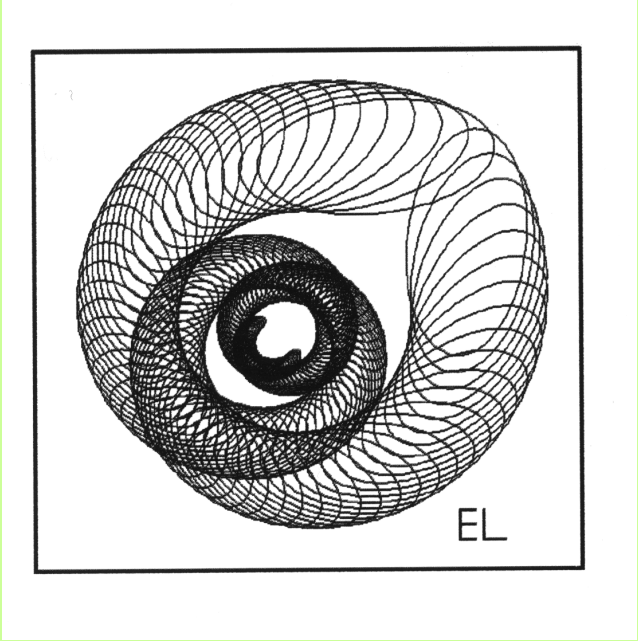
Das (3a+1)-Problem und das Busy-Beaver-Problem
Einstieg in das Thema „Nicht berechenbare Funktionen“

Vortrag im Fach Informatik auf der MNU-Tagung 2006 in Karlsruhe -
Mittwoch, d. 12.4.2006, 12.15 Uhr



Lehmanns-Logo

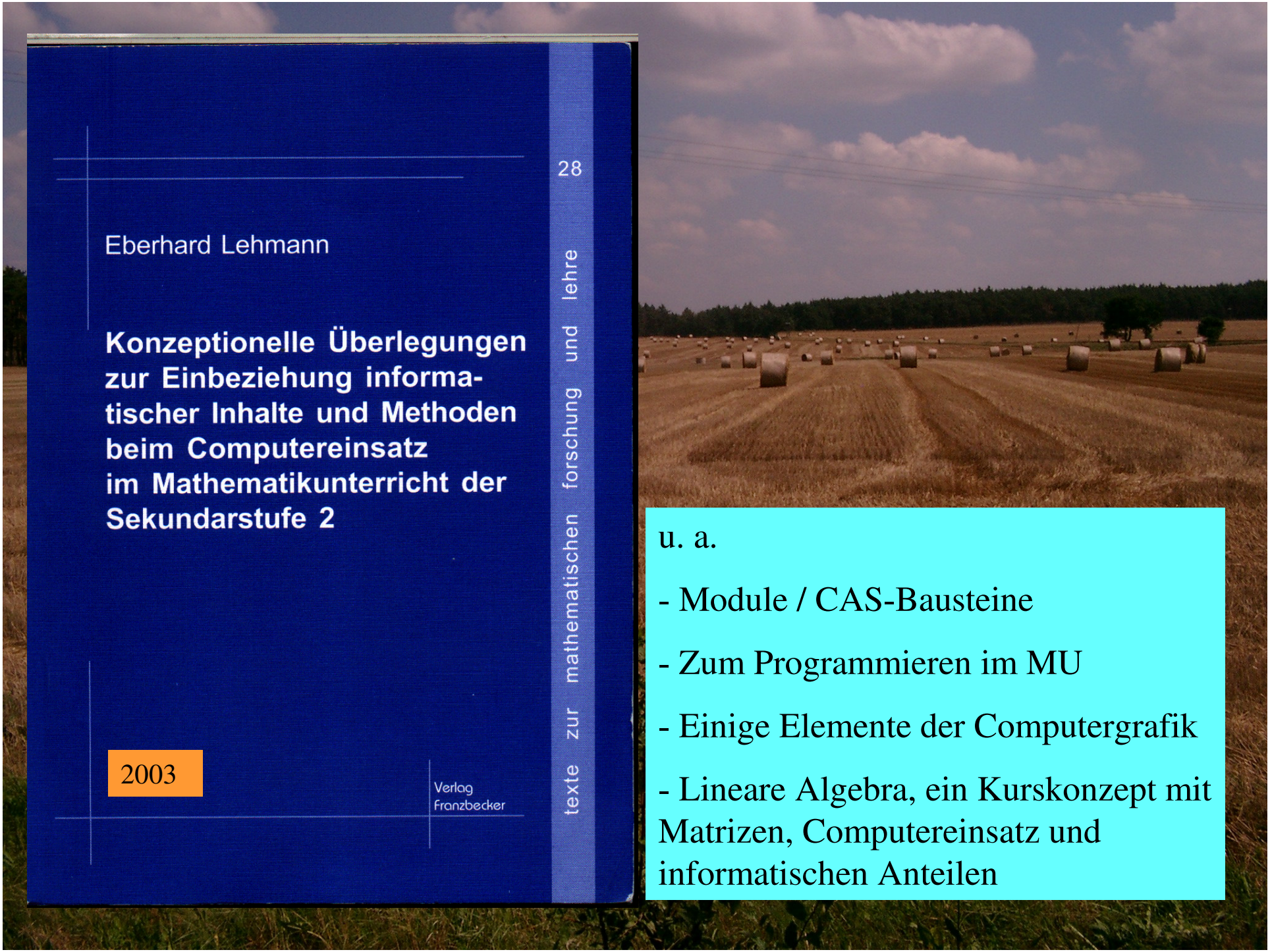
Plot2.exe

Dr. Eberhard Lehmann, Berlin

mirza@snafu.de --- www.snafu.de/~mirza

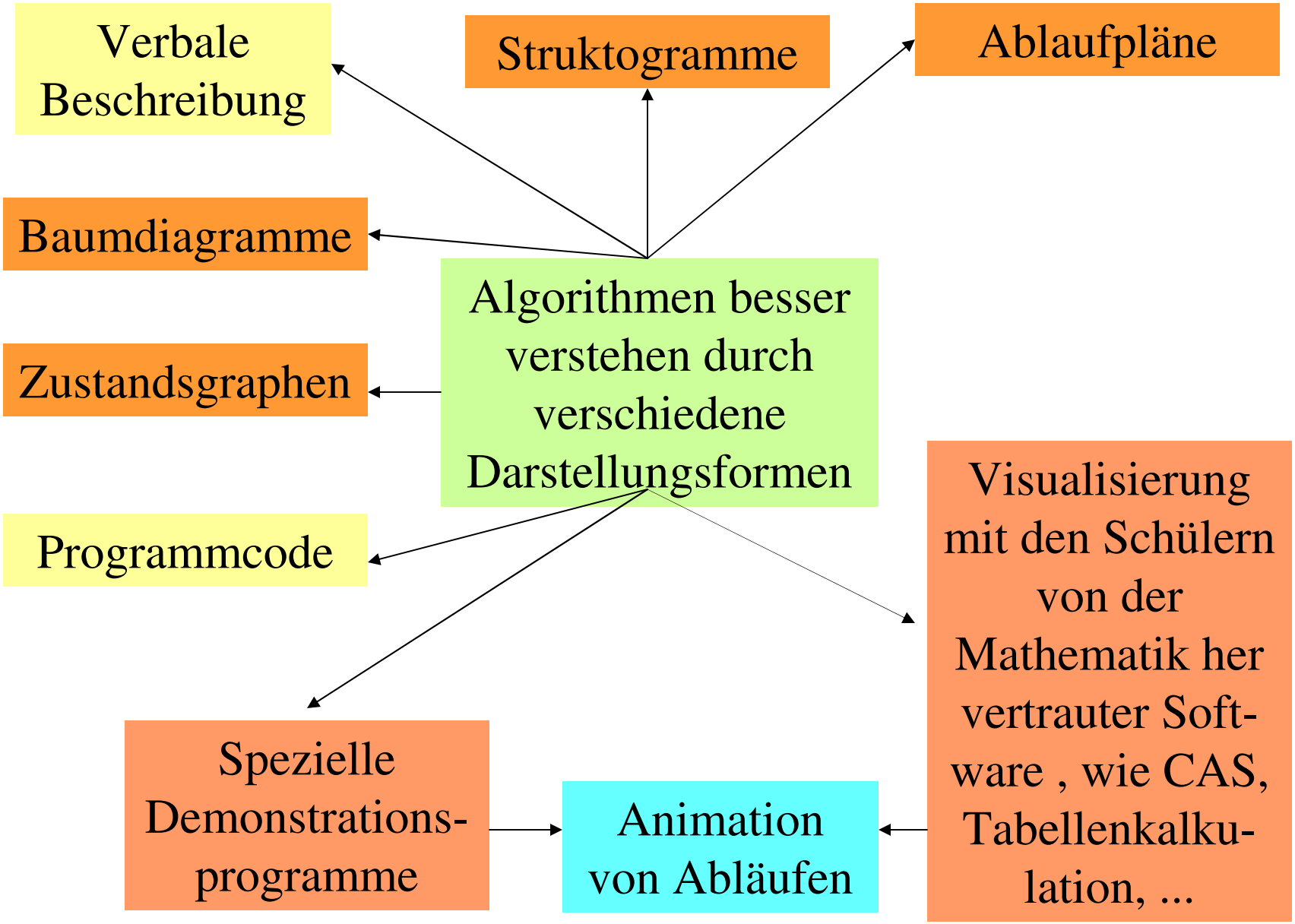
Das (3a+1)-Problem und das Busy-Beaver-Problem –
Einstieg in das Thema „Nicht berechenbare Funktionen“

Themen der theoretischen Informatik werden im Informatik-Unterricht häufig vernachlässigt, vermutlich wegen der Bezüge zur Mathematik. Die oben genannten Probleme sind für beide Fächer geeignet. Die Schulmathematik gewinnt an Transparenz für Schüler, wenn sie erkennen, dass es bei den vielen betrachteten Funktionen auch solche gibt, die „nicht berechenbar“ sind. Das bekannte (3a+1)-Problem bildet dafür einen leicht verständlichen Einstieg. Das Busy-Beaver-Problem schließt sich in günstiger Weise an und hat nach den Unterrichtserfahrungen noch stets das Interessen der Schüler geweckt. Beide Probleme lassen sich gut visualisieren. Die hierbei verwendeten Zustandsgraphen haben darüber hinaus auch für andere Gebiete der Informatik und Mathematik Bedeutung (Automaten, Markow-Ketten).



u. a.

- Module / CAS-Bausteine
- Zum Programmieren im MU
- Einige Elemente der Computergrafik
- Lineare Algebra, ein Kurskonzept mit Matrizen, Computereinsatz und informatischen Anteilen


Verbale Beschreibung
Struktogramme
Ablaufpläne
Baumdiagramme
Algorithmen besser verstehen durch verschiedene Darstellungsformen
Zustandsgraphen
Programmcode
Visualisierung mit den Schülern von der Mathematik her vertrauter Soft-ware , wie CAS, Tabellenkalku-lation, ...
Spezielle Demonstrations-programme
Animation von Abläufen

Informatik und Mathematik - Ein leicht verständlicher Algorithmus, der es in sich hat!
Das 3a+1-Problem, siehe A.Engel, Elementarmathematik vom algorithmischen Standpunkt Klett-Verlag, 1977 (Werte verglichen mit Buchwerten, sind ok)

Visualisierung des 3a+1-Problems, eine Vorstufe zum Halteproblem (theoretische Informatik) - Die Visualisierung erfolgt graphisch oder mit der Wertetafel.

Allgemeine Lösung

Startwert wählen, wenn Zahl gerade, dann zahl:=zahl / 2, sonst zahl:=3*zahl +1.

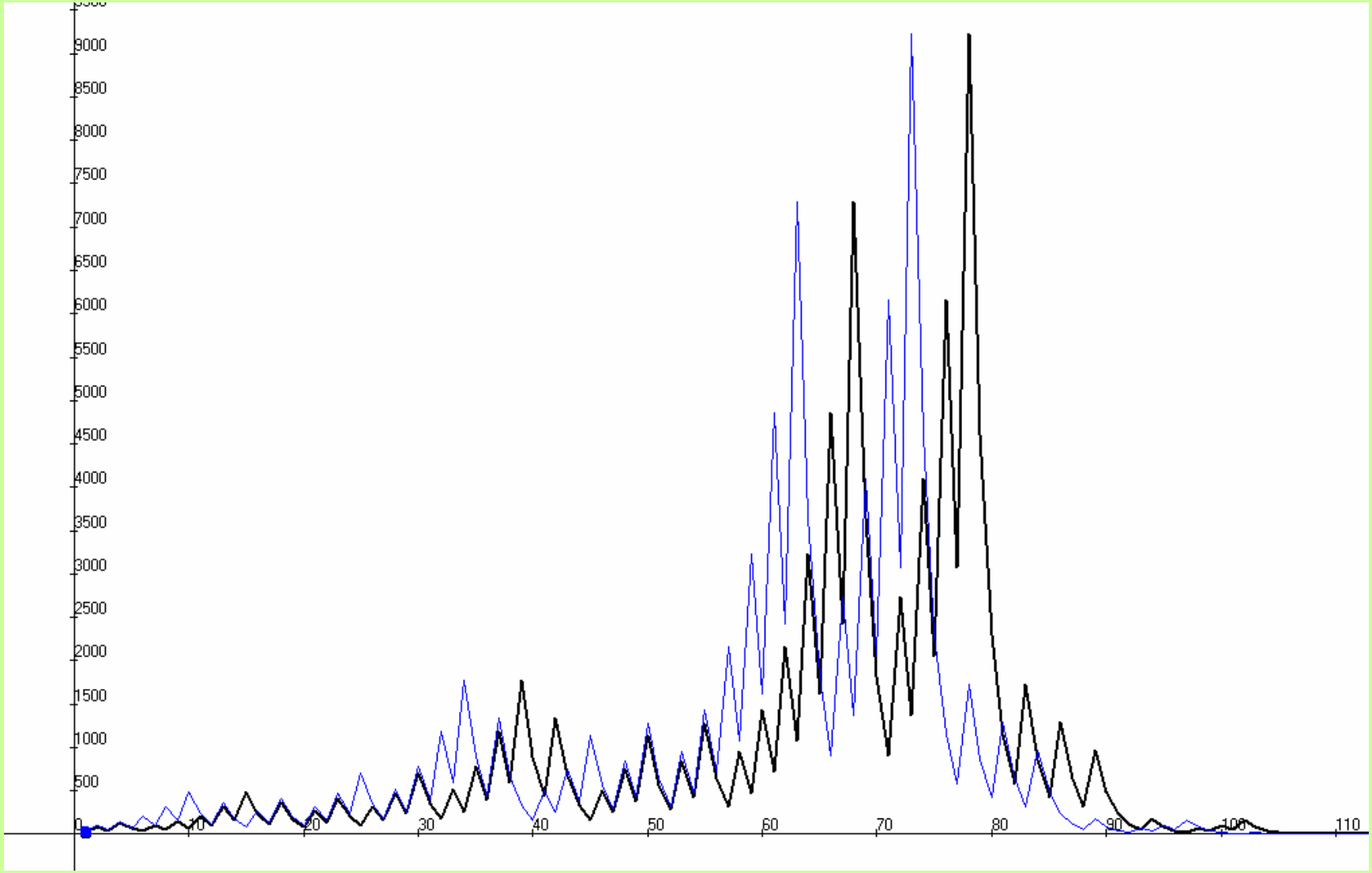
Zahlenbeispiel: 15, 46, 23, 70, 35, 106, 53, 160, 80, 40, 20, 20, 5, 16, 8, 4, 2, 1, 4, 2, 1

Der Algorithmus für ANIMATO:
f1: u Startwert
f2: {n = 1 : f1 : {(f2(n-1)/2 = int(f2(n-1)/2)) : f2(n-1)/2 : 3*f2(n-1)+1}}, Folgenwerte

Beispiele: Startwerte sind 27 bzw. 31 - hier zum Vergleich der Laufzeit bis zum Zyklus.
f3: 27 f4: {n=1:f3:**{(f4(n-1)/2 = int(f4(n-1)/2)) : f4(n-1)/2 : 3*f4(n-1)+1}**

f6: 31 f7: {n=1:f6:**{(f7(n-1)/2 = int(f7(n-1)/2)) : f7(n-1)/2 : 3*f7(n-1)+1}**

Informatik und Mathematik: Auf dem Weg zum Halteproblem - das (3a+1)-Problem



Inf-3a+1-Halteproblem.pl2

[PDF] Elementarer Lösungsansatz zum Collatz-**Ulam-Problem** CUP oder das " **...**
Dateiformat: PDF/Adobe Acrobat - HTML-Version
... November 2000 1 Elementarer Lösungsansatz zum Collatz-**Ulam-Problem** CUP oder das
"3a + 1"-Problem >>>>> Entwurf - Draft <<<<< Immo O. Kerner **...**
www.inf.hs-zigr.de/~wagenkn/CUP3AP1-neu.pdf - Ähnliche Seiten

www.mathematik.de | Pressestimmen
Pressestimmen, Das **Ulam-Problem**. 13.06.2004 Neue Zürcher Zeitung
www.mathematik.
de wurde bekannt, dass ein Artikel in der Neuen Zürcher **...**
www.mathematik.de/mde/presse/ pressestimmen/ulamproblem.html - 14k - Im Cache - Ähnliche Seiten

The 3x+1 problem and its generalizations - [Diese Seite übersetzen]
... The 3x+1 problem, also known as the Collatz problem, the Syracuse problem, Kakutani's
problem, Hasse's algorithm, and **Ulam's problem**, concerns the behavior of **...**
www.cecm.sfu.ca/organics/papers/lagarias/ - 4k - Im Cache - Ähnliche Seiten

Ulam's Problem -- from MathWorld - [Diese Seite übersetzen]
... U. **Ulam's Problem**. Collatz Problem. **...** Eric W. Weisstein. "**Ulam's Problem**." From MathWorld--A Wolfram Web Resource.
http://mathworld.wolfram.com/UlamsProblem.html. **...**
mathworld.wolfram.com/UlamsProblem.html - 15k - Im Cache - Ähnliche Seiten

Das Halteproblem fragt nach einem Algorithmus, mit dem man bei Programmen, Automaten oder Computern feststellen kann, ob sie für gewisse oder für alle Eingaben anhalten oder nicht.

Die Fortsetzung des Ansatzes “(3a+1)-Folge” kann mit dem Busy-Beaver-Problem erfolgen.

Man nehme ein nach beiden Seiten **unendliches Band**, das **Bandalphabet** { #, / } und eine **Turingmaschine TM** mit n Zuständen **Z0, Z1, Z2, Z(n-1)** und einem zusätzlichen Haltezustand ZE.

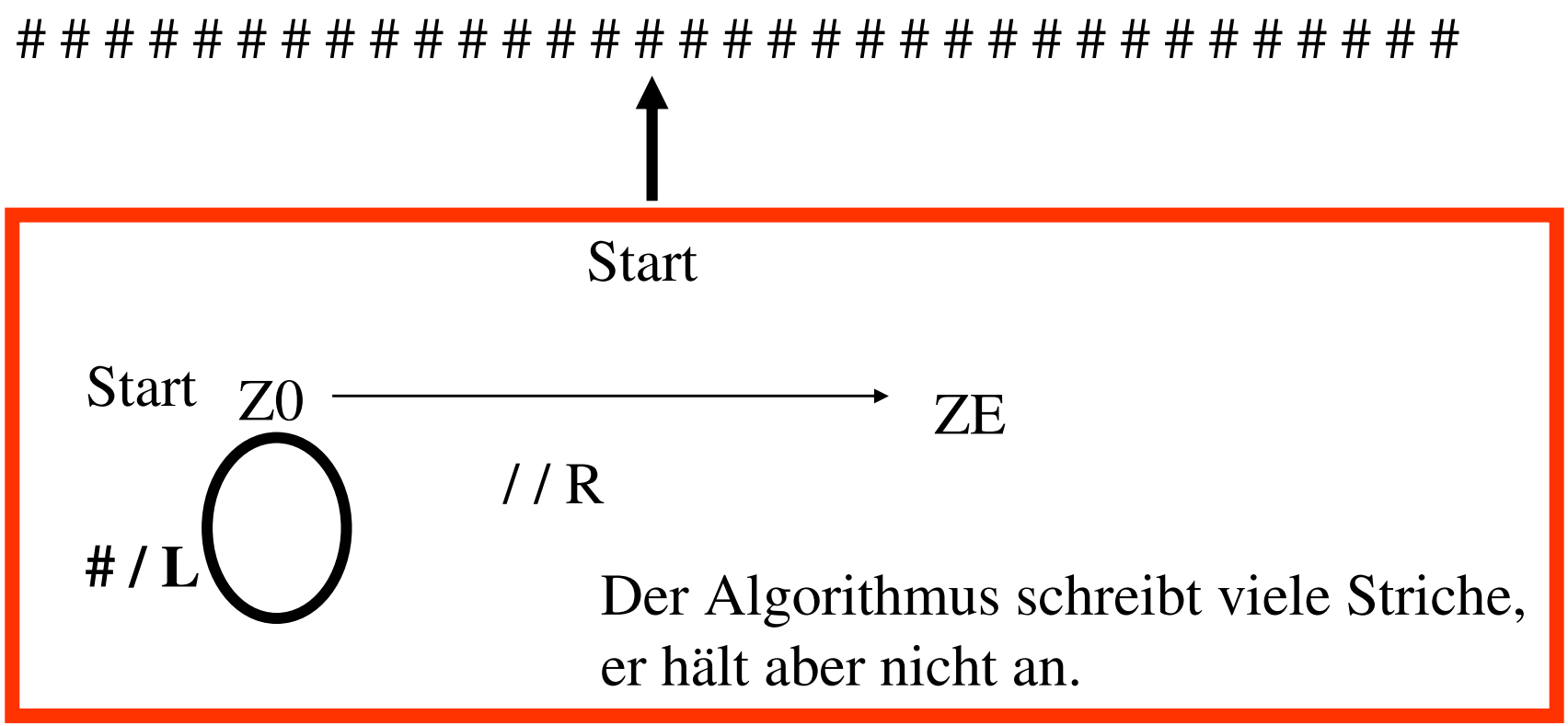
In jedem Schritt liest die TM entweder # oder / und ersetzt das Zeichen durch # oder / und macht eine Bewegung nach links L oder rechts R oder hält an.

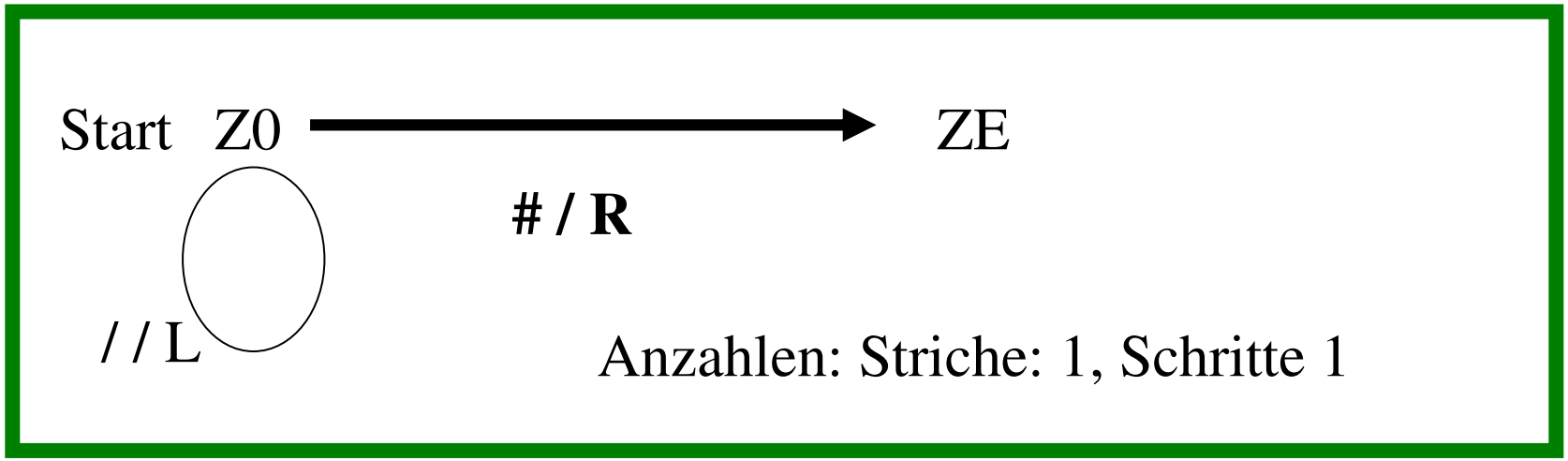
Die Turingmaschine mit n Zuständen, die auf dem anfangs leeren Band (anfangs lauter #-Zeichen) die meisten Striche schreibt, erhält den Titel **fleißiger Biber** (die Strichfolge darf Lücken enthalten).

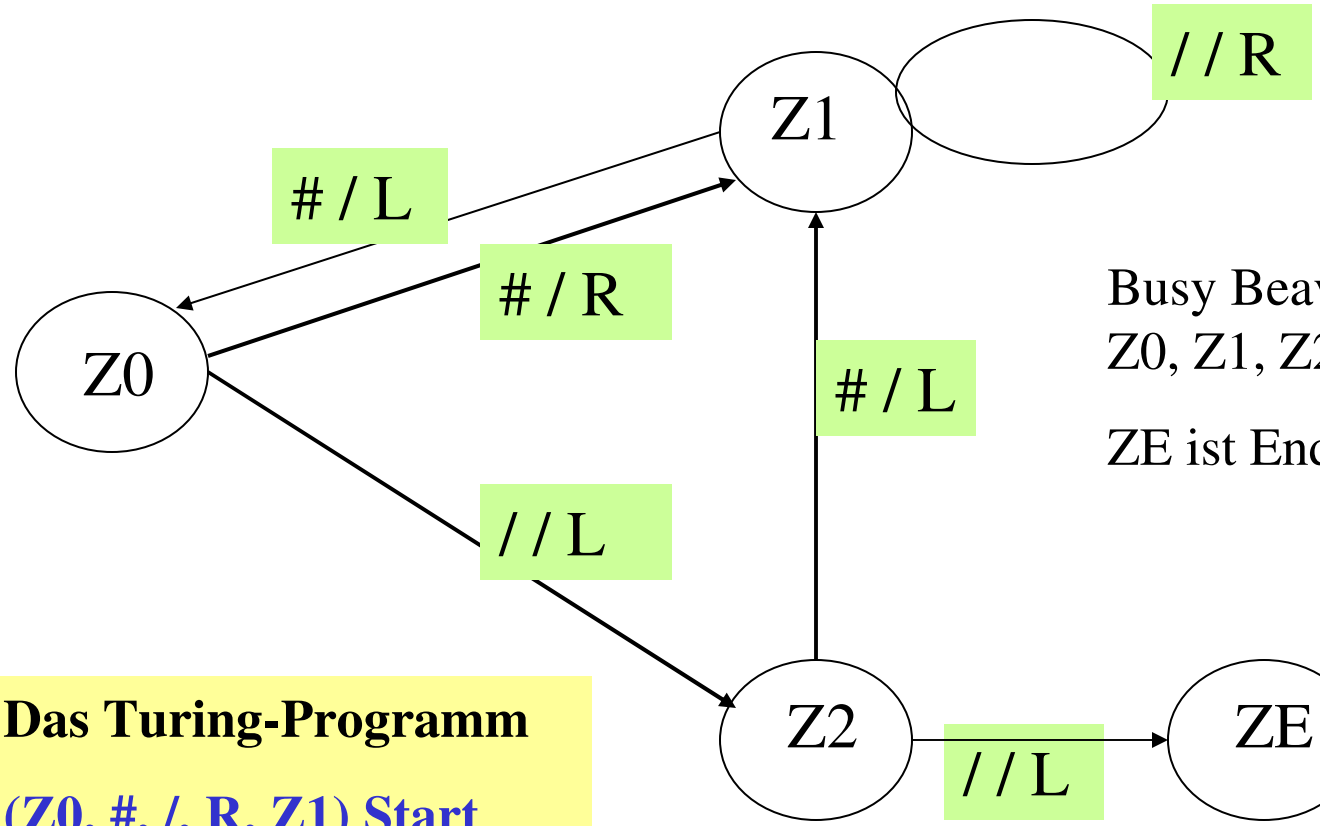
Das Halteproblem fragt nach einem Algorithmus, mit dem man bei Programmen, Automaten oder Computern feststellen kann, ob sie für gewisse oder für alle Eingaben anhalten oder nicht.

Unentscheidbarkeit des Halteproblems für Turingmaschinen:
Es gibt keinen Algorithmus, der für alle Turingmaschinen und alle Eingaben feststellt, ob die TM mit der entsprechenden Eingabe anhält oder nicht.

Es gibt keinen Algorithmus, der für jedes Programm feststellt, ob es eine Endlosschleife enthält oder nicht.


#####################################
Start
Start Z0
ZE
/ / R
/ L
Der Algorithmus schreibt viele Striche,
er hält aber nicht an.


Start Z0
ZE
/ R
/ / L
Anzahlen: Striche: 1, Schritte 1



Busy Beaver für 3 Zustände
Z0, Z1, Z2

ZE ist Endzustand

Das Turing-Programm

(Z0, #, /, R, Z1) Start

(Z0, /, /, L, Z2)

(Z1, #, /, L, Z0)

(Z1, /, /, R, Z1)

(Z2, #, /, L, Z1)

(Z2, /, /, L, ZE) Stop

…	#	#	#	#	#	#	#	#	#	#	…

Start mit Z0

	Gelesenes Zeichen #	Gelesenes Zeichen /
Zustand Z0	(/, R, Z1)	(/, L, Z2)
Zustand Z1	(/, L, Z0)	(/, R, Z1
Zustand Z2	(/, L, Z1)	(/, L, ZE)

Das Turing-Programm

(Z0, #, /, R, Z1) Start

(Z0, /, /, L, Z2)

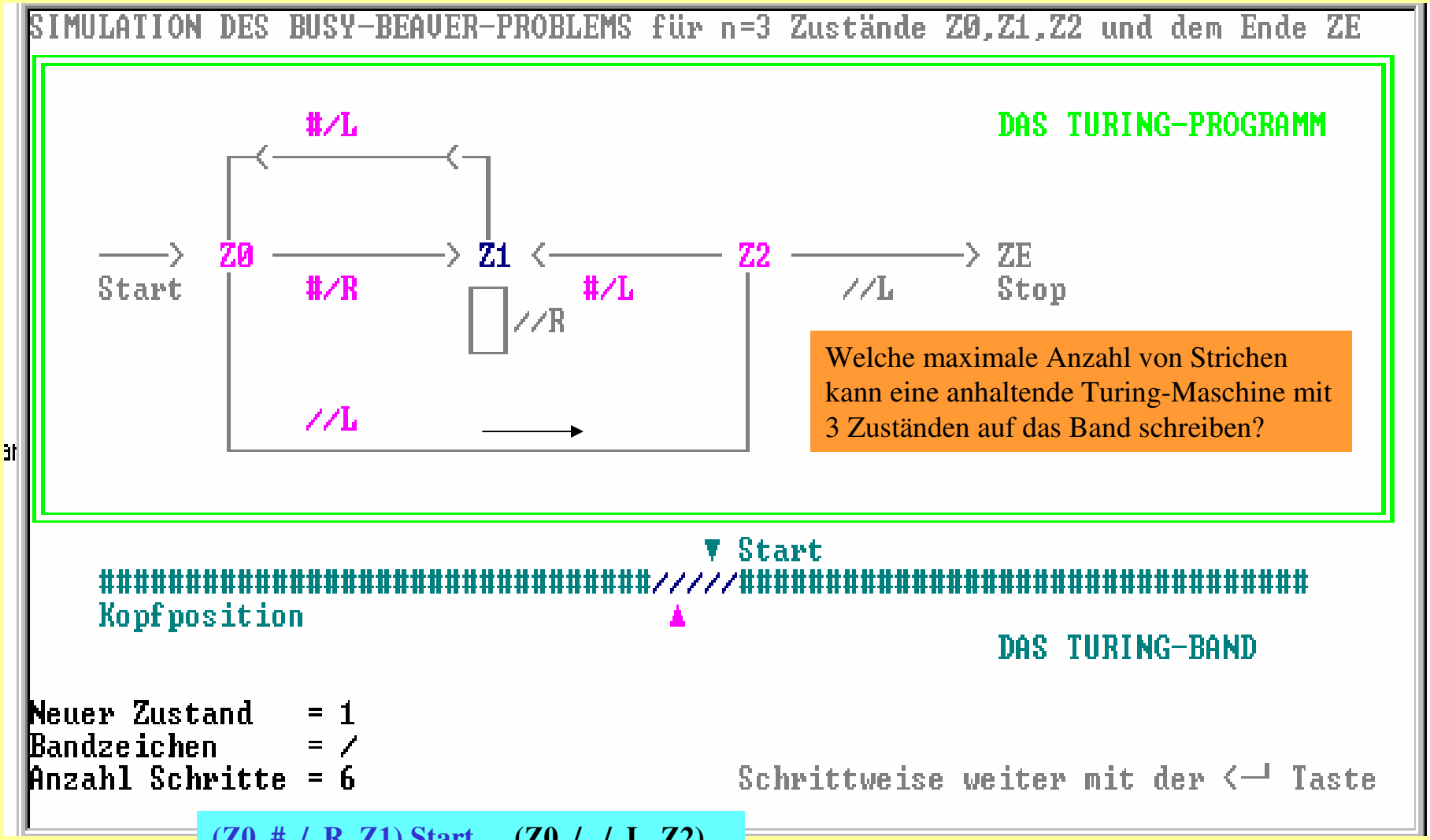
(Z1, #, /, L, Z0)

(Z1, /, /, R, Z1)

(Z2, #, /, L, Z1)

(Z2, /, /, L, ZE) Stop

Simulationsprogramm - Busy Beaver für 3 Zustände



(Z0, #, /, R, Z1) Start	(Z0, /, /, L, Z2)
(Z1, #, /, L, Z0)	(Z1, /, /, R, Z1)
(Z2, #, /, L, Z1)	(Z2, /, /, L, ZE)

Simulationsprogramm - Busy Beaver für 3 Zustände

Welche maximale Anzahl von Strichen kann eine anhaltende Turing-Maschine mit 3 Zuständen auf das Band schreiben?

(Z0, #, /, R, Z1)	**(Z0, /, /, L, Z2)**
(Z1, #, /, L, Z0)	**(Z1, /, /, R, Z1)**
(Z2, #, /, L, Z1)	**(Z2, /, /, L, ZE)**

Beaver3.exe

Die drei Funktionen T(n), Σ(n) und S(n)

$T(n) = (4(n+1))^{2n}$

Anzahl der Turingtafeln bei n Zuständen

Σ(n): N→N

Σ(n): Maximale Anzahl von Strichen (/), die eine haltende Turingmaschine mit N Zuständen und dem Bandalphabet {/, #} auf das leere Band schreibt.

S(n): N→N

S(n): Anzahl der Schritte (Kopfbewegungen), die für Σ(n): benötigt werden.

Aktueller Zustand → Zelle lesen:

n Zustände * je 2 Fälle (/ oder #)
= = **2n Tabellenpositionen**

Zelle beschreiben → bewegen → auf Zustand:

|{/,#}| * |{L,R} | * |{Z0,Z1,Z2,...Zn-1,ZE}| =
= 4(n+1) Einträge an jeder Tabellenposition

$$T(n) = (4(n+1))^{2n}$$

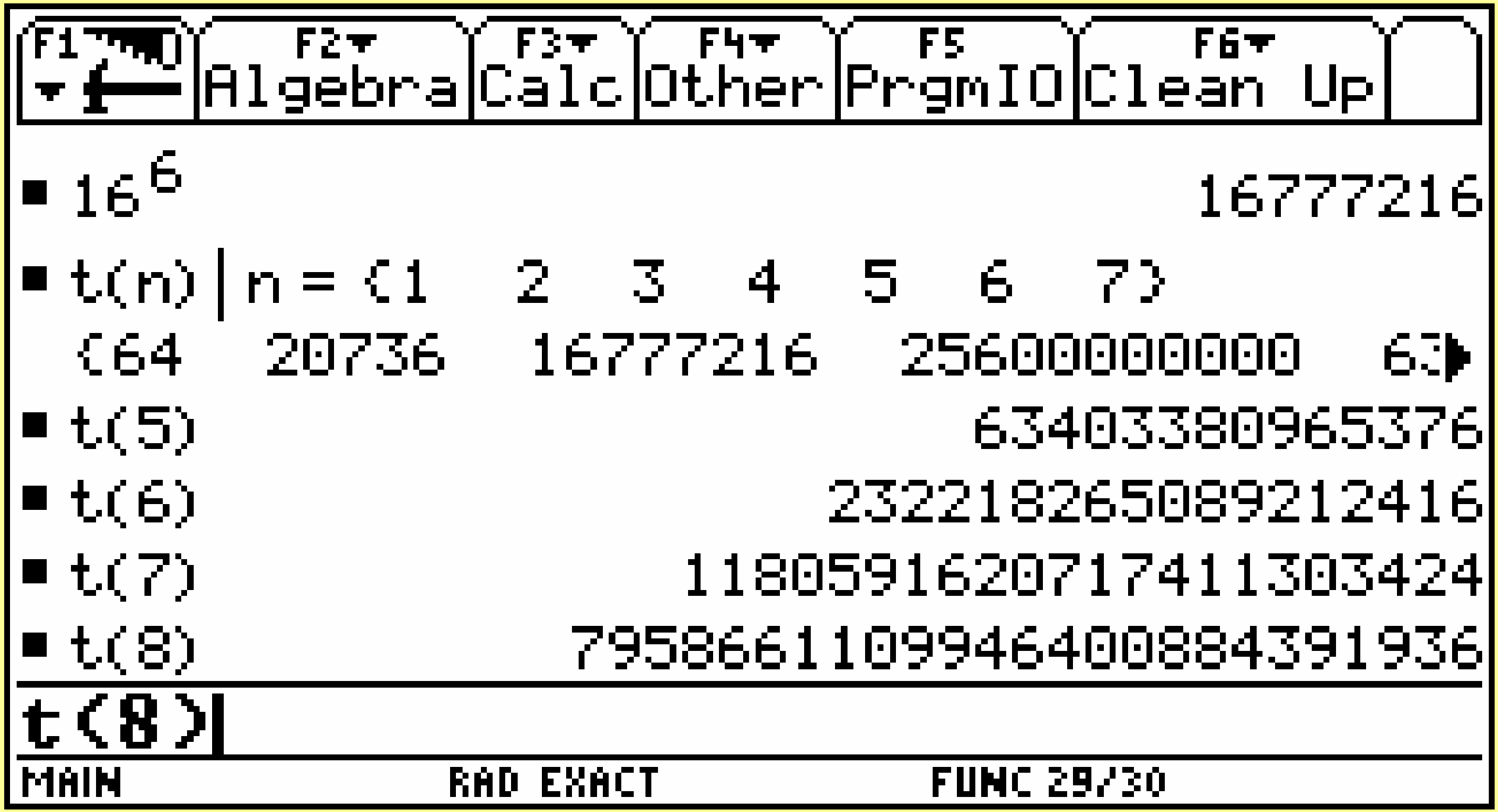
Anzahl der Zustände (ohne Endzustand) n	Anzahl der möglichen Turingtafeln $T(n)=(4(n+1))^{2n}$	Busy Beaver Anzahl der Striche $\sum(n)$	Busy Beaver Anzahl der Schritte S(n)
1	64	1	
2	20 736	4	
3	16 777 216	6	13
4	25 600 000 000	13	
5	63 403 380 965 376	≧ 4098	
6	28^{12}		
7	32^{14}		
8	36^{16}		
k	berechenbare Funktion	nicht berechenbare Funktion	nicht berechenbare Funktion

Abb. 3.3.2-k

Beweis siehe

Die maximale Anzahl der Striche bei n Zuständen bis der Busy-Beaver ermittelt ist, ist eine nicht berechenbare Funktion.

Einsatz mathematischer Software – hier ein CAS (TI, Voyage 200)



Anzahl von Turingtafeln mit n Zuständen

Zum Fall n = 5 schreibt *A. K. Dewdney*

„Der Sprung von $\sum = 13$ bei n=4 auf $\sum \geq 4098$ bei n=5 ist symptomatisch für die nichtberechenbare Natur von $\sum$. Die Zahl 4098 hat eine interessante Geschichte hinter sich.

In der Augustausgabe 1984 des *Scientific American* erschien ein Artikel über den damals fleißigsten 5-Zustands-Biber, der 1984 von Uwe Schult, einem deutschen Informatiker, gefunden wurde. Schults fleißiger Biber erzeugt 501 Einsen, bevor er anhielt. Als Antwort auf den Artikel führte George Uhing, ein amerikanischer Programmierer, eine Computersuche nach fleißigen Bibern mit fünf Zuständen durch und fand einen, der 1915 Einsen [*Anm.: Striche*] ausgab, bevor er anhielt. Später, 1989, wurde Uhings Rekordbiber durch einen neuen, fleißigeren Biber verdrängt, der von Jürgen Buntrock und Heiner Marxen in Deutschland entdeckt wurde. Der Buntrock-Marxen-Biber, der bei einer dreitägigen Suche auf einem Hochgeschwindigkeitsrechner gefunden wurde, schreibt 4098 Einsen, bevor er anhält!" *[A. K. Dewdney: Der Turing-Omnibus – eine Reise durch die Informatik mit 66 Stationen, Springer-Verlag, Berlin-Heidelberg, 1995, S. 287]*

Satz 1: Zu jeder Anzahl n von Zuständen muss ein Busy-beaver existieren!

Die Aussage ist plausibel, da zu dem Busy-beaver eine spezielle Turingtafel gehört und es ersichtlich nur endlich viele Turingtafeln zu jedem n und dem Bandalphabet B2 = { /, # } gibt. Satz 2 sagt aber noch Präziseres aus!

Satz 2: Die Anzahl der Turing-Maschinen mit n Zuständen beträgt

$$T(n) = (4(n+1))^{2n}$$

Satz 3: $\Sigma(n)$: N→N

(Anzahl der Striche bei n Zuständen bis der Busy-beaver ermittelt ist) und

S(n): N→N

(Anzahl der Schritte bei n Zuständen bis der Busy-beaver ermittelt ist)

sind nicht berechenbare Funktionen.

Der Beweis dieses Satzes ist für den Schulunterricht wohl zu schwierig. Man findet ihn u.a. im Internet, *[Reinhard Völler (http://www.informatik.fh-hambung.de/~voeller/th/thinf/node16/html)]*.

Mit den vorstehenden Überlegungen ist nun auch der Begriff der berechenbaren Funktion vorbereitet. *Duden Informatik, Duden-Verlag, 1993, S.80 f.:*

„Eine Funktion f: M→ N heisst berechenbar, wenn es einen Algorithmus gibt, der für jeden Eingabewert m aus M, für den f(m) definiert ist, nach endlich vielen Schritten anhält und als Ergebnis f(m) liefert; in allen Fällen, in denen f(m) nicht definiert ist, bricht der Algorithmus nicht ab.
Anschaulich gesprochen (Anm.: und für die Schüler sehr handlich) ist eine Funktion berechenbar, wenn man sie programmieren kann."

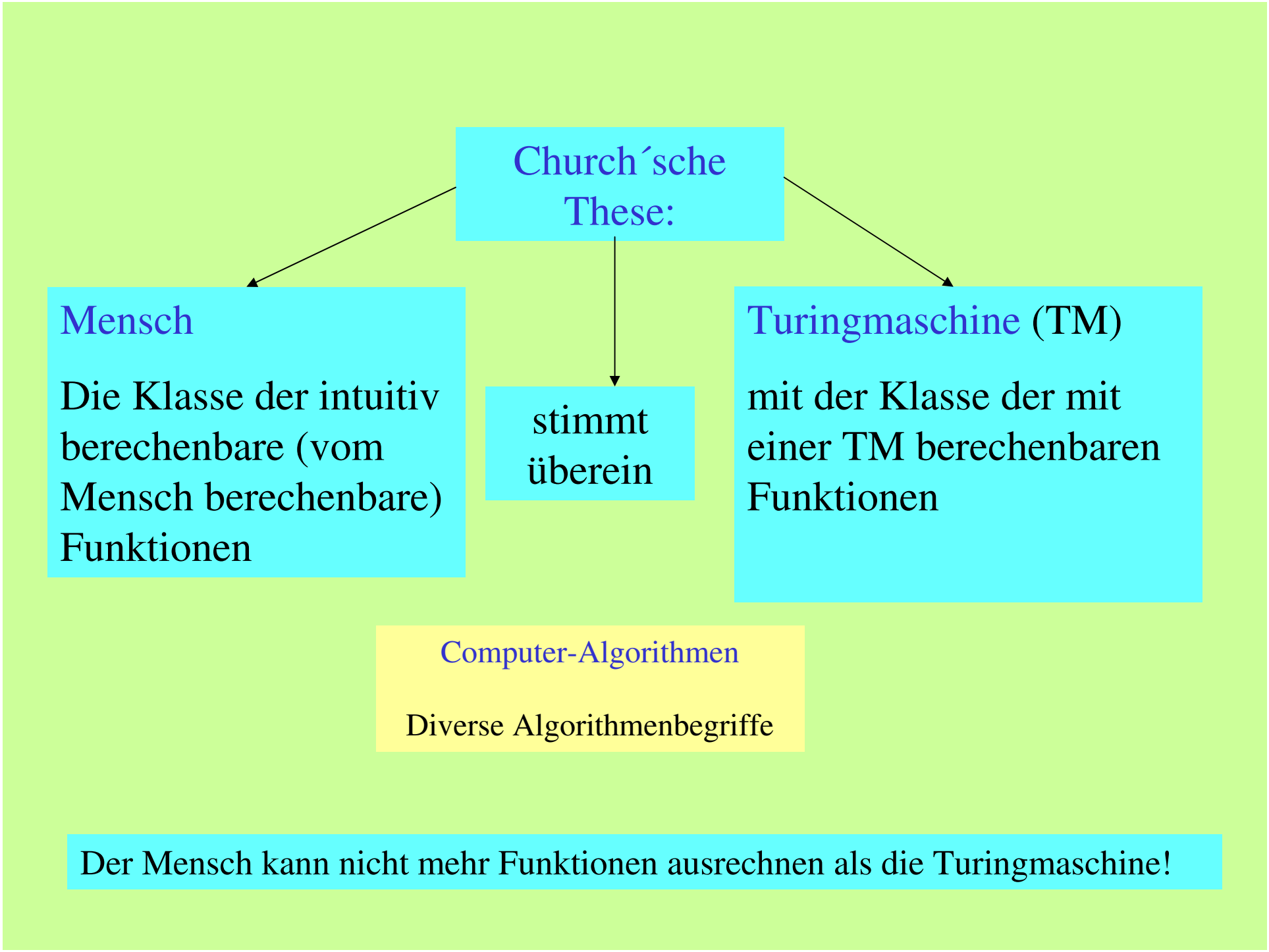
Im normalen Mathematikunterricht wird angesichts des dort verwendeten Funktionenmaterials auf den Begriff der berechenbaren Funktion nicht eingegangen, ein Mangel, der sich bei Berücksichtigung informatischer Aspekte, wie oben gezeigt, beseitigen lässt! Man beachte dazu auch Kapitel 3.4.2.

Eine Plausibilitätsüberlegung zur Nichtberechenbarkeit von **Σ(n): N→N**

Ein Algorithmus zur Berechnung von $\sum(n)$ für jedes n könnte folgendermaßen arbeiten:

(1) Eingabe von n
(2) Ermitteln aller Turingmaschinen mit n Zuständen (ihre Anzahl ist T(n))
(3) Ermitteln aller nicht haltenden Maschinen und diese aussondern.
(4) Alle haltenden Turingmaschinen (ihre Anzahl ist TH(n)) auf einem leeren Band (lauter #-Zeichen) starten
(5) Jeweils die entstehenden /-Anzahlen notieren (in einer Menge M sammeln)
(6) Das Maximum in M nehmen. Das ist dann der Busy-Beaver.

Dieser Algorithmus arbeitet jedoch nicht wie gewünscht, weil es in (3) nicht gelingt, alle nicht haltenden Maschinen zu finden. Damit sind auch (4)-(6) irrelevant.


Church´sche These:
Mensch
Die Klasse der intuitiv berechenbare (vom Mensch berechenbare) Funktionen
stimmt überein
Turingmaschine (TM)
mit der Klasse der mit einer TM berechenbaren Funktionen
Computer-Algorithmen
Diverse Algorithmenbegriffe
Der Mensch kann nicht mehr Funktionen ausrechnen als die Turingmaschine!

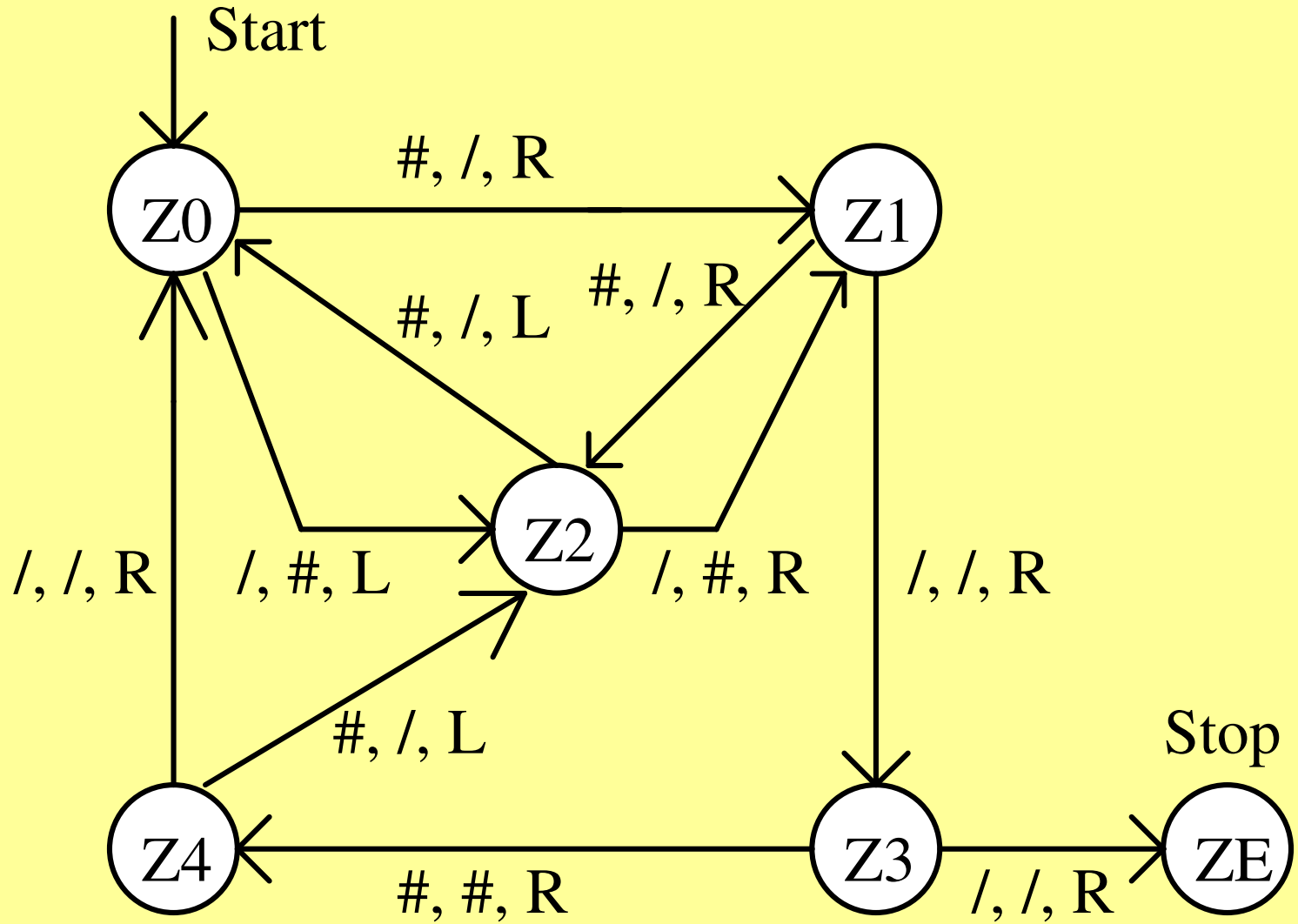


Abb.3.3.2-g: Beaver für 5 Zustände (und Endzustand), 501 Striche, aber kein Busy-Beaver (nach *Gasper, Leiß, Spengler, Stimm: Technische und theoretische Informatik, Bayerischer Schulbuch-Verlag, München 1992*)

Fleißiger Biber mit fünf Zuständen und 4098 Strichen /

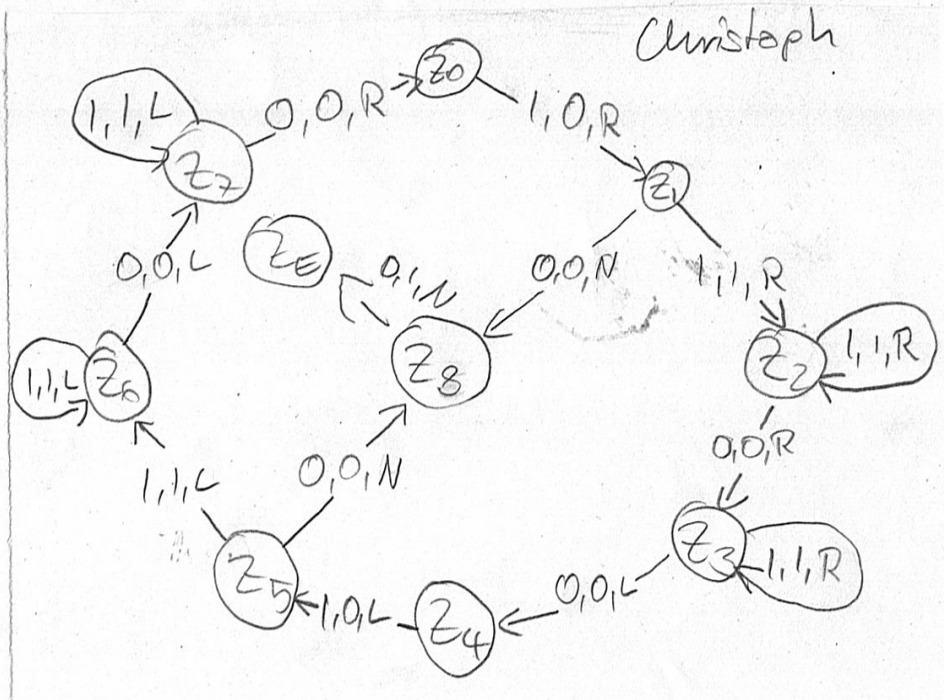
Zustand	# gelesenes Zeichen	/ gelesenes Zeichen
Z0	/; R; Z2 2*2*6=24	/; L; Z3 2*2*6=24
Z1	/; R; Z3 2*2*6=24	/; R; Z2 2*2*6=24
Z2	/; R; Z4 2*2*6=24	#; L; Z5 2*2*6=24
Z3	/; L; Z1 2*2*6=24	/; L; Z4 2*2*6=24
Z4	/; R; ZE 2*2*6=24	#; L; Z1 2*2*6=24

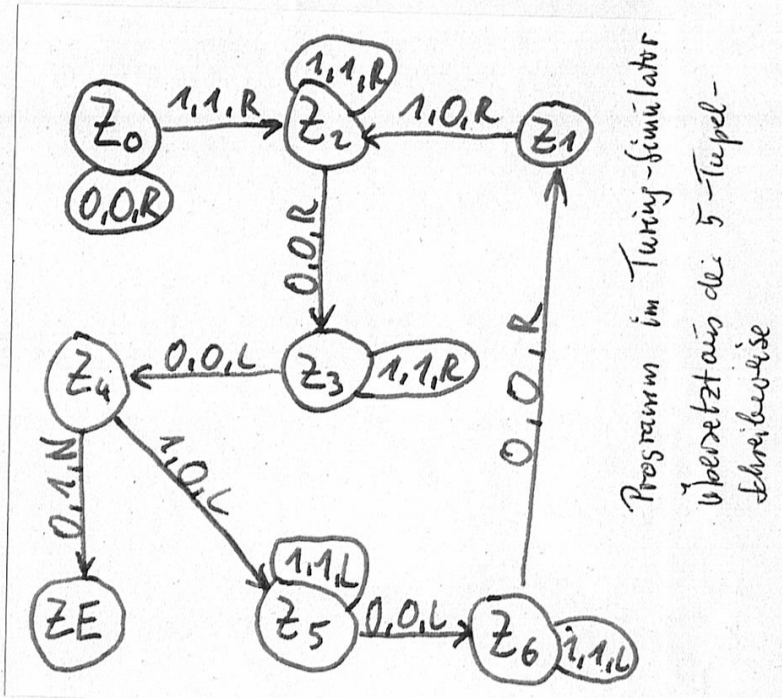
24^10 mögliche Turing-Tafeln

Im Unterricht wurden von den Schülern Zustandsgraphen zum Busy Beaver für n = … erstellt.

(hier nicht mehr vorhanden, siehe jedoch folgende Folie mit Zustandsgraphen zur Subtraktion auf der Turingmaschíne)

Christoph

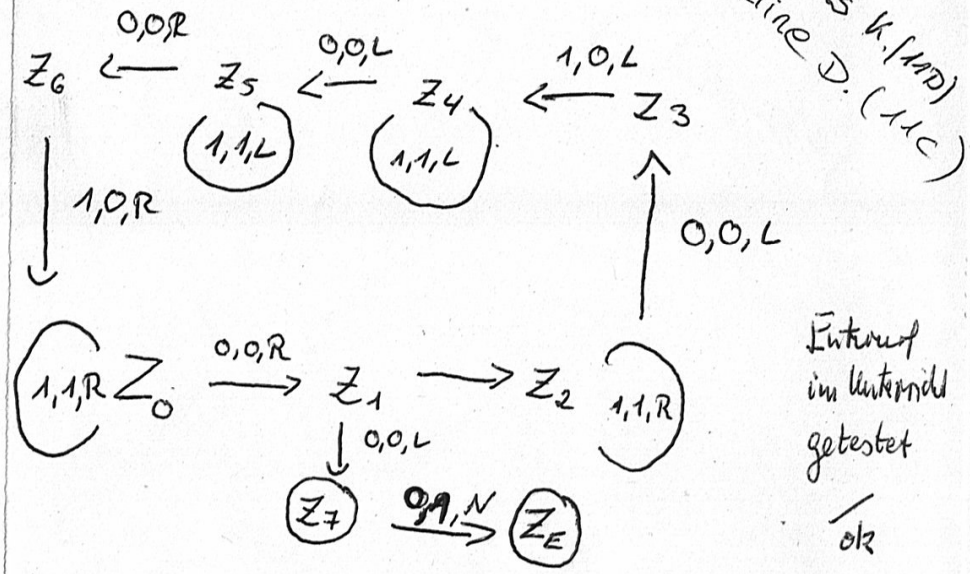




Programm im Turing-Simulator übersetzt aus der 5-Tupel-Schreibweise

Zusammenhang in-b | Schmann, 1.9.98
Stunde 5 des Kurses
Subtraktion a-b (a>b)
Turingprogramm-Entwürfe

Thomas K. (11D)
Céline D. (11C)



Entwurf im Unterricht getestet

o.k.

Eine Unterrichtseinheit über Zustandsgraphen

Als ein Weg (von vielen möglichen Wegen) durch eine Unterrichtseinheit „Zustandsgraphen" – gedacht für Leistungskurse – wird der in Abb. 3.3.1-b skizzierte Weg vorgeschlagen.

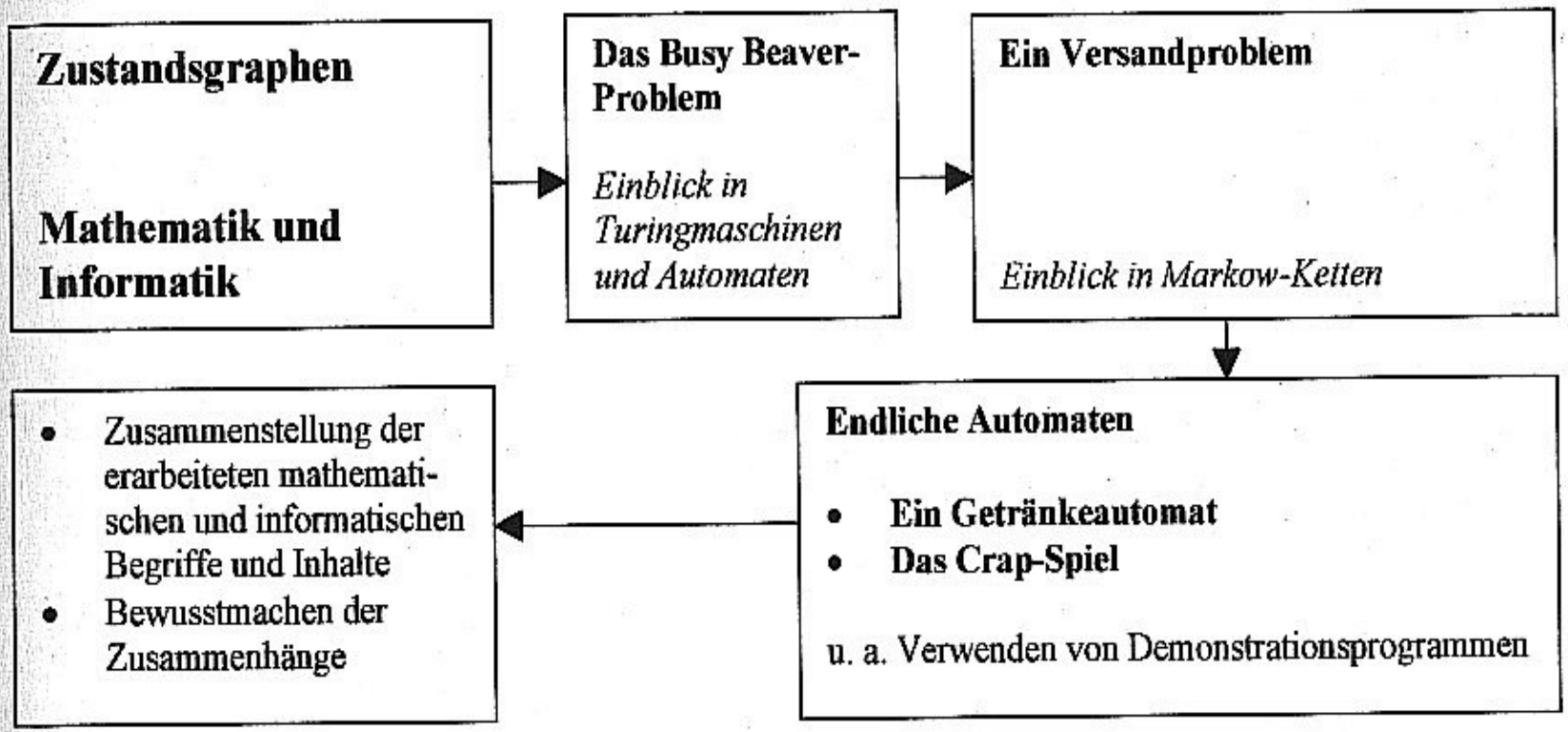
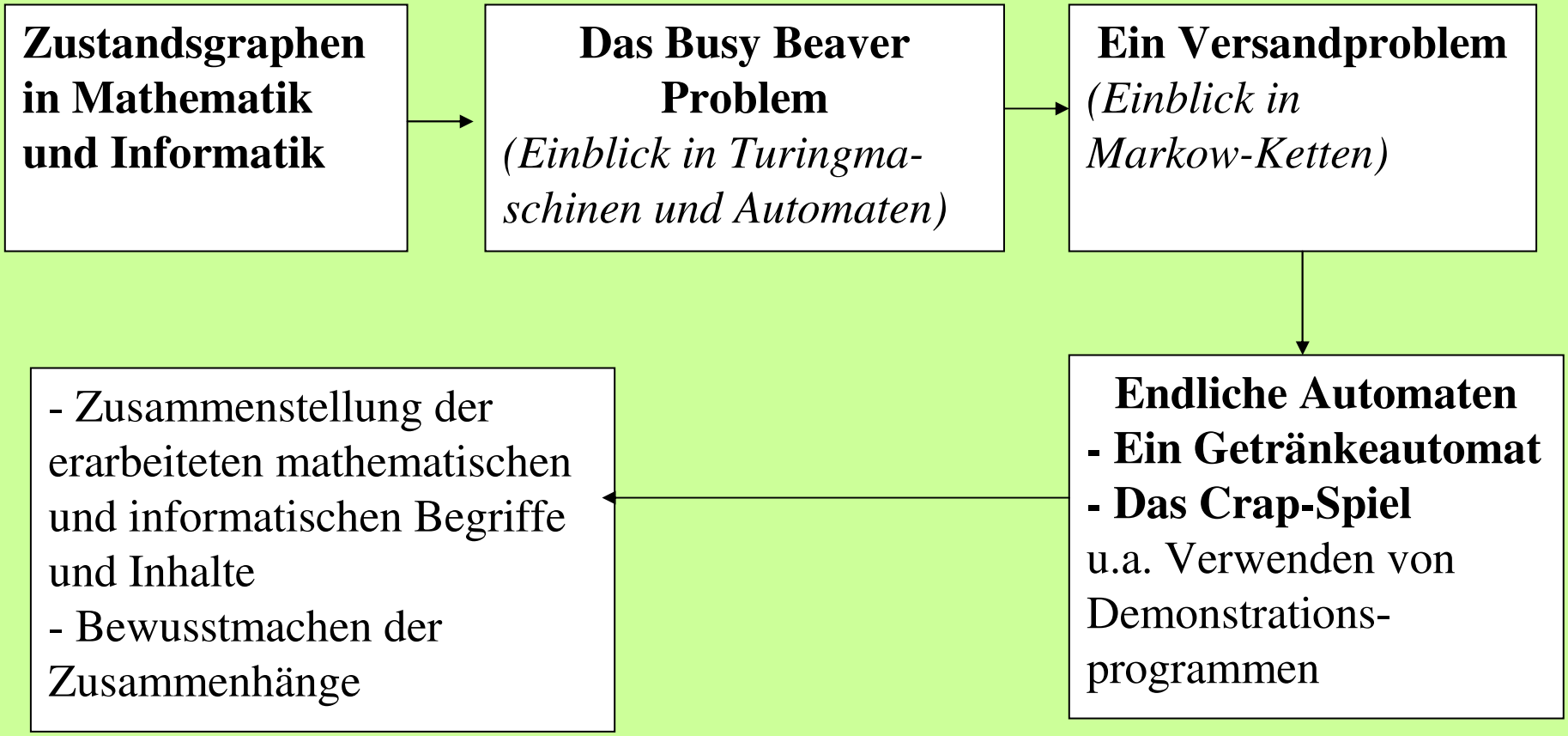


Abb. 3.3.1-b

Die bei diesem Lehrgang auftretenden Vernetzungen (und weitere, hier nicht verfolgte Zusammenhänge) sind vielfältig; Abbildung 3.3.1-d (Seite 112) sagt Einiges darüber aus.

Hinweis:

Als ein Weg (von vielen möglichen Wegen) durch eine **Unterrichtseinheit „Zustandsgraphen”** – gedacht für Leistungskurse – wird der in Abb. 3.3.1-b skizzierte Weg vorgeschlagen.



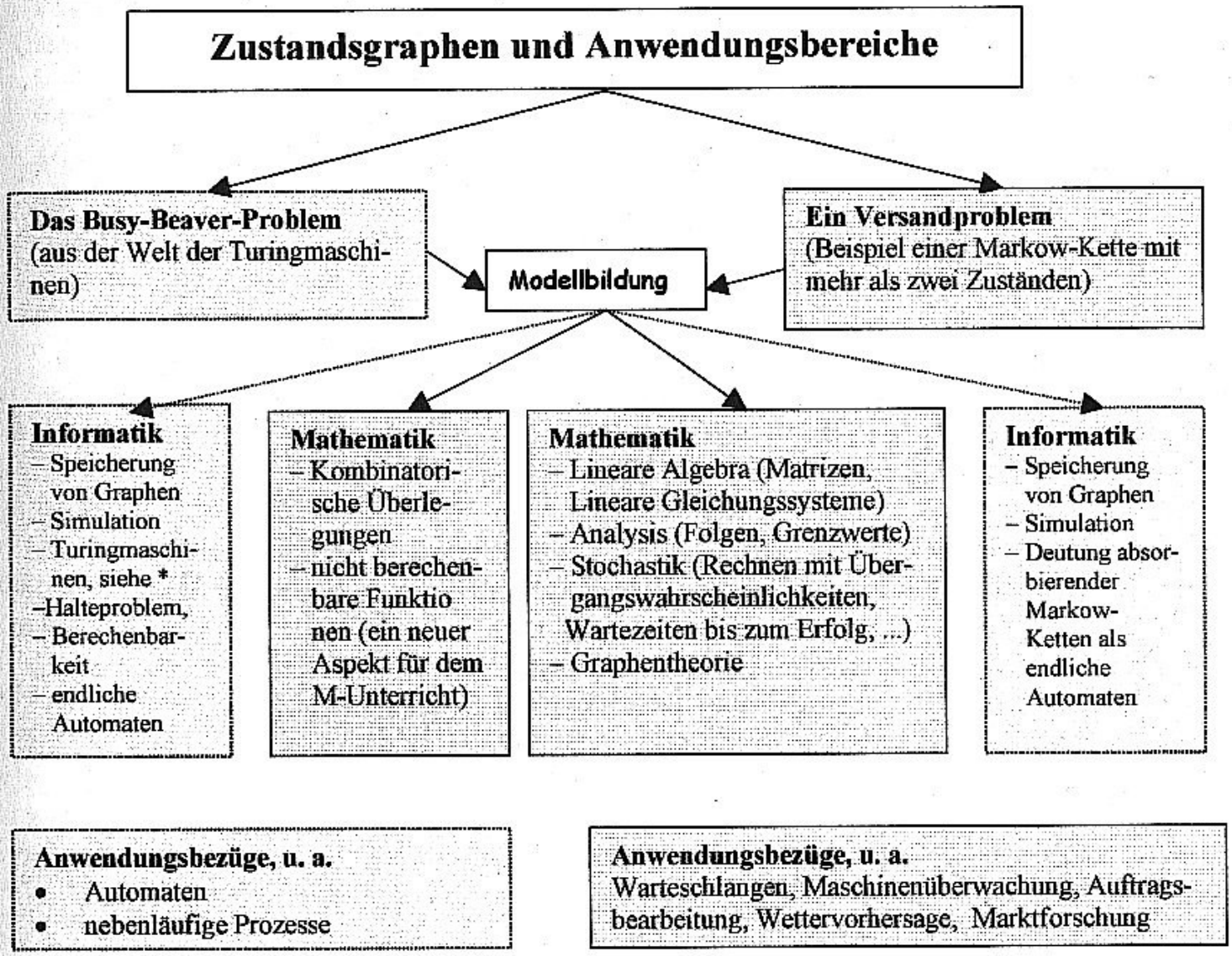


Abb. 3.3.1-d: Algorithmen und Anwendungen von Zustandsgraphen und verwandten Problemen im Mathematik- und Informatikunterricht

Zum Turingmaschinen-Simulator

1. Das Projekt 1997

Programm starten mit: tur-film.exe

2. Die Turingmaschine

Das Programm starten mit: turing-98.exe

Folgende Dateien verwenden:

- Beaver-c.tur (3 Zustände), Beaver-d.tur (4 Zustände)
- Leer.ban

1. Das Projekt 1997

Abituraufgabe zum Turing-Simulator

Mündliches Abitur mit Busy-Beaver

Abi98-hense.doc

Literaturhinweis

Busy-Beaver-Fang-diploma.pdf

Eine mögliche Fortsetzung oder auch ein anderer Einstieg in das Thema “nicht berechenbare Funktionen”eröffnet sich in der

Kryptologie – Sicherheit bei Codierungsverfahren usw.

Aber bezüglich theoretischer Überlegungen:

Stopp - und der Universität überlassen

HALT

sen, welche die Maschine auf ein anfangs leeres (mit Nullen gefülltes) Band schreiben kann, bevor sie anhält? Es steht außer Frage, daß diese Zahl, die wir als $\Sigma(n)$ bezeichnen, existiert, denn die Anzahl der Turing-Maschinen ist endlich.

Neben anderen Gründen ist es eine Aufgabe dann wert, gelöst zu werden, wenn sie schwierig ist. Das Busy-beaver-Problem kann allgemein mit einem Computer nicht gelöst werden, da die Funktion $\Sigma(n)$ schneller als jede berechenbare Funktion $f(n)$ wächst. Um erkennen zu können, daß dies zutrifft, nehmen wir an, daß B eine Turing-Maschine ist, welche die Busy-beaver-Funktion $\Sigma(n)$ berechnet und daß B q Zustände besitzt. Dies bedeutet, daß B, wenn sie mit einem Eingabeband konfrontiert wird, auf dem die ganze Zahl n geschrieben ist, die Zahl $\Sigma(n)$ ausgibt. Ohne Verlust der Allgemeingültigkeit können wir annehmen, daß sowohl n als auch $\Sigma(n)$ binär geschrieben sind.

Nun sei B_n eine Turing-Maschine, die n Zustände besitzt und $\Sigma(n)$ Einsen ausgibt, bevor sie anhält; C sei eine Maschine, die eine binäre in eine unäre Zahl umwandelt, und A sei eine Maschine, die ein leeres Band in ein anderes umwandelt, auf das die Zahl n binär geschrieben ist. Fügt man die drei Maschinen zusammen, können sie durch ABC dargestellt werden. Hierbei handelt es sich um eine einzige Maschine, die mit einem leeren Band beginnt und mit $\Sigma(n)$ Einsen auf dem Band anhält. Außerdem besitzt sie nur $[\log n] + q + r$ Zustände, da $[\log n]$ Zustände von A benötigt werden, und eine konstante Anzahl von Zuständen, q und r, für B bzw. C erforderlich sind.

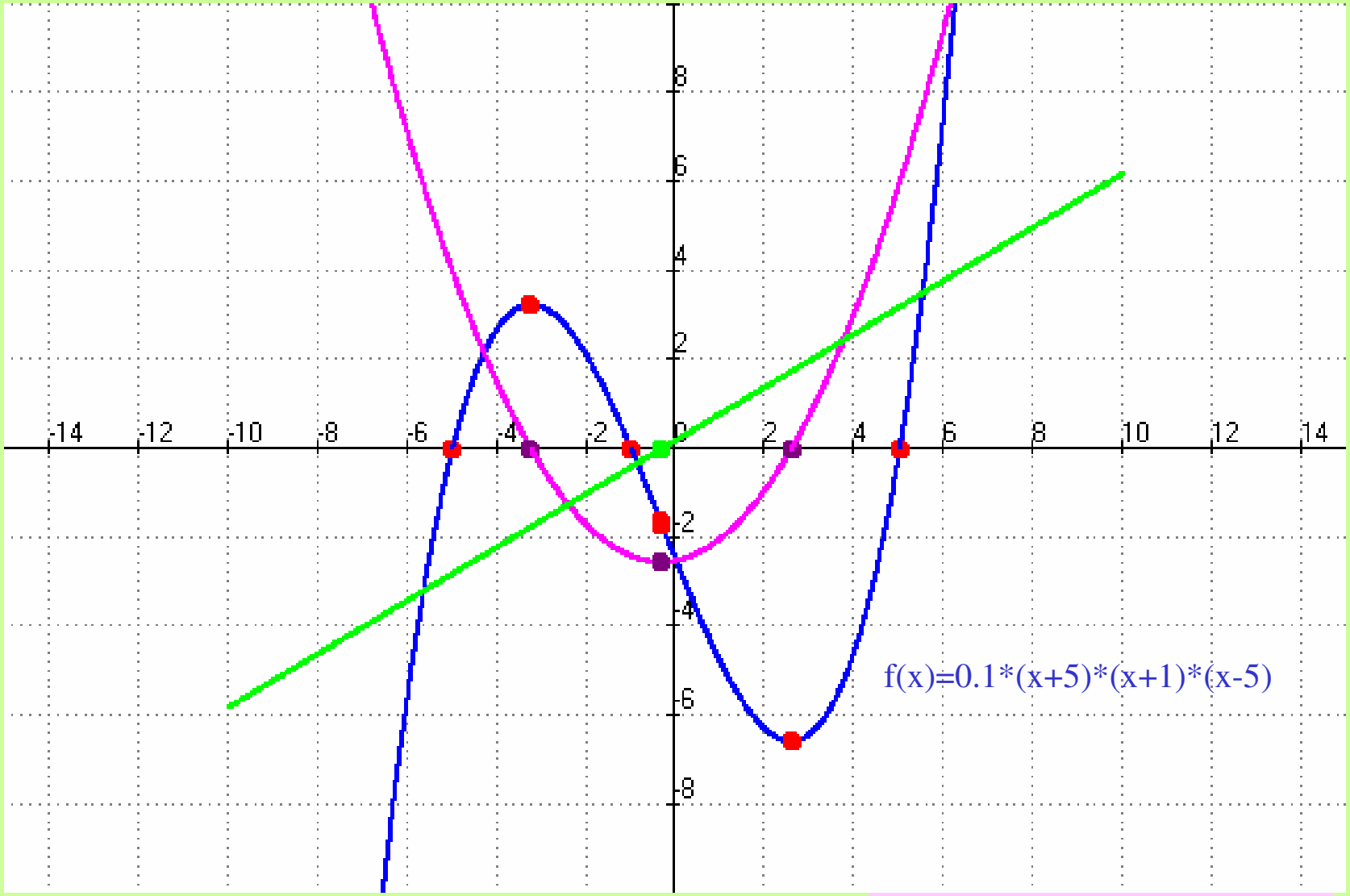
Für jede ganze Zahl n, so daß gilt $n > [\log n] + q + r$, schreibt Maschine ABC genauso viele Einsen wie B_n und benutzt trotzdem weniger Zustände! Da es jedoch leicht zu zeigen ist, daß $\Sigma(n) > \Sigma(m)$, wenn $n > m$, erhält man einen offensichtlichen Widerspruch. Da die Maschinen A und C eindeutig existieren, kann B nicht existieren. Entsprechend ist $\Sigma(n)$ keine berechenbare Funktion.

Die nachfolgende Tabelle faßt unseren augenblicklichen Wissensstand über die Werte von $\Sigma(n)$ zusammen:

Aus: Der Turing-Omnibus

Dewdney, 1995

Nullstellen suchen: Automatische Kurvendiskussion



Inf-Suchen-Kurvendiskussion1.pl2
Plot2.exe

f1: 0.1*(a+5)*(a+1)*(a-5) **// Ausgangsfunktion**
f2: {abs(f1(a))<0.01:a} // Nullstellen suchen
f3: f2(a),f1(a) // Nullstellen markieren

f5: (f1(a+0.001)-f1(a))/0.001 **// 1.Ableitung (angenähert!)**
f6: {abs(f5(a))<0.01:a} // Nullstellen von f5 suchen
(Extremwerte von f1)
f7: f6(a),f5(a) // Nullstellen von f1´= f5 markieren
f8: f6(a),f1(a) // Extremwerte von f1 markieren
f9: x,0
f10: (f5(a+0.001)-f5(a))/0.001 **// 2.Ableitung (angenähert!)**
f11: {abs(f10(a))<0.01:a} // Nullstellen von f10 suchen
f12: f11(a),f10(a) // Nullstellen von f ´´
f13: f11(a),f5(a) // Extremwerte von f ´
f14: f11(a),f1(a) // Wendepunkte von f

Ziele - Informatik

Fokus liegt auf dem Suchen in einer Datei

Probleme beim Suchen in Zahlenmengen (andere Terme), Kontrollstrukturen

Andere Suchverfahren

Wie ist es bei einer Namensdatei?

Funktionale Programmierung (funktional, imperativ,...)

Entwurf eine Animation

Ziele Mathematik

Fokus liegt auf Funktionsuntersuchung

Auswerten einer Animation

Probleme beim Suchen nach Nullstellen, Extremwerten, Wendepunkten in einer Wertetafel (andere Funktionsterme)

Arbeit mit Ungleichungen

Funktionsterme, die Schwierigkeiten machen

Sollte man doch lieber rechnen?

Entwurf einer Animation