ersten leeren Feld rechts neben einer Reihe von Einsen (eine Zahl im Unärcode).

Der Nachfolger dieser Zahl ist zu finden!

Die Maschine druckt dann in dieses erste leere Feld rechts neben der Einserreihe eine 1.

Dann geht sie, immer noch im Zustand 1, ein Feld nach rechts und wechselt in den Zustand 2.
Findet sie das Feld, über dem der Lese-Schreibkopf nun steht, leer vor, so bleibt sie stehen und geht in den Zustand 1 über.

Damit ist die Aufgabe erledigt; die Kette der Einsen ist um eine 1 verlängert.
Das ist die Darstellung des Nachfolgers der gegebenen Zahl im Unärcode.

Findet die Maschine dagegen eine 1 auf dem betreffenden Feld vor, so löscht sie diese zuerst und bleibt dann stehen.

1936: Turings Maschine: die Turingmaschine

Turing wollte einen präzisen Begriff der Berechenbarkeit definieren.

Dazu entwickelte er die Idee eines abstrakten Automaten.

- Dieser verfügt über endlich viele interne Zustände, die sich in endlicher Weise beschreiben lassen.
- Sein Speicher ist ein in beiden Richtungen unendlich langer Papierstreifen, der in nebeneinanderliegende Felder unterteilt; jedes Feld kann ein einzelnes Zeichen enthalten.

Turingmaschine

Trotz dieser sehr einfachen Einzelschritte ist es möglich, weit komplexere Formen von Rechenvorschriften in eine Turingmaschine zu „übersetzen“.

Selbst die umfangreichsten Programme heutiger Programmiersprachen können letztendlich als Turingmaschinen aufgefasst werden,

Allerdings sind deren Anweisungsbücher sehr, sehr dick!!!

Beliebige algorithmische Verfahren können auf Turingmaschinen übertragen werden!

Diese Behauptung entspricht der Church-These.

Diese These ist nicht beweisbar!

1936: Turings Maschine: die Turingmaschine

Die Turingmaschine hat einen Schreib-Lese-Kopf, der pro Arbeitsgang genau ein Feld auf dem Streifen bearbeiten kann; sie kann:

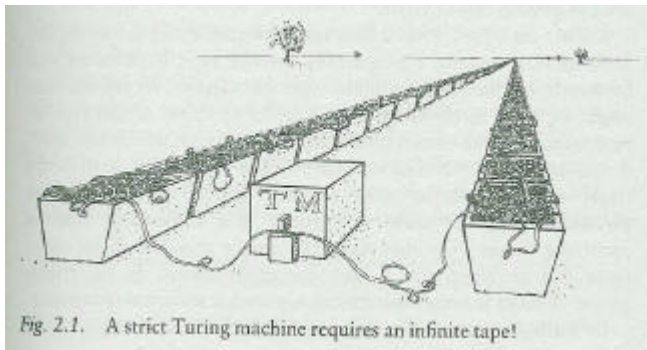
- ein Zeichen vom Streifen lesen,
- ein Zeichen vom Streifen löschen und evtl. ein anderes an dessen Stelle schreiben
- den Streifen um ein Feld nach links oder nach rechts bewegen.

Bei jedem Arbeitsgang wird der Zustand der Maschine geändert.

Der Inhalt des Streifens kann als Eingabe verstanden werden.

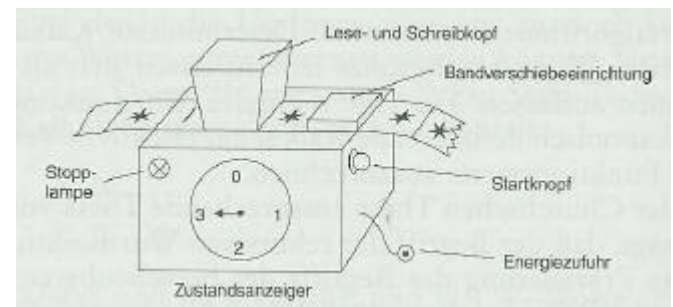
Der Zustand der Maschine kann, wenn die Maschine angehalten hat, als Ausgabe verstanden werden.

Turingmaschine, illustriert von Roger Penrose



Roger Penrose: The Emperors New Mind, Oxford University Press 1989.

Turingmaschine, illustriert von Klaus Mainzer

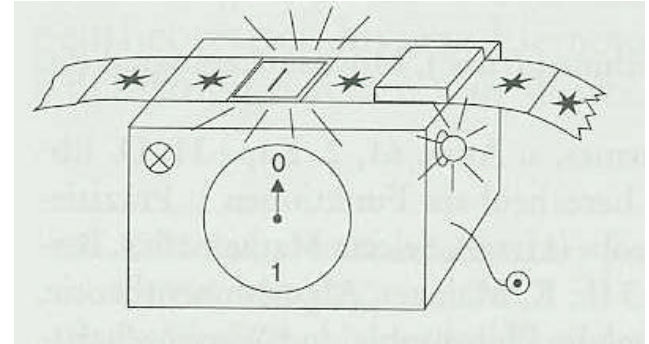


Klaus Mainzer: Computer – Neue Flügel des Geistes, de Gruyter: Berlin 1995.

Turingmaschine

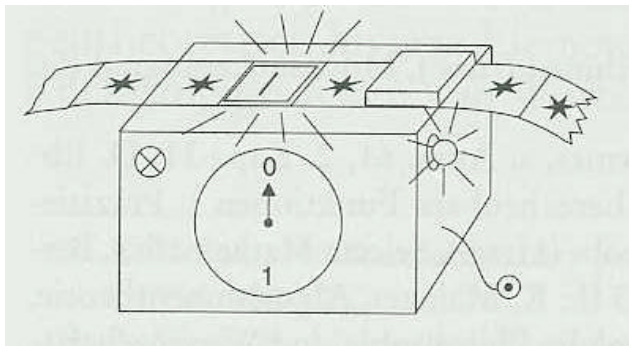
- Ein Operationswerk mit endlich vielen diskreten Zuständen 0 bis $m \geq 0$,
- ein kombinierter Lese-Schreibkopf,
- ein beiderseitig (potentiell) unbegrenztes Rechenband mit linearer Feldunterteilung und
- eine Verschiebeeinrichtung für das Band.
- 0 heißt der Anfangszustand der Turingmaschine,
- der Lese-Schreibkopf steht jeweils über einem Feld des Bandes, dem jeweiligen Arbeitsfeld.
- Mittels der Verschiebeeinrichtung kann das Arbeitsfeld um ein Feld nach rechts (r) oder links (l) verschoben werden.
- Der Lese-Schreibkopf kann die Zeichen lesen, löschen und drucken.
- Eine Lampe leuchtet auf, wenn die Turingmaschine stoppt (s).

Über dem Alphabet $\{*,i\}$ wird z. B. die Nachfolgerkonstruktion, die jeder Strichfolgen einen Strich anhängt, durch folgende Maschine T1 realisiert:



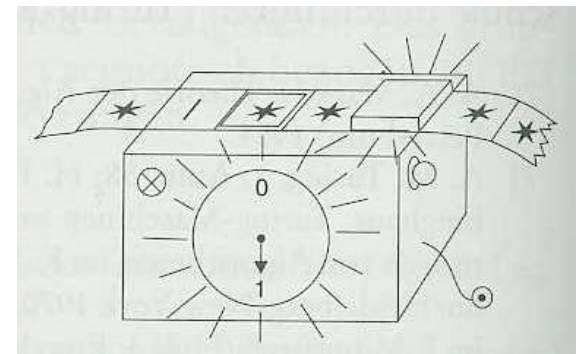
Klaus Mainzer: Computer – Neue Flügel des Geistes, de Gruyter: Berlin 1995.

Steht im Anfangszustand 0 im Arbeitsfeld ein Strich, dann rückt das Arbeitsfeld ein Feld nach rechts (r) und T1 geht in den Zustand 1 über.



Klaus Mainzer: Computer – Neue Flügel des Geistes, de Gruyter: Berlin 1995.

Arbeitsfeld ein Feld nach rechts (r) und T1 ist im Zustand 1.



Klaus Mainzer: Computer – Neue Flügel des Geistes, de Gruyter: Berlin 1995.

$$T_1 = \begin{pmatrix} 0 & | & r & 1 \\ 0 & * & * & 0 \\ 1 & | & s & 1 \\ 1 & * & | & 1 \end{pmatrix} \quad \begin{matrix} \text{(Abb. 1.21)} \\ \text{(Abb. 1.23)} \\ \text{(Abb. 1.22)} \end{matrix}$$

Turingmaschine

0	a_0	v_{00}	c_{00}
:	:	:	:
0	a_N	v_{0N}	c_{0N}
1	a_0	v_{10}	c_{10}
:	:	:	:
1	a_N	v_{1N}	c_{1N}
		:	
m	a_0	v_{m0}	c_{m0}
:	:	:	:
m	a_N	v_{mN}	c_{mN}

Formale Definition einer Turingmaschine

Eine Turingmaschine M ist ein 6-Tupel $\langle Q, A, f, q_0, \#, F \rangle$ wobei gilt:

- Q ist die Menge der Zustände von M ;
- A ist eine endliche Menge von Symbolen, das sogenannte Eingabealphabet;
- F ist eine partielle Übergangsfunktion;
 $\#$ ist ein spezielles Symbol, genannt „Leerstelle“;
- q_0 ist ein bestimmter Zustand aus A , der Anfangszustand;
- F ist eine Teilmenge von Q , die Menge der Endzustände.

Universelle Turingmaschine

- Das Anweisungsbuch muss nicht in der Turingmaschine selbst aufbewahrt werden,
- seine Einträge (Anweisungen) können auch im Speicherband (Gleisabschnitt) stehen.

Dazu müssen die Anweisungen nur als Folge von Symbolen dargestellt und in den Gleisfeldern notiert werden.

Die Draisine selbst erhält dann nur ein fixes Anweisungsbuch, in dem steht: Führe die Anweisungen aus, die in den jeweiligen Gleisfeldern stehen!

„Universelle Turingmaschine“: beliebige Algorithmen sind realisierbar

Turingmaschine - Konsequenzen

- Ein Problem ist algorithmisch *nicht* lösbar, wenn *keine* Turingmaschine die Lösung schaffen kann.
- Probleme dieser Art gibt es sehr , sehr viele!
- Die interessanten Fragen über Programme sind algorithmisch *nicht* lösbar!
- Z. B.: Es gibt *keinen* Algorithmus, der für ein vorgelegtes Programm entscheidet, ob es für jede Eingabe stoppt und eine Ausgabe liefert. (Halteproblem)

Fuzzy-Turingmaschine

Fuzzy-Turingmaschinen unterscheiden sich von Turings Maschinen vor allem durch die Art der Abhängigkeiten des (Fuzzy-)Zustandes zur Zeit $n+1$ sowohl vom (Fuzzy-)Zustand zur Zeit n als auch vom (Fuzzy-)Input zur Zeit n .

Zadeh betrachtete eine Turingmaschine mit $Q = \{q_0, q_1, \dots, q_n\}$ als Menge aller ihrer Zustände und $U = \{u_0, u_1, \dots, u_m\}$ als Menge ihrer Bandsymbole.

- Im Falle einer „nonfuzzy -deterministischen“ Turing Maschine ist der Zustand zur Zeit $n+1$ eine Funktion des Zustandes zur Zeit n und des Bandsymbols zur Zeit n , es gilt also:

$$q_{n+1} = f(q_n, u_n),$$

wobei f eine Funktion von $Q \times U$ nach Q ist, und q_n und u_n Variablen über Q bzw. U sind

Fuzzy-Turingmaschine

- Im Falle einer „nonfuzzy -nichtdeterministischen“ Turingmaschine Falle ist f eher eine mehrwertige als eine einwertige Funktion. Die Abhängigkeit kann daher auch eher durch eine Relation beschrieben werden:

$$R = \{(q_{n+1}, q_n, u_n)\}$$

wobei R eine Teilmenge des Produktraumes $Q \times Q \times U$ ist.

Im Falle einer Fuzzy -Turingmaschine ist die obige Relation ein Fuzzy Set des Produktraumes $Q \times Q \times U$, und durch die Zugehörigkeitsfunktion $\mu_R(q_{n+1}, q_n, u_n)$ charakterisiert, die jedem Tripel (q_{n+1}, q_n, u_n) einen Zugehörigkeitsgrad zur Relation R zuordnet.

Lotfi A. Zadeh, 1973: Fuzzy Algorithms

Beispiele:

- Cooking recipes (Kochrezepte),
- Directions for repairing a TV set, (Reparatur eines Fernsehgerätes)
- Instructions on how to treat a disease, (Krankheitsbehandlung)
- Instructions for parking a car (Einparken eines Autos)



Lotfi A. Zadeh, 1973: Fuzzy Algorithms

1) Anweisungen:

$$x \approx 5$$

$$x = \text{small}$$

$$x \text{ is small}$$

$$x \text{ is not large and not very small}$$

2) Fuzzy Bedingte Aussagen:

If x is small then y is large

else y is not large.

If x is positive then decrease y slightly.

If x is much greater than 5 then stop.

If x is very small then go to 7.

3) Unbedingte Befehle:

multiply x by y

decrease x slightly

delete the first few occurrences of 1

go to 7

print x

stop.

Lotfi A. Zadeh, 1973: Fuzzy Algorithms

Fuzzy Definitional Algorithms:

(definierend, kategorisierend) kategorisieren einen *fuzzy* Input

Fuzzy Generational Algorithms:

(erzeugend) erzeugen einen *fuzzy* Output

Fuzzy Relational Algorithms:

(beschreibend) beschreiben ein System im Fließzustand

Fuzzy Decisional Algorithms:

(entscheidend) bringen Befehle aufgrund fortlaufenden Feedbacks hervor

Fuzzy Definitional Algorithms

Beispiel: zum Begriff „oval“

T sei ein zu testendes Symbol;

CALL CONVEX ruft einen Unteralgorithmus CONVEX auf, der testet, ob T convex ist, oder nicht;

IF A THEN B wird interpretiert als: **IF A THEN B** oder gehe zum nächsten Befehl

Fuzzy Definitional Algorithms

Fuzzy-Algorithmus OVAL

- 1) **IF** T is not closed **THEN** T is not *oval*; stop.
- 2) **IF** T is self-intersecting **THEN** T is not *oval*; stop.
- 3) **IF** T is not **CALL CONVEX** **THEN** T is not *oval*; stop.
- 4) **IF** T does not have two *more or less* orthogonal axes of symmetry **THEN** T is not *oval*; stop.
- 5) **IF** the major axis of T is not *much* longer than the minor axis **THEN** T is not *oval*; stop.
- 6) T is *oval*; stop.

Fuzzy Definitional Algorithms

Unteralgorithmus CONVEX

- 1) $x = a$ (some initial point on T).
- 2) Choose a direction of movement along T .
- 3) $t \approx$ direction of tangent to T at x .
- 4) $x' \approx x+1$ (move from x to a neighboring point).
- 5) $t' \approx$ direction of tangent to T at x' .
- 6) $a \approx$ angle between t' and t .
- 7) $x \approx x'$.
- 8) $t \approx$ direction of tangent to T at x .
- 9) $x' \approx x+1$
- 10) $t' \approx$ direction of tangent to T at x' .
- 11) $b \approx$ angle between t' and t .
- 12) **IF** b does not have the same sign as a **THEN** T is not convex; return
- 13) Go to 7).

Fuzzy Generational Algorithms schaffen etwas, anstatt zu definieren!

Fuzzy Generational Algorithms

Beispiele:

- Produktion von Handschriften, Kochrezepten, Musik, natürlicher Sprache, Laute

- Schreiben eines großen „P“

- Herstellen von Schokoladen-karamel

- 1) $i = 1$.
- 2) $X(i) = b$ (first dot at base).
- 3) $X(i+1) \approx X(i) + h/6$
(put dot approximately $h/6$ units of distance above $X(i)$).
- 4) $i = i + 1$.
- 5) IF $i = 7$ THEN make right turn and go to 7).
- 6) Go to 3).
- 7) Move by $h/6$ units; put a dot.
- 8) Turn by 45° ; move by $h/6$ units; put a dot.
- 9) Turn by 45° ; move by $h/6$ units; put a dot.
- 10) Turn by 45° ; move by $h/6$ units; put a dot.
- 11) Turn by 45° ; move by $h/6$ units; put a dot; stop.

Fuzzy Relational Algorithms

- beschreiben Relationen zwischen Fuzzy-Variablen
- Spezielle Fuzzy Relational-Algorithms, die das Verhalten eines Systems beschreiben, heißen *Fuzzy Behavioral Algorithms*.

Beispiel: Algorithmus R(x,y,z):

- 1) IF x is *small* and y is *large* THEN z is *very small* ELSE z is *not small*.
- 2) IF x is *large* THEN (IF y is *small* THEN z is *very large* ELSE z is *small*) ELSE z and y are *very small*.

Fuzzy Behavioral Algorithms

Beispiel:

S sei ein Fuzzy-System, dessen mögliche

(nicht fuzzy) Zustände (states) seien:

q_1 und q_2 .

Es gebe zwei fuzzy inputs:

low und *high*.

Es gebe zwei fuzzy outputs:

large und *small*.

u_t sei input zur Zeit t .

y_t sei output zur Zeit t .

x_t sei state zur Zeit t .

$u_t \backslash x_t$	x_{t+1}		y_{t+1}	
	q_1	q_2	q_1	q_2
<i>low</i>	q_2	q_1	<i>large</i>	<i>small</i>
<i>high</i>	q_1	q_1	<i>small</i>	<i>large</i>

Fuzzy Behavioral Algorithms

Beispiel: Ballon

Variablen: Luftmenge im Ballon (*gegenwärtiger Zustand, state*)

Menge, um die die Luft im Ballon wächst (Vorgang)

Oberflächenspannung des Ballons (Ergebnis)

Fuzzy-Algorithmus:

WENN *wenig* Luft im Ballon ist und nur *wenig* hinzugepumpt wird, DANN wird die Oberflächenspannung nur *wenig* zunehmen.

WENN *wenig* Luft im Ballon ist und *viel* hinzugepumpt wird, DANN wird die Oberflächenspannung *stark* zunehmen.

WENN *viel* Luft im Ballon ist und *wenig* hinzugepumpt wird, DANN wird die Oberflächenspannung *mäßig* zunehmen.

WENN *viel* Luft im Ballon ist und *viel* hinzugepumpt wird, DANN wird die Oberflächenspannung nur *sehr stark* zunehmen.



Fuzzy Decisional Algorithms

Beispiele:

- Parken eines Autos,
- Überwinden einer Kreuzung,
- Transportieren eines Objekts,
- Hauskauf.

Algorithmus INTERSECTION:

- 1) **If** signal lights **then** *SIGNAL*
else if stop sign **then** call *SIGN*
else if blinking light **then** call *BLINKING*
else call *UNCONTROLLED*.

Fuzzy Decisional Algorithm

Unteralgorithmus *SIGN*:

- 1) **IF** no stop sign on your side
THEN IF no cars in the intersection
THEN cross at *normal* speed
ELSE wait for cars to leave the intersection and then cross.
- 2) **IF** not *close* to intersection
THEN continue approaching at normal speed for a *few* seconds; go to 2).
- 3) *Slow down*.

Fuzzy Decisional Algorithm

- 4) **IF** in a *great* hurry and no police cars in sight and no cars in the intersection or its *vicinity*
THEN cross the intersection at *slow* speed.
- 5) **IF** *very close* to intersection **THEN** stop; go to 7).
- 6) Continue *approaching* at *very slow* speed; go to 5).
- 7) **IF** no cars *approaching* or in the intersection
THEN cross.
- 8) Wait a *few* seconds; go to 7).

Fuzzy Decisional Algorithms

Beispiel: Transfer eines Subjekts mit verbundenen Augen

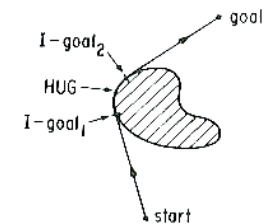


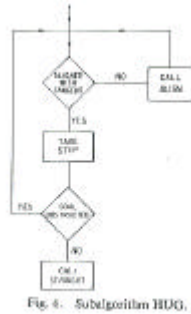
Fig. 4. Problem of transferring blindfolded subject from *start* to *goal*.

Fuzzy Decisional Algorithms

line g has by 30
 line g little g has by 30
 line very little g has by 30
 line very very little g
 $A = g$ close to 50°
 $B = g$ close to 0°
 $C = g$ very close to 0°

Fig. 5. Subalgorithm ALIGN.

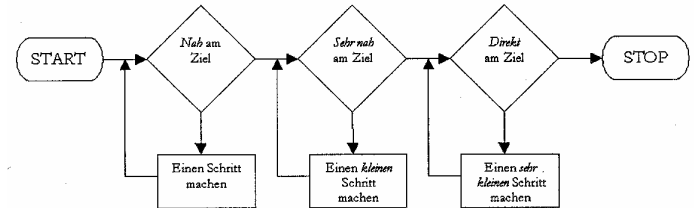
Fig. 6. Subalgorithm HUG.



Fuzzy Decisional Algorithm

```
graph LR; START([START]) --> D1{Nah am Ziel}; D1 --> A1[Einen Schritt machen]; A1 --> D1; D1 --> D2{Sehr nah am Ziel}; D2 --> A2[Einen kleinen Schritt machen]; A2 --> D2; D2 --> D3{Direkt am Ziel}; D3 --> A3[Einen sehr kleinen Schritt machen]; A3 --> D3; D3 --> STOP([STOP]);
```

The flowchart illustrates a fuzzy decisional algorithm for reaching a goal. It begins with a 'START' terminal, leading to a decision diamond 'Nah am Ziel' (Near the goal). If the condition is not met, it leads to 'Einen Schritt machen' (Make a step) and loops back. If met, it leads to the next decision diamond 'Sehr nah am Ziel' (Very near the goal). This pattern repeats for 'Sehr nah am Ziel' leading to 'Einen kleinen Schritt machen' (Make a small step) and 'Direkt am Ziel' (Directly at the goal) leading to 'Einen sehr kleinen Schritt machen' (Make a very small step). The final decision diamond 'Direkt am Ziel' leads to a 'STOP' terminal.



The diagram on the left shows a shaded, irregular shape representing a blindfolded subject. Four arrows point towards it from different directions, labeled: 'goal' (top right), 'I-goal₂' (top left), 'HUG' (middle left), and 'I-goal₁' (bottom left). An arrow labeled 'start' points away from the bottom of the shape.

The flowchart on the right, titled 'Subalgorithm STRAIGHT', is a vertical sequence of decision diamonds and action rectangles. It begins with an oval 'START' node. The first decision is 'TOO FAR GOAL'. If 'YES', it proceeds to the next decision; if 'NO', it goes to a rectangle 'TAKE STEP'. The second decision is 'VERY CLOSE'. If 'YES', it proceeds; if 'NO', it goes to 'TAKE SMALL STEP'. The third decision is 'VERY+ CLOSE'. If 'YES', it proceeds to the final oval 'STOP' node; if 'NO', it goes to 'TAKE VERY+ SMALL STEP'.

```

graph TD
    START([START]) --> D1{TOO FAR GOAL}
    D1 -- YES --> D2{VERY CLOSE}
    D1 -- NO --> A1[TAKE STEP]
    A1 --> D2
    D2 -- YES --> D3{VERY+ CLOSE}
    D2 -- NO --> A2[TAKE SMALL STEP]
    A2 --> D3
    D3 -- YES --> STOP([STOP])
    D3 -- NO --> A3[TAKE VERY+ SMALL STEP]
    A3 --> D3
  
```

Fig. 7. Subalgorithm STRAIGHT.

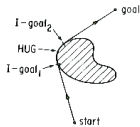
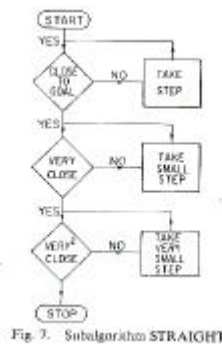


Fig. 4. Problem of transferring blindfolded subject from *start* to *goal*.



Alan M. Turing: *The State of the Art*
(Lecture to the London
Mathematical Society, 20th
February 1947)

Alan Mathison Turing (1912 - 1954)

Aus der Sicht des Mathematikers ist die Eigenschaft der Digitalität von größerem Interesse als das Elektronische. Das sie elektronisch ist, ist zweifellos wichtig, weil diese Maschinen dem ihre große Schnelligkeit verdanken, und ohne ihre Schnelligkeit wäre es zweifelhaft, ob ihre Konstruktion finanzielle Unterstützung erfahren würde.

Aber das ist nahezu alles, was zu diesem Thema zu sagen ist.

Turingmaschine

Als *Rechnen* gelten seit Turing Prozesse, wie sie in einer Turingmaschine ausgeführt werden,

als *berechenbar* gilt, was in einer Turingmaschine berechnet werden kann.



Zwischen Eifa und Alsfeld liegen wieder Schienen ...



Zwar ist die Strecke eigentlich nicht mehr befahrbar, dennoch findet sich hier dieses interessante Gefährt.



David Hilbert (1862-1943)

Zu Beginn des 19. Jahrhunderts: „Hilbert-Programm“:

- Zug um Zug sollten alle mathematischen Disziplinen *widerspruchsfrei* nachgewiesen werden.
- Axiomatische Theorien, d. h. als Sammlung von wahren Aussagen
 - in einer formalen Sprache
 - klar definiert
- Schlussregeln, mit denen aus diesen Axiomen Folgerungen gezogen werden können



Hilbert-Programm

- Hilbert nahm an, dass die Prädikatenlogik 1. Stufe eine geeignete Sprache sei.
- Hilbert nahm an, dass alle mathematischen Folgerungen in einem Logikkalkül formuliert werden können, der der axiomatisierten mathematischen Theorie zugrunde liegt.

Das war plausibel, weil Mathematiker seit Jahrhunderten intuitiv so vorgehen.

Beide Annahmen sind aber keine mathematischen Thesen und somit sind *nicht beweisbar!*

Kurt Gödel (1906-1978): Unentscheidbare Systeme

Erschütterung des Hilbert-Programms:

1931:

Kurt Gödel zeigte, dass jedes hinreichend umfassende mathematische System Sätze enthält, die weder bewiesen, noch widerlegt werden können.



Berechenbarkeit

- Entscheidungsverfahren
- Algorithmus (effektive Methode zur Lösung eines gegebenen Problems)

In Metamathematik und Logik:

Eine vollständige und eindeutige Berechnungsvorschrift, die bei richtiger Anwendung nach endlich vielen Schritten mit logischer Notwendigkeit zur richtigen Antwort (und niemals zu einer falschen Antwort) führt.



Was sind Berechnungen?

Was ist eine Berechnungsvorschrift?

- intuitiv: eine Anweisung, Berechnungen durchzuführen

Was sind Berechnungen?

Hier gerät man in den Grenzbereich zwischen Mathematik und Psychologie!

Die Mathematik

- kann die Fähigkeit des menschlichen Verstandes zu rechnen an Beispielen illustrieren,
- sie kann aber keine Definition dafür geben!

Alonzo Church (1903-1995)

1935: An unsolvable problem of elementary number theory.
The American Journal of Mathematics 58, 345-363.

Nicht jede Frage nach effektiver Berechenbarkeit ist lösbar!



effektiv berechenbare Funktion in den positiven ganzen Zahlen

=

rekursive Funktion in den positiven ganzen Zahlen
(oder λ -definierbare Funktion in den positiven ganzen Zahlen).

Kurt Gödel „rekursive Funktionen“

Nachfolgerfunktion $f_1(x) = x + 1$

Summenfunktion $f_2(x, y) = x + y$

Produktfunktion $f_3(x, y) = x \cdot y$

Potenzfunktion $f_4(x, y) = x^y$

Arithmetische Differenz $f_5(x, y) = \begin{cases} x - y & \text{falls } x > y \\ 0 & \text{andernfalls} \end{cases}$

Rekursive Funktionen

Aus gegebenen Funktionen können mit Hilfe zweier Operationen neue Funktionen generiert werden:

- Superposition von Funktionen

(Funktionen können als Argument in andere Funktionen eingesetzt werden)

- μ -Operation

(Eine Funktion, die es erlaubt, unter bestimmten Umständen aus einer gegebenen Funktion eine neue zu bilden, die minimale Nullstellen der Ausgangsfunktion findet.)

D. h.: $f(x_1, \dots, x_n, y)$ mit $\forall x_1, \dots, x_n > \in \mathbb{N} \exists y \in \mathbb{N} : f(x_1, \dots, x_n, y) = 0$

Führt zu einer Funktion $g(x_1, \dots, x_n) = z \Leftrightarrow_{df.}$

z ist die kleinste natürliche Zahl, für die gilt: $f(x_1, \dots, x_n, z) = 0$

Rekursive Funktionen

Die Gesamtheit der aus den fünf vorher genannten Grundfunktionen durch sukzessive Anwendung der beiden genannten Operationen erzeugbaren Funktionen ist die Klasse der *rekursiven* Funktionen.

Eine Funktion ist also genau dann rekursiv, wenn sie in endlich vielen Schritten durch Superposition und μ -Operation aus den fünf Grundfunktionen konstruiert werden kann.

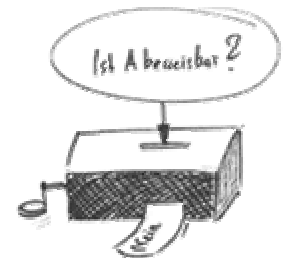
Alonzo Church (1903-1995)

Könnte ein Computer vielleicht auch entscheiden, ob eine vorgelegte Aussage A einen Beweis besitzt?

Alonzo Churchs Antwort auf diese Frage: Nein.

Begründung:

Natürlich kann A wahr sein und einen Beweis besitzen, den jemand findet. Hingegen kann grundsätzlich kein Computer existieren, der zutreffend entscheidet, ob mathematische Aussagen, die man ihm eingibt, beweisbar sind.



Alan Mathison Turing (1912 - 1954)

- englischer Mathematiker und Logiker
- studierte 1932 - 1935, Mathematik und Logik
- begründete die logische Theorie einer universellen abstrakten Rechenmaschine, die nach ihm benannte *Turing-Maschine*.
- ab 1945 ist Turing an der Entwicklung des Collossus (automatische Großrechenmaschine) beteiligt.
- Er beschäftigte sich bereits 1947 mit dem Problem, ob und wie Rechenmaschinen lernen können.
- 1950 entwickelte er den nach ihm benannten *Turing-Test*.
- ab 1950 widmete sich Turing kybernetischen Fragen der Biologie.
- Am 7. Juni 1954 starb Turing durch Zyanid.

