Informatik-Propädeutikum

Dozentin: Dr. Claudia Ermel

Betreuer: Sepp Hartung, André Nichterlein, Clemens Hoffmann

Sekretariat: Christlinde Thielcke (TEL 509b)

TU Berlin
Institut für Softwaretechnik und Theoretische Informatik
Prof. Niedermeier
Fachgruppe Algorithmik und Komplexitätstheorie
http://www.akt.tu-berlin.de

Wintersemester 2013/2014

Gliederung

Grenzen der Berechenbarkeit

Alan Mathison Turing, 1912–1954

Das Postsche Korrespondenzproblem I

Betrachte folgendes einfaches „Dominospiel“:

Problem: Gegeben ist eine Menge von Paaren (x_i, y_i) bestehend aus je zwei Binärzeichenketten.

Frage: Gibt es eine Aneinanderreihung der Paare, wobei diese beliebig oft wiederholt werden dürfen, sodass das durch die „x-Teile“ gebildete Wort demjenigen der „y-Teile“ entspricht?

Emil Leon Post, 1897–1954

Beispiel 1:

$(x_1, y_1) = (1, 101)$, $(x_2, y_2) = (10, 00)$, $(x_3, y_3) = (011, 11)$.

Dann gilt:

$$x_1; x_3; x_2; x_3 = \langle 1; 011; 10; 011 \rangle = \langle 101110011 \rangle$$

$$y_1; y_3; y_2; y_3 = \langle 101; 11; 00; 11 \rangle = \langle 101110011 \rangle$$

Das Postsche Korrespondenzproblem II

Beispiel 2:

$(x_1, y_1) = (10, 101)$, $(x_2, y_2) = (011, 11)$, $(x_3, y_3) = (101, 011)$.

Es gibt keine Lösung. Klar: Man muss mit erstem Tupel anfangen:

$$\begin{array}{l} \langle 10... \rangle \\ \langle 101... \rangle \end{array} \tag{1}$$

Dann muss das „x-Wort“ mit 1 fortgesetzt werden, also ist das erste oder dritte Tupel zu benutzen.
Das erste Tupel liefert $\langle 1010... \rangle$ und $\langle 101101... \rangle$ $\lightning$.

Also betrachte drittes Tupel:

$$\begin{array}{l} \langle 10101... \rangle \\ \langle 101011... \rangle \end{array}$$

Das x-Wort muss wieder mit 1 fortgesetzt werden, analog zu (1)
$\rightsquigarrow$ drittes Tupel wieder nehmen $\rightsquigarrow$ Endlosschleife.

Knobelaufgabe zum Postschen Korrespondenzproblem

Hat folgendes Postsche Korrespondenzproblem (PCP) eine Lösung? Welche?
$(x_1, y_1) = (1, 1111)$, $(x_2, y_2) = (111, 1)$,
$(x_3, y_3) = (111, 1111111111111111)$.
Eine Lösung: $x_2 \circ x_2 \circ x_2 \circ x_1 \circ x_1 = 11111111111 = y_2 \circ y_2 \circ y_2 \circ y_1 \circ y_1$

Bemerkung: Die kürzeste Lösung zu folgendem PCP hat eine Länge von 252 aneinandergereihten Paaren.

$(1101, 1)$, $(0110, 11)$, $(1, 110)$.

Finden Sie sie?

Allgemeine Frage:
Gibt es einen Algorithmus, der bei Eingabe beliebiger Tupel immer entscheiden kann, ob das entsprechende PCP eine Lösung hat?

Was ist berechenbar (und was nicht)?

Aus den Vorbetrachtungen zum PCP ergeben sich folgende Fragen:

- Ist prinzipiell alles berechenbar (algorithmisch entscheidbar)? Oder gibt es interessante und natürliche Probleme bzw. Funktionen, die nicht berechenbar sind?
- Was heißt überhaupt berechenbar? Wie formalisiert man den Begriff „Algorithmus“?
- Gibt es verschiedene Modelle der Berechenbarkeit? Gibt es verschiedene Typen von Berechenbarkeit oder nur die *eine* (sinnvolle) Definition von Berechenbarkeit?

Zur Formalisierung des Berechenbarkeitsbegriffs I

Bereits im vorigen Kapitel über Rekursion erwähnte Hypothese:
Alle berechenbaren Funktionen (über den natürlichen Zahlen) sind mit μ-**Rekursion** (genauer: **partiell rekursiven Funktionen**) berechenbar.
$\rightsquigarrow$ Rechnen also als Einsetzen und Auswerten von mathematisch definierten Funktionen...

Jedoch ist μ-Rekursion ein sehr mathematisches, sehr an den natürlichen Zahlen „klebendes“ Berechnungsmodell...
Gibt es ein elementares, unmittelbar verständliches, sehr einfaches **Maschinenmodell** für Berechenbarkeit?

Alan Turing hat den Berechenbarkeitsbegriff mit den (heute) sogenannten **Turing-Maschinen** (1936) definiert, nach wie vor das wohl wichtigste Berechnungsmodell mit großer Relevanz bis hinein in die Komplexitätstheorie und Algorithmik.

Zur Formalisierung des Berechenbarkeitsbegriffs II

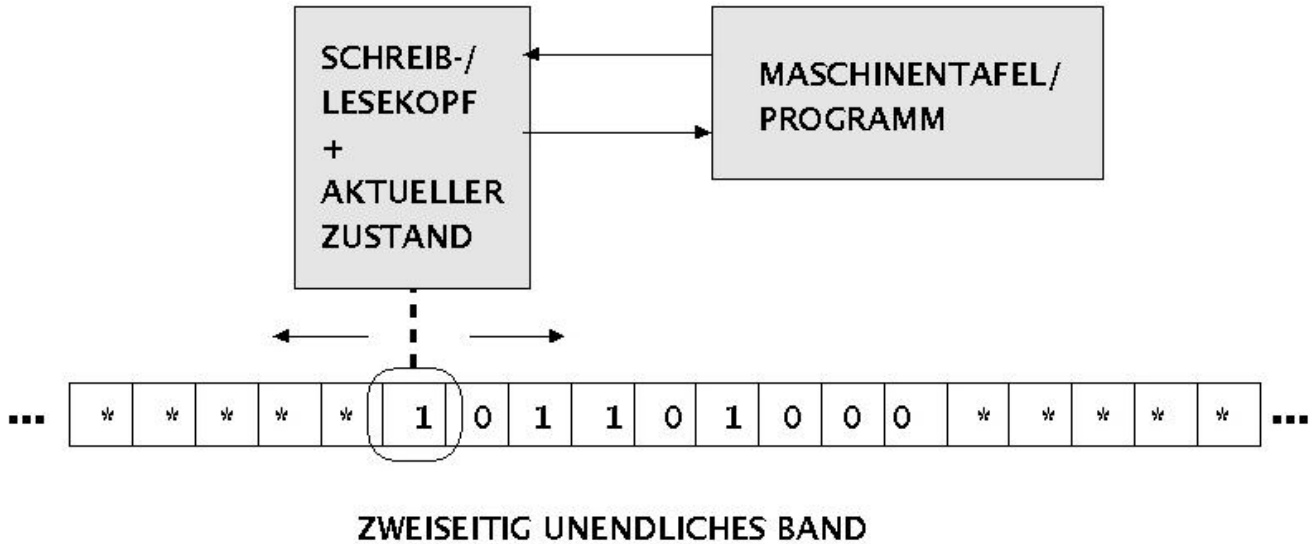
Die Turing-Maschine ist quasi ein zur Rekursion (und anderen Konzepten wie z.B. „λ-Kalkül“) konkurrierendes Modell für Berechenbarkeit. Sie modelliert die Arbeitsweise eines Computers auf besonders einfache und gut analysierbare (!) Weise.

So ergibt sich dann also die weitere Hypothese, dass alles was berechenbar ist, auch Turing-berechenbar ist; und somit insbesondere, dass die Berechnungsformalismen „partiell rekursive Funktionen“ und Turing-Maschinen und viele andere (in der Tat: alle bekannten) gleich mächtig sind $\rightsquigarrow$ **Church'sche These**.

Alonzo Church, 1903–1995

Es gilt insbesondere, dass alles was mit Registermaschinen (i.w. unseren Standardrechnern eins-zu-eins entsprechend) berechenbar ist, auch mit Turing-Maschinen berechenbar ist.

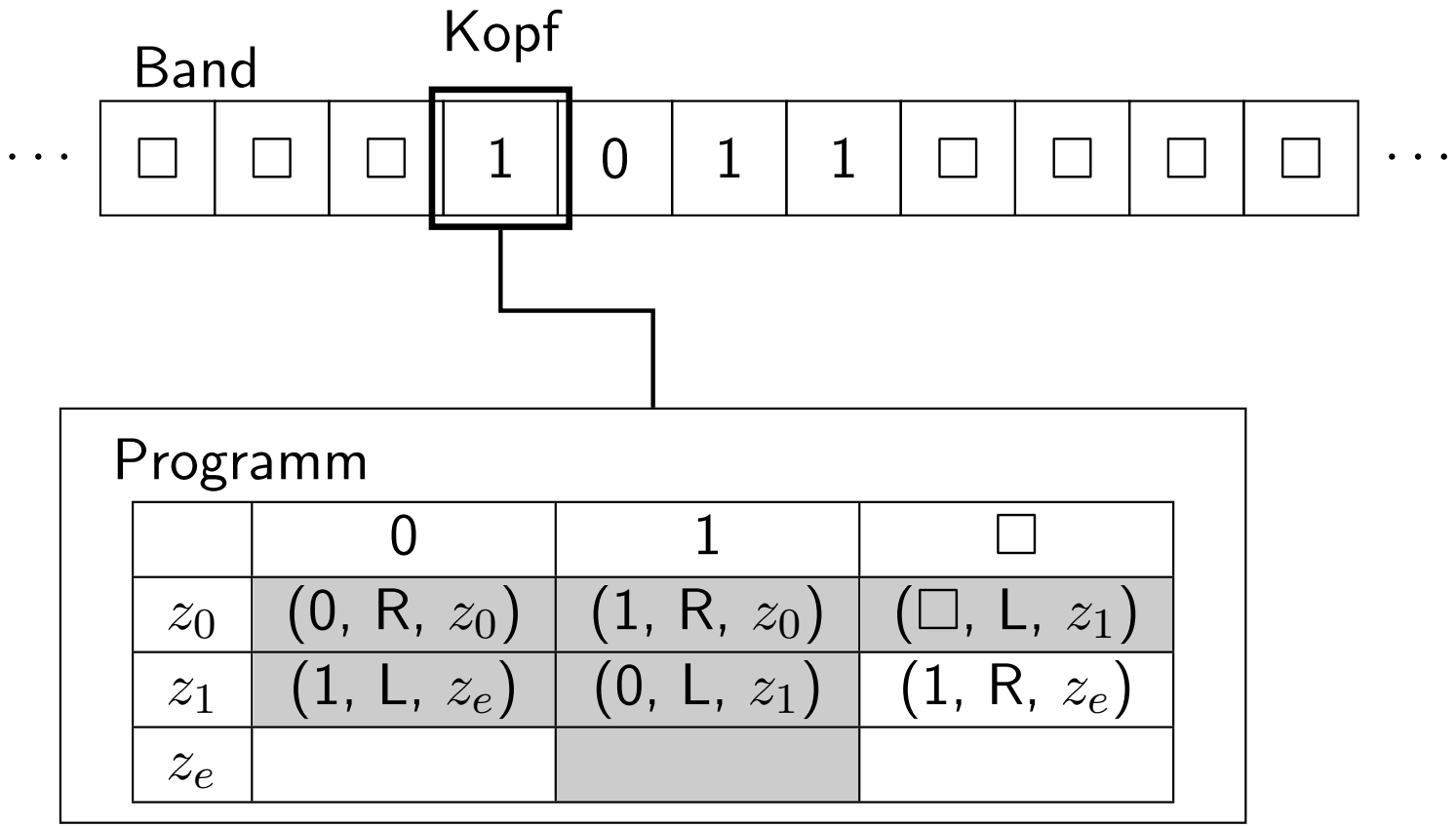
Die Turing-Maschine



Anschaulich gesprochen besteht eine **Turing-Maschine** aus

- einem (potenziell) unendlichen, in Felder unterteiltem **Band**, wobei jedes Feld ein Zeichen eines Alphabets (z.B. $\{0, 1\}$) speichern kann,
- einem **Schreib-Lese-Kopf**, der sich schrittweise über die Felder bewegt und jeweils ein Feld beschreiben oder lesen kann und
- einer **endlichen Kontrolleinheit**, die in jeweils genau einem von endlich vielen Zuständen ist.

Turingmaschine – Ein Rechenbeispiel



Programm

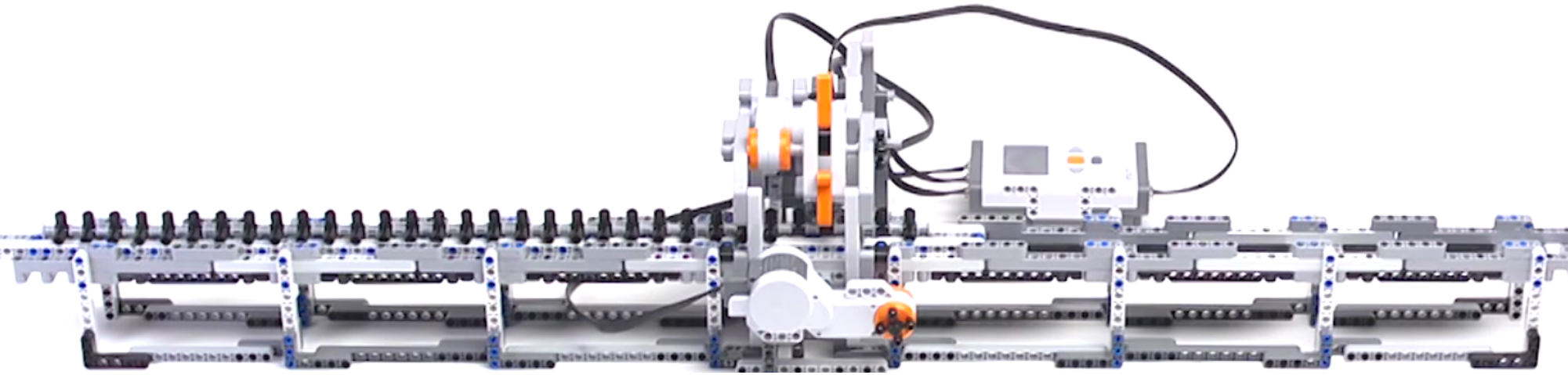
	0	1	□
z_0	(0, R, z_0)	(1, R, z_0)	(□, L, z_1)
z_1	(1, L, z_e)	(0, L, z_1)	(1, R, z_e)
z_e			

Die Turingmaschine hat drei Zustände z_0, z_1 und z_e.

- z_0: Bewege Kopf an das rechte Ende der Eingabebinärzeichenkette.
- z_1: „Flippe“ Bits bis zur ersten gelesenen 0.
- z_e: Endzustand.

Die Turingmaschine addiert eins zu einer gegebenen Binärzahl.

Turing-Maschinen sind real!



Video:

`http://www.elektronikpraxis.vogel.de/technologie-und-trends/articles/368631/`

Berechenbarkeit und Entscheidbarkeit

Wir halten fest: Eine Funktion heißt **berechenbar**, wenn es eine Turing-Maschine (also einen Algorithmus) gibt, die diese Funktion berechnet.

Eine berechenbare Funktion mit auf $\{0, 1\}$ eingeschränkten Wertebereich[4] wird **entscheidbar** genannt.

Weiterhin spricht man hier gleichbedeutend von **Entscheidungsproblemen**.

Die Church-Turing-These besagt, dass alle berechenbaren Entscheidungsprobleme durch eine Turing-Maschine entscheidbar sind.

[4] Solche Funktionen definieren 1:1 Sprachen: Genau diejenigen Elemente des Definitionsbereichs sind in der Sprache, die als Funktionswert 1 liefern.

Titelseite von Turings Aufsatz 1936

ON COMPUTABLE NUMBERS, WITH AN APPLICATION TO THE ENTSCHEIDUNGSPROBLEM

By A. M. Turing.

[Received 28 May, 1936.—Read 12 November, 1936.]

The "computable" numbers may be described briefly as the real numbers whose expressions as a decimal are calculable by finite means. Although the subject of this paper is ostensibly the computable *numbers*, it is almost equally easy to define and investigate computable functions of an integral variable or a real or computable variable, computable predicates, and so forth. The fundamental problems involved are, however, the same in each case, and I have chosen the computable numbers for explicit treatment as involving the least cumbrous technique. I hope shortly to give an account of the relations of the computable numbers, functions, and so forth to one another. This will include a development of the theory of functions of a real variable expressed in terms of computable numbers. According to my definition, a number is computable if its decimal can be written down by a machine.

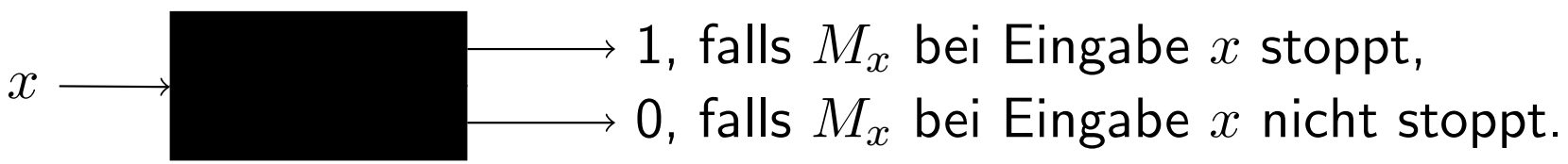
Einige Vorteile der Turing-Machine

- Prinzipiell nicht an Operationen auf natürlichen Zahlen gebunden.
- Maschinencharakter mit sehr intuitiver, leicht verständlicher Funktionsweise.
- Leichte Simulierbarkeit, insbesondere einer Turing-Maschine durch eine andere... $\rightsquigarrow$ Begriff der **universellen Turing-Maschine**.
- Elementar einfache, aber auch gut variierbare Definition.
- Leichte Variation des Berechnungsmodells ohne die Berechnungsmächtigkeit des Modells zu ändern.
- Große Relevanz in der Komplexitätstheorie.
- Kanonische Fortsetzung endlicher Automaten (und Kellerautomaten), die in der Theorie der formalen Sprachen und Grammatiken eine zentrale Rolle spielen.

Zur Unentscheidbarkeit des Halteproblems I

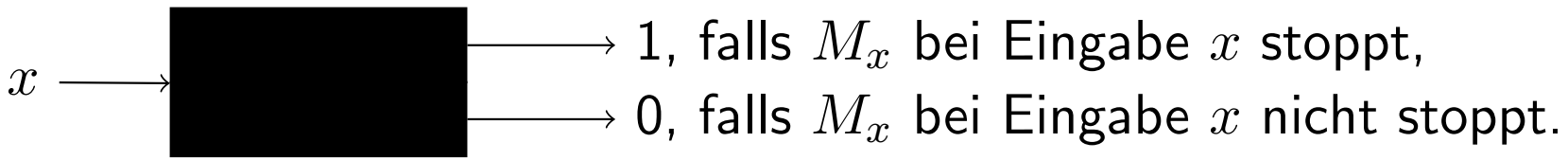
Eine zentrale Einsicht ist, dass jedes „Programm“ (sprich jede Turing-Maschine), das eine Zeichenkette $x \in \{0,1\}^*$ als Eingabe verarbeitet, sich selbst auch wieder als Zeichenkette aus $\{0,1\}^*$ „kodieren“ lässt. Damit kann man aber eine solche Turing-Maschine auf ihre eigene Kodierung als Eingabe anwenden! (Stichwort **Selbstanwendung**; vgl. Compiler.)

Nehmen wir also an, es gäbe eine Turing-Maschine M, die auf jeder Eingabe $x \in \{0,1\}^*$ berechnet, ob die durch x (eindeutig) kodierte Turing-Maschine M_x hält ($\rightsquigarrow$ Ausgabe 1) oder nicht ($\rightsquigarrow$ Ausgabe 0). Wir stellen uns M schematisch wie folgt vor:



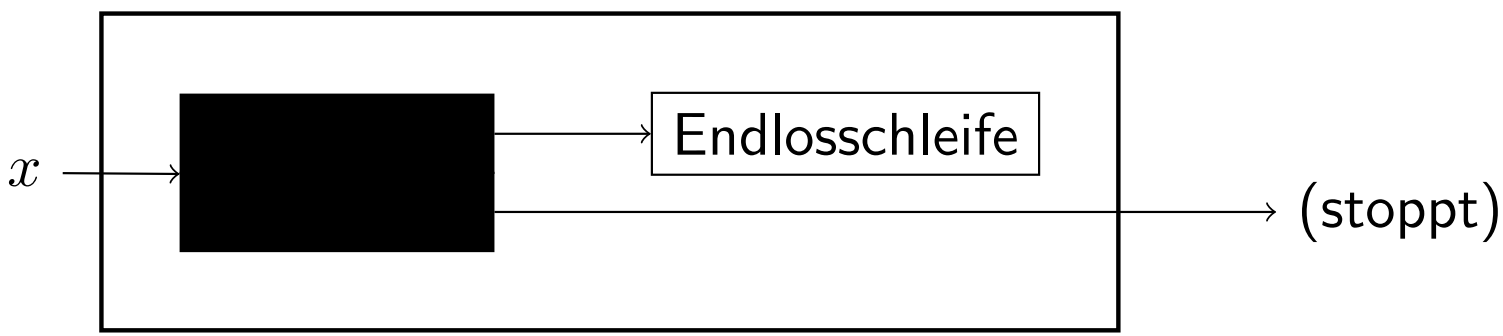
Zur Unentscheidbarkeit des Halteproblems II

Situation: M:



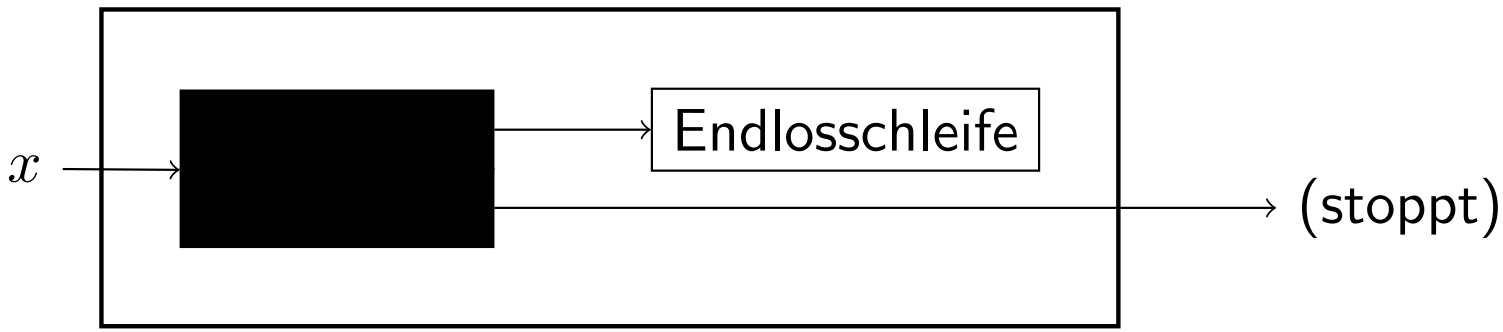
Nun baue mithilfe obiger Black Box eine neue Turing-Maschine M', die genau dann 0 ausgibt und stoppt, wenn die Black Box 0 ausgibt. Falls die Black Box 1 ausgibt, so gehe M' in eine Endlosschleife.

$\rightsquigarrow$ Bild für M':



Zur Unentscheidbarkeit des Halteproblems III

Situation: M':



Sei nun $z \in \{0,1\}^*$ die Kodierung der neuen Turing-Maschine M' als Binärzeichenkette. Was macht M' auf Eingabe $x = z$ (**Selbstanwendung!**)? Stoppt M', ja oder nein?

Beide Antworten führen zu einem logischen Widerspruch! Daraus folgt, dass die Ursprungsannahme, dass M existiere, falsch gewesen sein muss!

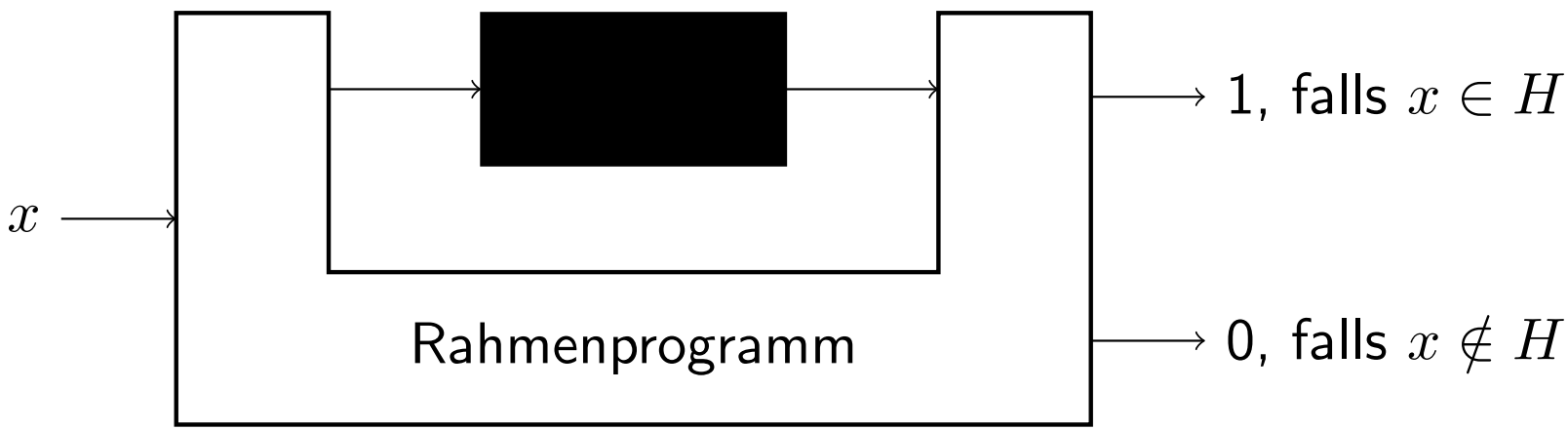
$\rightsquigarrow$ Das **(spezielle) Halteproblem** H ist also unentscheidbar (d.h. nicht berechenbar), wobei H die Menge aller Binärwörter x ist, so dass die durch x kodierte Turing-Maschine auf Eingabe x hält!

Weitere unentscheidbare Probleme I

Mithilfe des Halteproblems kann man eine Vielzahl anderer Probleme (Funktionen) als unentscheidbar (nicht berechenbar) nachweisen.

Die zentrale Idee hierbei beruht auf dem Konzept der **Reduktion**: Falls wir zeigen können, dass wir mithilfe eines Algorithmus A einen neuen Algorithmus B bauen könnten, der das Halteproblem H entscheidet, so muss daraus folgen, dass A nicht entscheidbar sein kann.

$\rightsquigarrow$ Bild (A entspricht der Black Box, B dem Gesamtbild):



Weitere unentscheidbare Probleme II

Mithilfe des Reduktionsprinzips lassen sich nun viele Probleme als unentscheidbar nachweisen:

- Postsches Korrespondenzproblem.
- Bestimme, ob ein gegegebenes Gleichungssystem (Polynome) eine ganzzahlige Lösung besitzt (Stichwort „Lösbarkeit Diophantischer Gleichungen“).
- Bestimme, ob eine gegebene Funktion bijektiv (oder surjektiv oder injektiv) ist.
- Bestimme, ob zwei Funktionen (Programme) äquivalent sind (d.h., beide Programme liefern für dieselbe Eingabe auch immer dieselbe Ausgabe).

Unentscheidbare Probleme III

Der **Satz von Rice** besagt, dass es unmöglich ist, irgendeinen nichttrivialen Aspekt des funktionalen Verhaltens einer Turingmaschine (oder eines Algorithmus in einem anderen Berechnungsmodell) algorithmisch zu entscheiden.

Es ist zum Beispiel unentscheidbar, ob ein Programm (eine Turingmaschine)

- irgendwo hält,
- überall hält,
- bei Eingabe 42 Ausgabe 42 produziert.

($\rightsquigarrow$ „Fast nichts" ist berechenbar!)

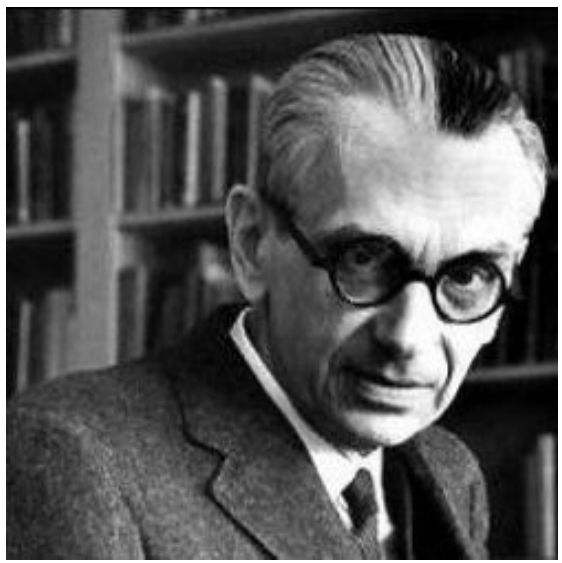
Kreative Analyse lässt sich also wohl nicht vollends automatisieren...

Ein Papst der Unentscheidbarkeit und Unvollständigkeit

Gödelscher Unvollständigkeitssatz (1931): Alle widerspruchsfreien axiomatischen Formulierungen der Zahlentheorie[5] enthalten unentscheidbare Aussagen.

Beweisidee: einen selbstbezüglichen mathematischen Satz formulieren

- Jede Aussage der Zahlentheorie erhält eine Nummer („Gödel-Nummer").
- So kann ein zahlentheoretischer Satz als zahlentheoretische Aussage verstanden werden oder als Aussage über zahlentheoretische Aussagen.
- Neue Aussage: „Für diesen Satz gibt es keinen Beweis." (unbeweisbar, aber wahr)



Kurt Gödel, 1906–1978

Es gibt also wahre zahlentheoretische Aussagen, für deren Beweis die Methoden der Zahlentheorie zu schwach sind.

[5] wie z.B. *Principia Mathematica* von Russell u. Whitehead, 1913