Fundamentale Ideen der theoretischen Informatik

Dr. Eckart Modrow

Max-Planck-Gymnasium
Theaterplatz 10
37073 Göttingen
emodrow@gmx.de

Abstract: Der folgende Artikel ist ein Auszug des dritten Abschnitts einer umfangreicheren Arbeit [Mo03b], deren ersten beiden Teile sich mit allgemein- und fachdidaktischen Fragen beschäftigen. Dort werden aus einer Diskussion z. B. des Allgemeinbildungsbegriffs bei Bussmann und Heymann [BH87] aus allgemein-didaktischer Sicht Kriterien für Unterrichtsinhalte abgeleitet, die als Kandidaten für Bausteine einer Unterrichtsfolge Informatik in der Sekundarstufe II infrage kommen. Im Artikel soll gezeigt werden, wie ein an fundamentalen Ideen [Sc93] ausgerichteter Unterricht der theoretischen Informatik gestaltet werden kann, um diesen Kriterien zu genügen. Inhalte der Kerninformatik werden mit aktuellen Werkzeugen und vor allem anhand aktueller Beispiele unterrichtet, die einen motivierenden Kontext schaffen. Aus allgemeindidaktischen Überlegungen, aus diesem Kontext sowie aus den benutzten Unterrichtsmethoden folgen die Akzentuierung, mit der, und der Zeitrahmen, in dem die Fachinhalte unterrichtet werden.

1 Zur Theorie im Informatikunterricht

Bevor der Eleganz theoretischer Einsichten gehuldigt wird, sollte man sich klarmachen, dass in der Schule die theoretischen Anteile zu den unbeliebtesten überhaupt gehören. Ein (derzeit meist noch) reines Wahlfach wie die Informatik läuft deshalb ein hohes Risiko, wenn es den Anteil theoretischer Inhalte zu erhöhen versucht. Es muss sich also sehr genau klarmachen, was es bewirken will – und kann –, bevor es sich darauf einlässt. Da Wahlfächer für die Lernenden attraktiv sein müssen, muss die vorhandene Attraktivität der Schulinformatik auch bei einer Erhöhung des Theorieanteils erhalten bleiben[1]. Ich meine, dass dieses auch problemlos möglich ist, weil die Lernenden eher an den Werkzeugen und Anwendungsbeispielen, vor allem aber am eigenen Tun interessiert sind als an den Fachinhalten. Behandeln wir also auch die Theorieanteile in einem Kontext, der selbständige Arbeit an motivierenden Problemstellungen ermöglicht, dann steht einer stärkeren Theoretisierung nichts im Wege.

[1] Das gilt natürlich auch für Pflichtfächer, denn auch diese müssen die Möglichkeiten des Faches ausschöpfen.

2 Fundamentale Ideen (FI) der theoretischen Informatik

Die FI müssen nach Andreas Schwill dem Horizontal-, Vertikal-, Zeit- und Sinnkriterium genügen [Sc93]. Er kommt durch deren Anwendung – ausgehend von drei „Masterideen“ – zu einer großen Zahl von FI, die sich übersichtlich in drei Bäumen anordnen lassen. Die eingehende Diskussion dieses Ansatzes in [Mo03b] legt es nahe, den dritten Baum so zu erweitern, dass er die theoretischen Anteile der Informatik deutlicher hervorhebt. Wir kommen so zu einem veränderten „Ideenbaum“, der für das hier betrachtete Thema wesentlich ist.

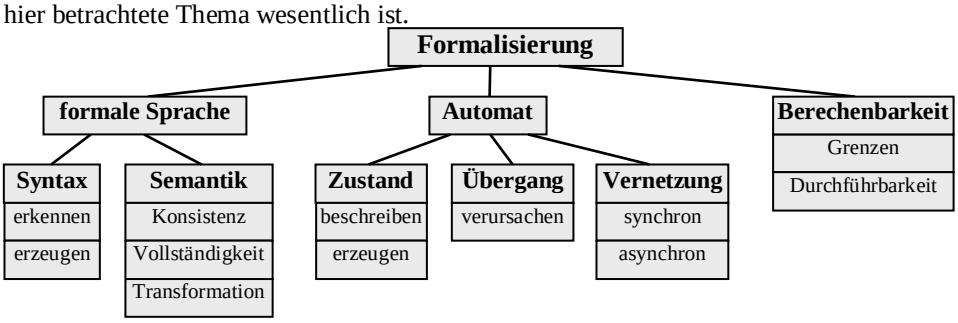


Abb. 1: Zur fundamentalen Masteridee „Formalisierung“

Zentral für die theoretische (Schul-)Informatik scheinen den meisten Autorinnen und Autoren die informatikspezifischen Maschinenmodelle (Automaten) zu sein, die im Gegensatz zu den statisch beschriebenen Zusammenhängen in mathematischen Formulierungen das dynamische Verhalten eines Systems als Prozesse modellieren. Der momentane Zustand des Systems wird gespeichert und durch Zustandsübergänge verändert, die durch Eingaben ausgelöst werden. Ggf. kann der Automat auch Ergebnisse in Form von Ausgaben produzieren. *Die zentrale fundamentale Idee dieses Gebiets scheint mir deshalb die des Zustands zu sein, wobei sie sich im Modell des Automaten manifestiert.* Seine Anschaulichkeit macht dieses Modell besonders für die Schule geeignet. Es findet in der Fachwissenschaft auf sehr unterschiedlichen Gebieten und sehr unterschiedlichen Niveaus Anwendung, wird seit den Anfängen der Informatik benutzt und besitzt gerade durch seine Anschaulichkeit eine lebensweltliche Bedeutung, weil sich Aspekte sehr unterschiedlicher im Alltag benutzter Systeme und eben der Computer selbst auf diese Idee reduzieren lassen.

Damit kommen wir zum Bereich der Eingaben, der „Bedienung“ der Maschinen. Traditionell werden diese durch Eingabebänder beschrieben, die mit den geplanten Eingaben vorab beschrieben werden, also bevor die Maschine zu arbeiten beginnt. Diese Zusammenfassung von Eingabe und Maschine zu einer Einheit ist notwendig für die Arbeitsweise von Turingmaschinen, da die das Eingabeband selbst manipulieren. Die einfacheren Maschinenmodelle (endliche Automaten und Kellerautomaten) können m. E. durchaus getrennt vom Eingabeband als Entitäten aufgefasst werden, so dass diesen die Einschränkung der „Vorabeingabe“ nicht eigen ist. Sie müssen nur zu jedem Arbeitstakt über ein zulässiges Eingabezeichen verfügen. Nach welchen Regeln dieses produziert wird, ob es vielleicht sogar von Ausgaben anderer Automaten stammt, wie in Netzen üblich, darüber ist erst mal gar nichts gesagt. Erst die gelesene Eingabefolge

muss gewissen syntaktischen Regeln genügen. Damit entfällt für diese Automatenklassen, die für die praktische Arbeit wichtig sind, weitgehend auch der Einwand, dass Turingmaschinen kein adäquates Modell für „interagierende“ und „kommunizierende“ Computer(systeme) seien. Einfache ereignisgesteuerte Systeme lassen sich sehr wohl auf einfache Automaten abbilden. Obwohl die Arbeitsweise der traditionellen Automatenklassen sequentiell ist, kann die Wirkung von Interaktionen leicht über die Kopplung solcher Maschinen erfahren werden. Ordnet man einfache Automaten in einem Netz als zellulären Automat an, dann können unter geeigneten Bedingungen ganz neue Verhaltensweisen erscheinen. Auch die Modellierung von OOP-Systemen, deren Objekte meist gut als Automaten zu beschreiben sind, die mithilfe von Events kommunizieren, liefert über die nicht sequenziell ablaufende Ereignissteuerung Erfahrungen in „Vernetzung“. Ich halte also die Automatenmodelle gerade auch dann für geeignet, wenn wie heute (hoffentlich) üblich mit objektorientierten Sprachen gearbeitet wird, *wobei als zweite fundamentale Idee die Vernetzung zum Tragen kommt.*

Statt uns auf die Arbeitsweise der Maschine selbst zu konzentrieren, betrachten wir nun deren Steuerung. Die Syntax der von einem Automaten akzeptierten Zeichenfolgen klassifiziert diesen ebenso wie seine innere Konstruktion. Die Verlagerung des Blickpunkts bewirkt einen Wechsel des Modells, ohne das beschriebene System selbst zu ändern. Interessieren wir uns eher für die Konstruktion des eigentlichen „Apparats“, z. B. eines Parsers oder einer Schaltung, dann wählen wir das Automatenmodell. Wollen wir eher dessen „Drumherum“ beschreiben, dann arbeiten wir mit Grammatiken. Bei Bedarf wechseln wir zwischen diesen Beschreibungen, z. B. weil im alternativen Modell bessere Werkzeuge zur Verfügung stehen. *Damit haben wir als dritte fundamentale Idee die Sprache gefunden.* Die Idee, Kommunikationsmittel syntaktisch zu beschreiben, gehört im Bereich der Schule zum Standard. Die Analyse von Sprachkonstrukten mithilfe von Grammatiken ist zwar mit der Verdrängung der alten Sprachen etwas in den Hintergrund geraten. Das ändert aber nichts am Wert der Erfahrung, formale Regelsysteme systematisch einzusetzen und deren Ergebnisse kritisch nach semantischen Gesichtspunkten zu bewerten.

Es bleibt als Letztes die Idee der Berechenbarkeit. Dass Computer rechnen können, ist sicherlich für niemanden eine Überraschung. Berechnungen gehörten schließlich zu den ersten Aufgaben des „Rechners“. Ansatzpunkte aus der Erfahrungswelt der Lernenden gibt es dafür genug. Unter Berechenbarkeit verstehen wir allerdings mehr die Frage danach, was alles berechnet werden kann, also Betrachtungen über die Grenzen der Computer in dieser Hinsicht. Zu diesen Grenzen gehören sowohl die Frage, ob es Grenzen für algorithmische Verfahren gibt, als auch, wo die Grenzen der Gültigkeit der Ergebnisse von Berechnungen liegen. Beide Themen durchziehen die Informatik von Anfang an, spielen auf unterschiedlichen Ebenen und in verschiedenen Bereichen eine zentrale Rolle. Beide Themen liefern aber auch sowohl sehr tiefgehende Fragen (und einige Antworten) als auch eine Fülle außerordentlich motivierender Beispiele, etwa aus den Bereichen des deterministischen Chaos oder der Simulation vernetzter Systeme.

3 Zur Rekonstruktion der fundamentalen Ideen

Beschränken wir uns auf die genannten fundamentalen Ideen der Informatik, so ist noch nicht geklärt, wie sich diese bei den Unterrichteten denn bilden sollen. Im Sinne des Konstruktivismus ist zu fordern:

- Einerseits müssen die individuellen Vorstellungen der Lernenden berücksichtigt und zielgerichtet weiterentwickelt werden,
- andererseits erfordert die Ausbildung veränderter mentaler Strukturen die aktive Auseinandersetzung mit Fragestellungen des bearbeiteten Bereichs auf möglichst vielfältige Weise.

Beginnen wir mit den mentalen Modellen: Eine diffuse Vorstellung von **Automaten** als Beispielen von *„(1) Maschinen, die etwas tun“*, hat vermutlich jeder. Das Tun beinhaltet Aktivität, also Dynamik. Wenn das Tun nicht nur aus einer einzigen Aktion besteht, dann kann das Anfangsmodell weiterentwickelt werden zu *„(2) Maschinen, die schrittweise etwas tun“*. Im einfachsten Fall wird dieses Nacheinander der Aktionen durch eine Art Zeittakt ausgelöst werden. Damit benötigen wir *„(3) Maschinen, die wissen, was als nächstes zu tun ist“*. Diese Information muss in den Maschinen vorhanden sein, dort gespeichert werden. Nach jeder Aktion muss sich damit diese Information ändern. Wir können sagen, dass wir es nun mit *„(4) Maschinen, die sich in unterschiedlichen Zuständen befinden können“* zu tun haben. Damit legt der **Zustand** fest, was als nächstes zu tun ist. Wir haben eine sequenziell arbeitende Maschine, die wir durch eine Folge von Zuständen und **Übergängen** wie üblich beschreiben können. Erweitern wir das Modell um einzugebende Steuerzeichen, dann können unsere Maschinen aus einem Zustand in unterschiedliche Folgezustände übergehen. Wir kommen zu den Transitionsgraphen.

Diese kurze Folge von Modellen will nun konstruiert sein. Dazu benötigen wir möglichst vielfältiges und möglichst interessantes Aufgabenmaterial, das an die Erfahrungswelt der Schülerinnen und Schüler anknüpft. Da es sich bei den hier zugrunde gelegten endlichen Automaten meist um ziemlich einfache Maschinen handelt, brauchen wir dafür keine besondere Unterrichtseinheit. Im Gegenteil: Weil die Automaten sich so leicht aufzeichnen lassen, sollten sie als Hilfsmittel innerhalb von umfangreicheren Problemstellungen auftauchen:

- Schon ganz am Anfang der Kursfolge, wenn Zeichenketten modifiziert, durchsucht, verändert werden, z. B. bei Verschlüsselungsproblemen.
- In Unterrichtsprojekten etwa aus der Bioinformatik, wenn z. B. DNA-Replikation durch Polymerasen simuliert wird.
- Zur Beschreibung von Lichtschranken, Bahnübergängen, Alarmanlagen, …
- Zur Entwicklung von Schaltwerken wie Speichern, Addierern, …

Die übergreifende Einsetzbarkeit von Automatenmodellen lässt die Lernenden deren Fundamentalität erfahren. Die aktive Nutzung macht sie mit deren Möglichkeiten, Tücken und Grenzen vertraut. Die unterschiedliche Realisierung über Funktionen, OOP-Objekte und Schaltungen verdeutlicht deren Modellcharakter und ihre Brauchbarkeit als Werkzeug. Die solide mentale Verankerung dieser Modelle bietet dann den Zugang zu den weiteren fundamentalen Ideen des Theorieteils.

Verlagern wir nun unser Interesse auf die Bedienung von Automaten, die auf unterschiedliche Eingabezeichen reagieren können, so kommen wir auf die Frage nach deren Steuerbarkeit. Es gibt Eingabefolgen, die zu unsinnigen Ergebnissen führen, und solche, die einen angestrebten Zweck erfüllen. Wahrscheinlich gibt es auch unterschiedliche Zeichenfolgen, die zum gleichen Ergebnis führen. Diese bilden **Sprachen**, deren Worte etwas bewirken. Die Idee der Sprache ist den Lernenden natürlich vertraut. Neu für sie ist die Anwendung dieses Begriffs auf die Kommunikation mit Maschinen, vor allem die Anwendung auf so einfache Konstrukte, wie wir sie meist behandeln. Da wir es noch nicht mit Programmiersprachen zu tun haben, halten die Unterrichteten es für absonderlich, kurzen Befehlsfolgen das Attribut „Sprache“ zuzuschreiben. Sie brauchen deshalb Zeit zur Gewöhnung, vor allem daran, Sprachen rein formal zu behandeln. Sie finden diese Zeit in der konstruktiven Auseinandersetzung mit entsprechenden Problemen.

Die **Struktur** dieser Sprachen kann durch Grammatiken beschrieben werden, wobei die Zugehörigkeit eines Wortes zu einer Sprache diesem eine **Bedeutung** gibt, wenn man sie auf den Zielautomaten anwendet. Die Schülerinnen und Schüler können nun im selben Kontext wie oben, aber mit einer veränderten Sichtweise Automaten steuern. Sie können geeignete Befehlsworte erzeugen, solche mithilfe von Parsern (also „Prüfautomaten“) testen, die Ergebnisse simulieren und erst dann dem „echten“ Zielautomaten zuführen. Ein außerordentlich motivierender Kontext hierfür ist die Beschäftigung mit kleinen Robotern, aber auch Technikmodelle, deren Motoren durch Relais geschaltet werden, Turtlegrafik-Umgebungen, Sprachspiele („Zufallsgedichte“, „Elisa“, …) sind beliebt. Bleiben die Automaten und ihre Sprachen nur Hilfsmittel, so sind auch diese Teile in vorgelagerte Kurse integrierbar. Beschäftigen wir uns systematisch mit ihnen, dann sind eigene Unterrichtseinheiten etwa zum Thema „Compilerbau“ erforderlich.

Die Idee der **Vernetzung** wird besonders deutlich, wenn wir die vernetzten Komponenten durch Automatenmodelle beschreiben. Die Kommunikation erfolgt durch die Verknüpfung der Ein- und Ausgabekanäle der Maschinen – ein sehr anschauliches Modell. Setzen wir die Automaten in ein festes Gitter, dann erhalten wir zelluläre Automaten mit all ihren vielfältigen und auch optisch interessanten Einsatzmöglichkeiten. Verknüpfen wir sie durch frei zu sendende Botschaften, dann haben wir ein Modell für Teilbereiche der OOP. Bilden wir sie auf die Knoten eines Netzes ab, dann finden wir Zugang z. B. zu den Protokollen des Internets. Gerade in diesem Bereich bieten die aktuellen Programmiersprachen für die Schule neue und relativ einfach realisierbare motivierende Möglichkeiten.

Die Idee der **Berechenbarkeit** ist, wenn man sie so wie in der theoretischen Informatik üblich auffasst, für Lernende neu und fremd. Hier halte ich entsprechende Einsichten nur als Endergebnis des Lernprozesses für möglich. Der Weg von Alltagserfahrungen hin zu Entscheidbarkeitsproblemen ist zu weit, um ihn weitgehend durch Eigenaktivitäten zu finden. Es ist aber möglich, in Eigenarbeit so viel Erfahrungen mit den Bausteinen dieses Weges (Codierungen, Turingmaschinen, …) zu machen, dass eine geschlossene Darstellung mit ihren verblüffend weittragenden Aussagen machbar wird. Die numerischen und durch Iterationen und Kombinationen auftretenden Grenzen sind

allerdings nahe liegend und durch Experimente direkt erfahrbar. Sie bieten ebenso wie die zellulären Automaten hochinteressante Probleme schon für den Anfangsunterricht.

4 Kriterien für Unterrichtsinhalte

Die Inhalte eines Theoriekurses sollten nicht nur aus fachimmanenten Überlegungen abgeleitet, sondern auch allgemein bildenden Aspekten gerecht werden. Dafür müssen sie – wie in [Mo03b] gezeigt – entsprechenden Kriterien genügen. Sie sollten m. E.

- ***formale Bildung ermöglichen***

Unterrichtsinhalte sollten den Prozess der Begriffs- und Modellbildung betonen, in dem ausgehend von konkreten Beispielen zielgerichtet abstrahiert wird, um übergreifende Eigenschaften zu finden und hantierbar zu machen. Die Anwendung formaler Verfahren und der Wechsel zwischen Modellen, etwa beim Entwurf, die enge Verknüpfung von Theorie und Praxis und die Rückkoppelungen zwischen diesen müssen deutlich werden. An wenigen, aber geeigneten Stellen muss im Theoriekurs die herausragende Stellung der Mathematik hilfreich in Erscheinung treten.

- ***anhand fundamentaler Ideen ausgewählt werden***

Aspekte der Algorithmisierung müssen in den Entwurfsverfahren und bei den Anwendungen (Simulationen, Realisierung von Modellen, …) herausgearbeitet werden. Modularisierung und Hierarchisierung sind fast allen Aufgabenfeldern aus diesem Gebiet immanent. Die Ideen des Formalisierungs-Baums müssen deutlich zu Tage treten.

- ***einen Bezug zu Schlüsselproblemen haben***

Hier muss der Bezug einerseits über die Auswahl der Beispiele erfolgen (Anwendungen der Codierung, Anwendungen und Grenzen der formalen Sprachen, Komplexität von zellulären Automaten, …), andererseits halte ich den Weg zu Grenzfragen im Bereich der Berechen- und Entscheidbarkeit für so wichtig, dass er auch inhaltlich den Kurs rechtfertigt.

- ***Transfer einüben***

Es dürfen nicht nur einzelne Beispiele „besprochen“ werden, sondern die entwickelten Methoden müssen in einem Themenfeld selbstständig auf modifizierte Beispiele angewandt werden und so Werkzeugcharakter gewinnen. Im Bereich Technik/Theorie bieten sich für sehr ähnliche Verfahren drastisch unterschiedliche Anwendungsbereiche, so dass die Übertragbarkeit sehr deutlich wird. Diese Möglichkeit muss genutzt werden.

- ***Modellbildung ermöglichen***

Die Erzeugung gültiger Modelle für Informatiksysteme ist eine der Hauptaufgaben des Technik-/Theorieteils der Kursfolge. Der Unterricht muss deshalb aus den vorhergehenden Kursen vorliegende Erfahrungen ordnen und daraus jeweils ein stark reduziertes, aber valides Hardwaremodell entwickeln, um sie „nach unten“ abzusichern, und ein Softwaremodell, das einerseits den Algorithmusbegriff präzisiert und auf seine Grenzen abklopft, anderseits die Grundfunktionen erklärt.

- ***Projektunterricht unterstützen***

Die Anwendungsmöglichkeiten der Modelle des Theorie-/Technikkurses sind so vielfältig, dass es schon fast fahrlässig wäre, auf projektartige Phasen zu verzichten. Einerseits können die gleichen Methoden auf unterschiedliche Aufgabenstellungen in Gruppen angewandt, dokumentiert und vorgestellt werden (jeweils in den Bereichen Steuern und Regeln, „Taschenrechner“, programmierbare Schaltungen, einfache Computersprachen, generative Grammatiken, Robotersteuerung, spezielle Turingmaschinen, …), andererseits können auch unterschiedliche (und damit unterschiedlich anspruchsvolle) Methoden bei ähnlichen Aufgabenstellungen Anwendung finden (Entwicklung und Parsen einer Sprache, Entwicklung einer Schaltung für bestimmte Aufgaben, …).

- ***und als Bausteine eines offenen Kanons dienen.***

Die technische und theoretische Informatik sind m. E. die Musterbeispiele für Themenbereiche, an denen sich dieses Kriterium demonstrieren lässt. Inhaltlich ändert sich hier kaum etwas, da in der Schule weder die Theorie noch die Technik an die aktuelle Entwicklung gekoppelt sind. Damit lassen sich die fachlichen Ziele „zeitlos“ festschreiben. Es ändern sich aber die Werkzeuge, die den Schulen zur Verfügung stehen. Wollen wir also z. B. die Modularisierung und Hierarchisierung am Beispiel von (Modell-)Rechnern behandeln, dann können wir das (fast zeitlos) mithilfe von TTL-Bausteinen erreichen. Die Simulation der Bauteile kann aber mit aktuellen Werkzeugen erfolgen. Ob hier als OOP-Anwendung selbst programmiert, ein spezielles Simulationsprogramm benutzt oder sogar mit Spreadsheets gearbeitet wird, bleibt der aktuellen Situation überlassen. In jedem Fall liegen Inhalte und Ziele fest, die konkrete Umsetzung ist offen.

Ein entsprechender Unterrichtsgang, der insbesondere den Zugang zu Berechen- und Entscheidbarkeitsproblemen beschreibt, findet man z. B. in [Mo92a] und [Mo92b].

5 Ein Beispiel: Gekoppelte Automaten

Das überraschende Verhalten gekoppelter Systeme kann mit zellulären Automaten sehr gut demonstriert werden[2]. Einerseits nutzt man hier ziemlich einfache endliche Automaten, so dass die Benutzung der entsprechenden Beschreibungsmittel geübt werden kann, andererseits sind die Resultate so vielfältig, dass die Arbeit trotz dieser Einfachheit alles andere als langweilig ist. Man kann leicht Systeme modellieren, deren Verhalten sich kaum aus dem Simulationsprogramm selbst erschließen lässt. Die Anordnung der Elementarautomaten in Gittern liefert ein exzellentes Beispiel für den Umgang mit zweidimensionalen Feldern. Verteilt man die Funktionalität auf geeignete Objektklassen, dann kann die Darstellung der Ergebnisse standardisiert werden, so dass Experimente an den Automaten ohne großen Programmieraufwand zu realisieren sind. Die Unterrichtseinheit ist je nach der Fortschrittlichkeit der eingesetzten Programmiertechniken im Unterricht der Sek. II anzusetzen. Der erforderliche

[2] Zahlreiche gut umsetzbare Beispiele finden sich z. B. in [GS95].

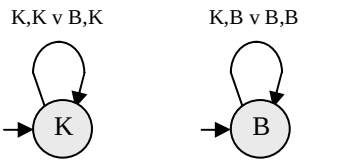
Zeitbedarf wird wesentlich dadurch bestimmt, ob – und wenn, welche – Teile des Simulationsprogramms vorgegeben werden.

Wir wollen einen zellulären Automaten bauen, der auf dem Gefangenendilemma aufbaut, aber etwas abgewandelt auf den Handel im Internet. Das Verhalten der Handelspartner wird durch endliche Automaten simuliert, die auf einem in beiden Dimensionen abgeschlossenen Gitter sitzen und innerhalb einer Moore-Nachbarschaft Handel mit den Partnern treiben. Sie tauschen – wie im Internet üblich – Waren gegen Geld. Dabei gibt es unterschiedliche Arten von Geschäftpartnern:

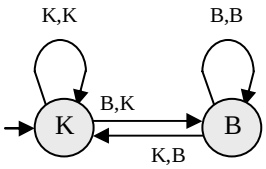
- Naive kooperieren immer, liefern also den korrekten Gegenwert.
- Betrüger kooperieren nie.
- Gewitzte kooperieren anfangs und reagieren danach so, wie der Partner zuletzt.

Hier ist die Idee des Zustands und seiner Wechsel zentral. Des Weiteren spielt aber auch die Idee der Berechenbarkeit in ihrem prognostischen Aspekt eine Rolle, da sich die Frage stellt, ob Voraussagen über das Verhalten des Systems möglich sind, ohne es vollständig zu realisieren, also wirklich „laufen zu lassen“. Die Implementierung über Objekte liefert Beispiele für strukturierte Zerlegungen und einfache Programmierkonzepte.

Wir können dieses Verhalten der Handelspartner durch Zustandsdiagramme beschreiben. Entsprechende, zufällig erzeugte Automaten ordnen wir in einem Gitter an und färben sie entsprechend ihrem Zustand (schwarz als Betrüger, weiß als Naiver und grau als Gewitzter): (K: „kooperieren“, B: „betrügen“)



Der „Naive“ Der „Betrüger“



Der „Gewitzte“

Abb. 3: Drei Strategien

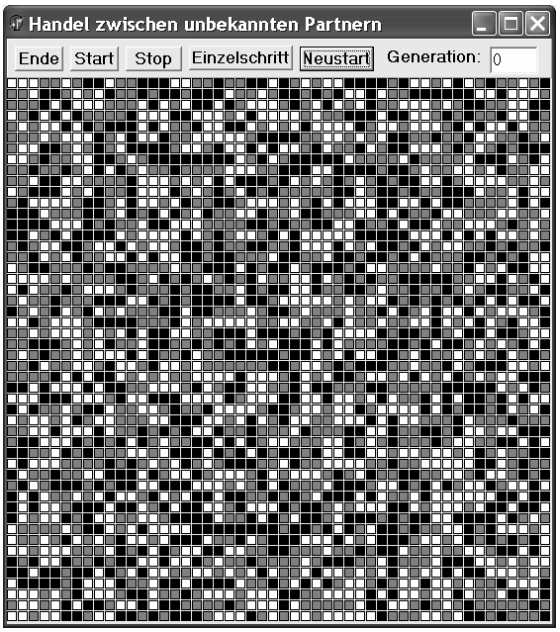


Abb. 2: Der Gitterautomat

Der weitere Ablauf ist einfach: Zuerst handeln alle Partner einmal mit ihren Nachbarn aus der Moore-Nachbarschaft. Dabei ist einiges an Orientierung im Gitter (als Array) vonnöten: es wechselt die „Blickrichtung“ (Stellung in der Nachbarschaft), und an den Rändern muss man auch überlegen. Danach bewerten alle Partner den Erfolg ihrer Nachbarn. Als Opportunisten übernehmen sie den Zustand des erfolgreichsten Nachbarn oder behalten ihren Zustand bei, wenn sie selbst besser waren. Im Beispiel wird eine Automatenklasse in einer eigenen Delphi-Unit vereinbart und darauf aufbauend eine „Welt“ (auch in einer Unit), die mit einem Gitter aus Automaten hantiert. Beides wird von der Programmoberfläche des zellulären Automaten gesteuert. Die Abläufe zwischen diesen Klassen sind zwar nicht ganz trivial, dafür aber nur einmal zu lösen. Danach können die Automaten in ihrer Unit bzw. die „Welt“ in der anderen getrennt und ohne direkte Beeinflussung manipuliert werden. Man kann „schön einfach“ experimentieren.
In den ersten Generationen setzen sich meist „die Bösen“ durch. Danach bilden sich Cluster aus „Guten“ bzw. „Gewitzten“, und dann beginnt eine wilde „Schlacht“. Zwar werden die „Guten“ hart von den „Betrügern“ bedrängt, sie halten sich aber in Gruppen. Die „Gewitzten“ setzen sich gegenüber den „Betrügern“ meist durch und kooperieren mit den „Guten“. Am Ende siegen meist die „Gewitzten“ – aber eben nicht immer.

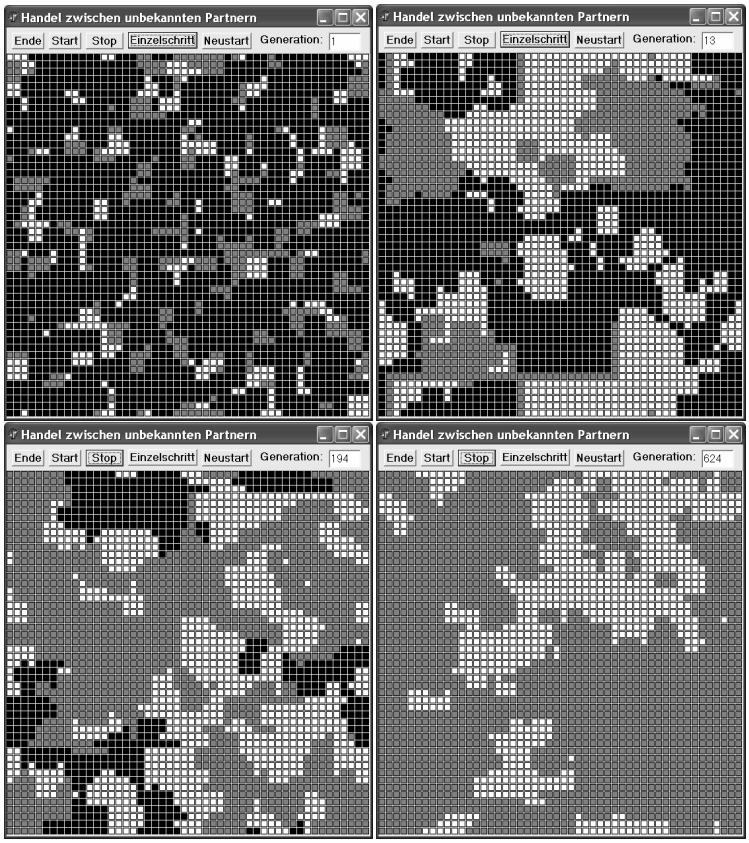


Abb. 4: Entwicklung des Gitterautomaten

Das Beispiel ist als Vorbereitung auf die Automatentheorie eher schlicht. Es bietet aber eine extreme Bandbreite von sowohl programmiertechnischen wie inhaltlichen Variationen und legt sozialwissenschaftliche bzw. naturwissenschaftliche Interpretationen nahe. In dieser Beziehung wird der Modellcharakter besonders deutlich.

Bleiben wir zuerst bei den Variationen: Das Verhalten der „Partner" kann leicht durch veränderte Strategien ergänzt werden. „Wettbewerbe" zwischen verschiedenen Strategien werden möglich, wobei Statistiken geführt werden müssen, da der Einzelfall nicht viel über das Gesamtverhalten aussagt. Schon hier ist systematisches Arbeiten gefragt. Die „Welt" kann verändert werden durch veränderte Nachbarschaften (von-Neumann-Nachbarschaft, andere Reichweiten, …), andere Abläufe, ... Statt des zweidimensionalen Gitters kann die zeitliche Entwicklung linearer Automaten in der üblichen Weise dargestellt werden, wobei chaotische Vorgänge auftreten ([Sc91] ab S. 388, [Ma91] S. 436). Bei beidem können sowohl die Art der Automaten wie das Steuerprogramm völlig ignoriert werden. Die Gewichtungsfaktoren, die den Gewinn bei unterschiedlichen Vorgängen bestimmen, sind veränderbar. [3] Auch hier sind die anderen Größen irrelevant. Das Steuerprogramm kann verbessert werden, z. B. um während des Programmlaufs die unterschiedlichen Faktoren zu verändern. Dafür sind „Oberflächenprogrammierer" gefragt. Die Eigenschaften der Gitterautomaten sollten auch per Mausklick gesetzt werden können (einige „Gewitzte" in einer Welt aus „Bösen", …), um das Verhalten bestimmter Konfigurationen gezielt untersuchen zu können.

Es kann aber auch versucht werden, die beobachteten Vorgänge systematisch zu bewerten. Dazu sind globale Größen geeignet („Bruttosozialprodukt" als Summe aller „Handelspunkte"). Der Einfluss der Parameter auf das Erreichen und die Art des ggf. erreichten Endzustands kann abgeschätzt werden. Lineare Automaten lassen sich entsprechend klassifizieren (z. B. [GS95] S. 59 nach Stephen Wolfram). Es können Beschränkungen eingeführt werden („Anzahl der handelbaren Güter", „Geldmenge", …), und die Verteilung der Größen auf die Gruppen sowie deren zeitliche Entwicklung ist darstellbar.

Die Beobachtung der manchmal überraschenden Abläufe liefert Ansatzpunkte zur Diskussion ethischer Fragen. Auch wenn das Beispiel natürlich nicht direkt auf gesellschaftliche Systeme übertragbar ist, so haben wir doch ein für die meisten neuartiges Argument für kooperatives, soziales Verhalten gefunden, das nicht aus transzendenten oder philosophischen Überlegungen gewonnen wird, sondern aus Effizienzbetrachtungen. Es steht darin in klarem Gegensatz zur Egozentrik des Primitivdarwinismus, der oft die öffentliche Diskussion in dieser Hinsicht beherrscht.

Unsere zellulären Automaten sollten den Handel im Internet simulieren. Das erhaltene Modell ist aber auch ganz anders interpretierbar: Betrachten wir den „Handel" als Energieaustausch benachbarter Teilchen, dann kommen wir recht schnell zum Ising-Modell für Spingitter ([AFH94] S. 405 ff). Die globalen Größen „Temperatur" und „Magnetisierung" liefern Bewertungsmaßstäbe zur Beurteilung des Systems. Relativ

[3] Im vorgestellten Programm wurde dafür etwas „Zufall" eingebaut.

kleine Änderungen führen auf Strukturbildungsprozesse und/oder Modelle für biologische und chemische Systeme ([Cr88]).

Die Interpretation desselben Modells in unterschiedlichem Kontext, die (teilweise) Unvorhersagbarkeit der Ergebnisse, das dynamische Verhalten und sein Bezug zu nichtlinearen Systemen liefert Erfahrungen mit Modellen, Simulationen, deren Mächtigkeit und deren Grenzen – und das fast ohne Mathematik. Die Visualisierungsmöglichkeiten der Computer machen so einerseits der Schule völlig neue Gebiete zugänglich und verdeutlichen andererseits den von der Mathematik unterschiedenen Charakter der Informatik als „Prognosesystem". Nebenbei ist das Thema eine unerschöpfliche Quelle von „Facharbeiten" der Schülerinnen und Schüler.

Haben wir als Lerngruppe z. B. den ersten Leistungskurs[4] der Stufe 12, dann sollte der mit der elementaren Algorithmik, primitiven Datentypen und einfacher Computergrafik halbwegs vertraut sein. Beginnen wir also eine Unterrichtseinheit über „Visualisierung großer Datenmengen", dann steht uns einerseits das weite Feld der Falschfarbendarstellungen von astronomischen, geografischen oder medizinischen Daten mit ihren Interpretationsmöglichkeiten offen (z. B. aus dem Projekt „Hands-On Universe", Satellitenbildern oder NMR-Daten der Tomografie), andererseits bilden Gitterautomaten ein interessantes Arbeitsgebiet, das hier betrachtet wird. Repräsentieren wir die Teilautomaten durch Objekte, dann können die entsprechenden Klassen aus grafischen Komponenten der GUI abgeleitet werden. Wir haben damit neben einem elementaren Zugang zu endlichen Automaten auch einen einfachen Einstieg in OOP-Methoden gefunden. In der angegebenen Form erfordert die Nutzung der Transitionsgraphen kaum Zeit: Bei einer Diskussion unterschiedlicher Strategien, z. B. der des „Langmütigen", wird diese Notationsform nebenbei eingeführt und gefestigt. Erheblich mehr Zeit benötigt eine eingehende Analyse der Abläufe, die daraus folgende Verteilung der Informationen und ihre Repräsentation. Wird hier nicht sorgfältig gearbeitet, dann kann es später erhebliche Probleme geben. Sind die Alternativen und deren Konsequenzen betrachtet und die wesentlichen Entscheidungen getroffen, dann werden die Teilprobleme angegangen:

- Eine Automatenklasse mit den erforderlichen Methoden wird vereinbart, implementiert und an einzelnen Objekten getestet.
- Ein Gitter solcher Automaten wird als Array definiert.
- Die Aktionen im Gitter werden realisiert.

Ziel des Unterrichts ist es, bei der Implementierung eines relativ einfachen Modells elementare informatische Techniken kennen zu lernen, sowie sich mit Simulationen zu beschäftigen, deren Ergebnisse kaum prognostizierbar sind und die zu Diskussionen anregen. Unter diesen Voraussetzungen wird eine anfängliche eingehende Erörterung der Problematik im Unterrichtsgespräch unbedingt erforderlich sein. Aus dieser sollten sich dann die zu lösenden Teilprobleme ergeben, und aus diesen folgen die erforderlichen Programmiertechniken. Die Vereinbarung einer Tochterklasse z. B. von GUI-Panels, die mit Automateneigenschaften ausgestattet wird, sollte ebenfalls gemeinsam durchgeführt

[4] Hier wurde die Unterrichtseinheit durchgeführt.

werden, wobei der Neuigkeitswert nicht bei den in Methoden auftretenden Algorithmen, sondern in den Zugriffstechniken liegt. So etwas lässt sich schnell abhandeln, die notwendigen Erfahrungen werden bei der Anwendung gewonnen. Sind zweidimensionale Felder schon bekannt, dann können die Lernenden das Zusammenspiel der Automaten selbst realisieren. Die dabei möglichen Fehler sollten erkannt und korrigiert werden. Als Hilfe wurde in der durchgeführten Unterrichtseinheit nur das zufällige Erzeugen einer Anfangsbelegung des Feldes vorgegeben. Danach wurde das Modell wie beschrieben variiert und erprobt.

Literaturverzeichnis

[AFH94] Argyris, J.; Faust, G.; Haase, M.: Die Erforschung des Chaos. Vieweg, 1994.

[BH87] Bussmann, H.; Heymann, H.-W.: Computer und Allgemeinbildung. Neue Sammlung 1, 1987.

[Cr88] Cramer, Friedrich: Chaos und Ordnung, die komplexe Struktur des Lebendigen. Deutsche Verlagsanstalt, 1988.

[GS95] Gerhard, M.; Schuster, H.: Das digitale Universum - Zelluläre Automaten als Modelle der Natur. Vieweg, 1995.

[Ma91] Mandelbrot, B.: Die fraktale Geometrie der Natur. Birkhäuser, 1991.

[Mo92a] Modrow, E.: Zur Didaktik des Informatik-Unterrichts, Band 2. Dümmler, 1992.

[Mo92b] Modrow, E.: Automaten-Schaltwerke-Sprachen. Dümmler, 1992.

[Mo02] Modrow, E.: Informatik mit Delphi - Band 1. emu-online, 2002, www.emu-online.de

[Mo03a] Modrow, E.: Informatik mit Delphi - Band 2. emu-online, 2003, www.emu-online.de

[Mo03b] Modrow, E.: Pragmatischer Konstruktivismus und Fundamentale Ideen als Leitlinien der Curriculumentwicklung, Dissertation 2003
http://alpha.uni-sw.gwdg.de/~emodrow/dissertation/dissert.pdf

[Sc91] Schroeder, M.: Fraktale, Chaos und Selbstähnlichkeit. Spektrum, 1991.

[Sc93] Schwill, A.: Fundamentale Ideen in Mathematik und Informatik. Script, 1993.