Klaus Prätor

Logik und Handlung im Computer

Einleitung

Der Computer ist für die Informatik nicht primär ein naturwissenschaftlicher Gegenstand. Vielmehr spricht diese von ihm in einem Paradigma von Sprache und Handlung. Das beginnt schon auf der Hardware-Ebene, wo die Elektronik mittels Logikschaltungen realisiert wird, Diese Schaltungen werden mit Hilfe des so genannten Quine-McCluskey-Verfahren minimiert, womit ein Sprachphilosoph und Logiker als Namensgeber Eingang gefunden hat.
Noch deutlicher wird das Paradigma auf der Ebene der Programmierung und Datenbanken, wo es sich in „Befehlen“ und „Datensätzen“ zeigt, was bei genauem Hinsehen keine zufälligen und als solche entbehrliche Metaphern sind. Die Nähe von Computer und Logik wird ja auch weitgehend durchaus als nahe liegend empfunden. Überraschend ist schon eher die Tatsache, dass die Realisierung der Logik auf dem Computer gar nicht so einfach und unproblematisch ist. Eine vollständige Prädikatenlogik zweiter Stufe entzieht sich der Berechenbarkeit. Existierende Implementierungen, wie die Programmiersprache Prolog, beschränken sich auf eine effizient realisierbare Untermenge logischer Formeln, in diesem Fall auf Hornklauseln. Andere Probleme resultieren zum Beispiel daraus, dass Logik aufgrund der inhärenten Monotonie für bestimmte Aufgaben nicht geeignet ist.

Die nachfolgenden Überlegungen beschäftigen sich mit zwei Themen, die auf den ersten Blick wenig miteinander zu tun haben: Zum einen mit computerbasierten Planerstellungsverfahren und einigen Schwierigkeiten, die bei ihrem Einsatz auftauchen, zum anderen mit dialogbasierter konstruktiver Logik – und einigen Problemen, die sich bei ihrer Begründung ergeben. Die These ist, dass beide Problemfelder sich aus der jeweiligen Konzeptualisierung des Zusammenhangs von Logik und Handlung ergeben – und dass ihre Lösung aus der Reflexion dieses Zusammenhangs gewonnen werden kann.

Planungskalküle

Neben Diagnoseexpertensystemen sind Planerstellungsverfahren viel diskutierte Ansätze, die ihren Ursprung im Bereich der Künstlichen Intelligenz haben. Die klassische Beispielsanwendung dieser Systeme sind Blockwelten (z. B. SHRDLU von Winograd), in denen Blöcke auf einer Grundfläche von einer Greifhand manipuliert werden. Diese Welten wirken zwar künstlich, modellieren aber einen Großteil der real auftretenden Probleme automatisierten Planens. In ihnen wird durch eine Reihe von Operationen jeweils eine Situation in eine andere überführt bis schließlich die gewünschte Zielsituation erreicht ist. Im logikbasierten Situationskalkül (McCarthy / Hayes) werden Situationen als Mengen von Merkmalen bestimmt, die in ihnen gelten. So lässt sich beispielsweise sagen

ONTABLE(b,s) [Block b befindet sich auf der Fläche]
ON(a,b,s) [Block a steht auf Block b]
CLEAR(a,s) [Auf Block A steht nichts]
HOLDING(nil,s) [Die Greifhand hält nichts]

Die Aussagen beziehen sich jeweils auf eine Situation. Deshalb erhalten die Prädikate jeweils ein weiteres Argument für die Situation, in der sie gelten, hier S. Die aufgeführten Aussagen in ihrer Gesamtheit, d.h. als Konjunktion verstanden, stehen für eine Situation. Geplant werden soll der Übergang von einer derartigen Startsituation zu einer entsprechenden Zielsituation, in der Regel über mehrere Zwischenstufen. Dieser Übergang erfolgt durch geeignete Operationen. In unserer Beispielswelt können dies sein:

PICKUP(X,S) [Die Greifhand nimmt einen Block X von der Fläche auf und hält ihn]
PUTDOWN(X,S) [Die Greifhand legt einen Block X auf der Fläche ab]
UNSTACK(X,Y,S) [Die Greifhand nimmt einen Block X von einem Block Y ab und hält ihn]
STACK(X,Y,S) [Die Greifhand setzt einen Block X auf einen Block Y]

Die Operationen erfolgen ausgehend von einer Situation S. Die Großschreibung signalisiert den Übergang zu Gegenstands- und Situationsvariablen.
Am Beispiel der Operation UNSTACK wollen wir uns deren Formalisierung genauer betrachten: Für alle Blöcke X und Y und alle Situationen S müssen folgende Vorbedingungen gelten:

HOLDING(nil,S) &
CLEAR(X,S) &
ON(X,Y,S)

Nach ihrer Anwendung gelten folgende Nachbedingungen:
HOLDING(X,UNSTACK(X,Y,S)) &
CLEAR(Y,UNSTACK(X,Y,S)) &
NON HOLDING(nil,UNSTACK(X,Y,S)) &
NON CLEAR(X,UNSTACK(X,Y,S)) &
NON ON(X,Y,UNSTACK(X,Y,S))

UNSTACK(X,Y,S) bezeichnet hier die Situation, die durch die UNSTACK-Operation aus einer Situation S entsteht, & und NON stehen für Konjunktion und Negation. Das Planungsverfahren kann nun so aussehen: Beginnend mit der Startsituation suchen wir Situationen, die die Vorbedingungen geeigneter Operationen erfüllen, um schließlich eine Situation zu erreichen, die die Anforderungen an die Zielsituation erfüllt. Damit ist das Planungsplan auf ein Ableitungsproblem zurückgeführt und es besteht die Hoffnung, es an ein automatisches Beweisverfahren übertragen zu können. Betrachten wir (jedoch) ein Beispiel: Wir gehen von der eingangs angeführten Situation aus (Block a auf Block b) und definieren als Zielsituation eine, in der beide Blöcke auf der Fläche stehen in der also gilt:

ONTABLE(a,T) &
ONTABLE(b,T)

Die Zielsituation ist durch die sukzessive Anwendung von UNSTACK und PUTDOWN auf s0 erreichbar. Ausgehend von der Situation s0 kommen wir damit zur Situation s2 = PUTDOWN(a,UNSTACK(a,b,s0).
Zu den Nachbedingungen von UNSTACK(a,b,s0) gehört HOLDING(a,UNSTACK(a,b,s0)) und dies ist die einzige Vorbedingung der (hier nicht ausformulierten) PUTDOWN-Operation. Diese enthält wiederum als eine Nachbedingung ONTABLE(X,PUTDOWN(X,Y,S)), was sich in unserem Zusammenhang auflöst zu ONTABLE(b,s2). Damit ist die zweite Anforderung unserer Zielsituation erfüllt. Dass die erste, nämlich ONTABLE(a,s2) auch erfüllt ist, ist intuitiv klar, denn der Block a blieb ja die ganze Zeit auf der Grundfläche. Bleibt nur die Frage, wie das formal herleitbar ist. Damit kommen wir zu unserem eigentlichen Thema, dem

Rahmenproblem (frame problem)

Die traurige Wahrheit ist nämlich, dass sich dieser Fakt formal gar nicht herleiten lässt. Wir haben in unseren Übergangsoperationen nur die Veränderungen beschrieben, nicht das, was gleich bleibt. Da die Situationsbeschreibungen immer nur für die jeweilige Situation gelten, lässt sich über die Fortdauer ihrer Geltung nichts folgern. Wir müssen zusätzliche Rahmenaxiome formulieren, die festhalten, was sich durch einen Übergang nicht verändert. Das ist grundsätzlich möglich, aber da diese im Situationskalkül bei jedem Übergang einzeln gehandhabt werden müssen, machen sie das Vorgehen sehr ineffizient und bei hinreichender Komplexität praktisch unmöglich. Die Aussage, dass sich alle nicht genannten Merkmale nicht geändert haben lässt sich erst mit der Prädikatenlogik zweiter Stufe formulieren – und dafür gibt es keine effizienten Implementierungen.
Nun ist das Rahmenproblem nicht nur ein Problem der logikbasierten, sondern aller Planverfahren; und seine effiziente Lösung deshalb immer ein Hauptanliegen. Der Situationskalkül schneidet dabei aber eben nicht gut ab. Dies ist ein Grund, warum sich de Forschung von inferenzbasierten Planerstellungsverfahren weitgehend abgewandt hat.
Es gibt natürlich innerhalb und außerhalb des logikbasierten Vorgehens eine Reihe von Vorschlägen zur Handhabung des Problems. Zwei davon sollen kurz betrachtet werden. Der eine ist Kowalskis Idee der Benutzung von Eigenschaftsvariablen, die durch eine kleine syntaktische Umformung das Problem. Kowalski formt die Merkmalsprädikate zu Termen in einem generellen Prädikat GILT um und gliedert dabei die Situation als eigenes Argument aus.
Aus ON(a,b,s0) wird so GILT(ON(a,b),s0).
Für die Merkmale können jetzt auch innerhalb der ersten Stufe Variablen benutzt werden und es lässt sich für alle M,X,S formulieren
GILT(M,S) & M /= HOLDING(X) -> GILT(M,PUTDOWN(X,S))
Wenn man die ursprüngliche Formalisierung der PUTDOWN-Operation um dieses Rahmenaxiom ergänzt, ergibt sich in der Tat eine elegante theoretische Lösung des Rahmenproblems. Auch in der Effizienz ist diese dem Situationenkalkül deutlich überlegen. Wenn gleichwohl Bedenken

bleiben, dann betreffen diese nach wie vor die Effizienz, einerseits weil die Rahmenaxiome auch hier unangenehm lang werden können, andererseits weil die Gefahr nutzloser Berechnungen steigt. Das hängt aber letztendlich von der jeweiligen Implementierung ab.

Der Hauptgrund dafür, dass sich diese Lösung nicht allgemein durchgesetzt hat, liegt wohl in der Konkurrenz des, zumindest in Hinsicht auf die effiziente Lösung des Rahmenproblems, Besseren, was in diesem Fall das STRIPS-Verfahrens nach Fikes und Nilsson bedeutet, insbesondere in seinen technisch verbesserten Varianten. Damit wurde allerdings nach allgemeiner Überzeugung auch der logikbasierte Ansatz zugunsten eines suchbasierten aufgegeben.
Was heißt das? Die Suche erfolgt in einem Zustandsraum aller durch die formulierten Transformationsoperationen erreichbaren Situationen. Das ist nicht so grundsätzlich anders als im Situationskalkül und auch die Probleme liegen ähnlich. Auch die Suche in einem Graphen ist NP-vollständig, das heißt die Rechenzeiten wachsen exponentiell in Bezug auf die Größe des Zustandsraums. Die Aussicht auf problemlose Algorithmen für große Zustandsräume ist also gering. Ein wesentlicher Unterschied zum Situationskalkül liegt in der Annahme, dass für jede Situation eine vollständige Beschreibung vorliegt. Für den Übergang lassen sich im Wesentlichen die Übergangsoperationen aus dem Situationskalkül verwenden. Dabei werden die Nachbedingungen so modifiziert, dass sie in wegfallende und neu entstehende eingeteilt werden. Beim Übergang zu einer neuen Situation, und das ist nun in der Tat ein wesentlicher Unterschied, werden die alten Merkmale „kopiert“, die wegfallenden gelöscht und die neuen hinzugefügt. Damit ist nun in der Tat das Rahmenproblem sehr elegant gelöst, weil in jeder neuen Situation alle erforderlichen Informationen zur Verfügung stehen.
Es bleibt noch zu klären, wie nun dieser Zustandsraum konkret durchsucht werden soll. Ohne das hier begründen zu können, erweist es sich als zweckmäßig, vom Ziel her zu beginnen, also eine Rückwärtssuche durchzuführen. Man sucht also für alle Zielmerkmale Operationen, die sie erzeugen, und betrachtet die Vorbedingungen dieser Operationen als neue Ziele, für die man entsprechend vorgeht bis man die Startsituation erreicht hat. Solange noch Ziele offen sind, wird das Verfahren fortgesetzt und wenn eine anwendbare Operation nicht zum Erfolg führt, wird eine andere versucht. Dieses Verfahren wird als Zweck-Mittel-Analyse (means ends analysis) oder auch Mittel-Ziel-Analyse (MZA) bezeichnet. Sie ist in STRIPS so realisiert, dass neue Ziele zuerst behandelt werden, die Suche also zuerst in die Tiefe statt in die Breite geht. Das Verfahren lässt sich somit als eine rückwärts gerichtete Tiefensuche mit Backtracking (bzw. mit Zielkeller) beschreiben.

Logik und Handlung

Warum ist dieses Verfahren nicht logikbasiert? Die Frage ist nicht so absurd, wie sie angesichts seiner üblichen Einordnung als Suchverfahren in Abhebung von Inferenzsystemen klingen mag. Auch seine Urheber (Fikes und Nilsson) rechneten es interessanterweise zunächst den

Inferenzsystemen zu und betitelten ihre Arbeit dementsprechend „STRIPS: A New Approach to Theorem Proving in Problem Solving“. Dafür spricht zunächst einmal, dass die Beschreibung der Situationsmerkmale in logischer Formalisierung erfolgt. Sie ist ja gegenüber der im Situationskalkül praktisch unverändert. Das gilt auch auf für die Formalisierung der Übergangsoperationen. wo die Neuigkeit auch nicht in der benutzten Formalisierung liegt.
Die Übereinstimmungen zwischen den Verfahren reichen aber weiter. So gibt es eine deutliche strukturelle Entsprechung zwischen der means ends analysis und dem Resolutionsverfahren nach Robinson. Ein Schritt in der MZA sieht so aus: Um ein Merkmal M zu erreichen, muss man die Vorbedingungen eines Operators erreichen, der M erzeugt, und ihn dann anwenden. Entsprechend führt ein Resolutionsschritt von begründeten Vorbedingungen einer Resolutionsregel zu ihrer Konsequenz. Und auch in der Gesamtfolge der Schritte kann man Ähnlichkeiten entdecken. Denn selbstverständlich muss sich auch ein Theorembeweiser durch den Raum seiner Regeln und Fakten bewegen. Betrachtet man nun den verbreitetsten Beweiser, nämlich die Logikprogrammiersprache PROLOG, so sieht man, dass sie dem gleichen Vorgehen folgt wie STRIPS, nämlich einer rückwärtsgerichteten Tiefensuche mit Backtracking.
Der grundliegende Unterschied der means ends analysis (MZA) zu den traditionellen logikbasierten Planungssystemen liegt meines Erachtens in der andersartigen Handhabung der Situationen und Operationen. Statt einer Beschreibung der Situationsveränderung wird sozusagen die Situationsbeschreibung verändert. Durch die hinzukommenden und wegfallenden Nachbedingungen ändert sich die Situation, bleibt aber dabei als gesamte erhalten. Eben das geschieht in Planungsprozessen: Nicht die Situationen werden verändert, wohl aber probeweise ihre Beschreibungen. Genau dadurch wird auch das Rahmenproblem umgangen. Mit diesem Einbeziehen von Veränderungshandlungen verlässt man vielleicht die klassische Logik, die ja auf Aussagen, d.h. Beschreibungen beschränkt ist, nicht aber notwendig das Paradigma der Logikprogrammierung. Innerhalb dessen ist es zum einen sehr wohl möglich, die MZA zu implementieren. Zum anderen gehört es zum Wesen der Logikprogrammierung wie jeder Programmierung, dass in ihr Zustandsveränderungen vorgenommen werden.

Ein Grund, die MZA gleichwohl nicht zu den deduktiven zu zählen, mag darin liegen, dass man unterscheiden will zwischen der Logik (in reiner Gestalt) und den Fragen ihrer Implemenierung in einzelnen Inferenzsystemen. Das ist dem Grundsatz nach natürlich berechtigt. Zum einen unterscheiden sich, wie schon erwähnt, die Logik und ihre Implementierungen in ihrer Mächtigkeit. Zum anderen gibt es verschiedene Möglichkeiten ihrer Implementierung, die technische Vor- und Nachteile haben mögen, die aber im Prinzip gleichrangig sind. So ist es weder für die Zweck-Mittel-Analyse noch für die Logik zwingend, dass sie als Tiefensuche implementiert werden.
Dies alles zugestanden, bleibt aber klärungsbedürftig, ob alle Implementierungsfragen der Logik nur äußerlich sind oder ob ein Kern davon vielleicht zum Verständnis der Logik dazugehört. Sind der Gegenstand der Logik nur ideale Objekte oder Formalismen oder stellt sie

eine Lehre von Begründungshandlungen dar und lässt sich nur als solche wirklich verstehen? Letztgenannte Position wird von der konstruktiven Logik vertreten. Wie für eine philosophische Frage nicht anders zu erwarten, gibt es umfangreiche Diskussionen über das Für und Wider. Ich kann nicht hoffen, dieser Diskussion hier Wesentliches hinzuzufügen und belasse es deshalb bei der Bekundung, dass der konstruktiven Position in diesem Punkt meine Sympathie gehört.
Zwischen der konstruktiven Logik und informatischen Logikimplementierungen gibt es eine Reihe von Berührungspunkten. So folgt in PROLOG ebenso wie in der konstruktiven Logik, anders als in der klassischen, aus der doppelten Verneinung nicht die Affirmation. Dem entsprechen gleichartige Konzepte. Die „negation as failure“ in PROLOG ergibt sich, vergleichbar der konstruktiven aus dem gescheiterten Versuch, das Verneinte zu beweisen. Dieser konstruktive Grundgestus lässt sich durchgehend feststlellen. Im Grund ist es natürlich auch nicht überraschend, dass eine reale Implementierung konstruktive Verfahren verwendet. Logikprogrammierung lässt sich mithin zwanglos als die Organisation von Begründungshandlungen verstehen und entspricht damit weitgehend dem Konzept konstruktiver Logik, allerdings bleibt diese Übereinstimmung nicht ohne Rest.

Dialogische Begründung konstruktiver Logik

Vor allem durch die Erlanger Schule hat sich die dialogische Begründung konstruktiver Logik als Paradigma durchgesetzt. Ihr Modell ist ein Dialog bzw. eine Auseinandersetzung zwischen einem Proponenten, der eine logisch zusammengesetzte Aussage verteidigt und einem Opponenten, der diese bestreitet. Auf einer ersten Stufe geht es um materiale (inhaltliche) Dialoge, erst auf einer zweiten Ebene mit entsprechend erweiterten bzw. modifizierten Regeln geht es um formale Gewinnbarkeit und logische Wahrheit. Für die verwendeten Junktoren gibt es Partikelregeln, die z.B für das einschließende Oder (VEL) besagen, dass der Opponent die Gesamtbehauptung A VEL B angreifen kann und der Proponent dann beliebig A oder B begründen kann. Im Fall der Subjunktion muss mit dem Angriff auf A SUB B der Angreifer die Begründung von A übernehmen muss und der Verteidiger ist dann zur Begründung von B verpflichtet. In ähnlicher Weise wird NON A durch die Begründung von A angegriffen. Gelingt diese, so gibt es dagegen keine Verteidigung. Wie jedes Spiel braucht auch die dialogische Logik einige Rahmenregeln. Dazu gehört z.B. dass Proponent und Opponent abwechselnd am Zuge sind, Argumente vorzubringen, die jeweils Angriffe auf gegnerische Behauptungen oder Verteidigungen sein können. Der Proponent hat gewonnen, wenn der Opponent keine Argumente vorbringen kann, der Opponent, wenn dem Proponenten dies nicht gelingt.

Das Schöne an der dialogischen Logik ist, dass sie Logik als System von Begründungshandlungen konzipiert. Schönheitsfehler sind die folgenden: In der konstruktiven Logik gilt nicht das tertium non datur nicht allgemein, sondern nur in eingeschränkten Bereichen. Desgleichen folgt auch aus der doppelten Negation einer Aussage ihre Affirmation nicht zwangsläufig. Veranschaulicht man dies im Dialog, so wird die Behauptung NON NON A

SUB A vom Opponenten durch die Postulierung von NON NON A angegriffen, dieses wiederum vom Proponenten mit NON A und dieses vom Opponenten mit A. Das scheint nun schon den Sieg des Proponenten im formalen Dialog zu bedeuten, denn er muss ja nur die Begründung des Opponenten für A nachmachen, um seiner eigenen Begründungsverpflichtung für A nachzukommen, zu der er sich in der Ausgangsbehauptung für den Fall verpflichtet hat, dass der Vordersatz der Subjunktion durch den Gegner begründet werden kann. Es scheint aber nur so, weil eine Rahmenregel besagt, dass die Dialogpartner sich gegen den letzten Angriff immer zuerst verteidigen müssen. In dieser Situation muss das den Anhängern der klassischen Logik (mit tertium non datur) als Taschenspielertrick erscheinen. Das ist nun nicht wahr. Es gibt gute Gründe dafür, die Rahmenregeln genau so abzufassen, aber es zeigt doch eine Grundschwäche des Dialogmodells, dass nämlich Argumente von unterschiedlichem Status in gleicher Position erscheinen. Die Rahmenregel mit dem letzten Angriff ist im Grunde nur eine Kompensation für diese Schwäche.
Der zweite Schönheitsfehler ist vielleicht Geschmackssache. Die Dialoge verbergen kaum oder nicht, dass es um Zweipersonennullsummenspiele geht, also um Spiele wie Schach, das wiederum kaum oder nicht verbirgt, dass es einen Krieg modelliert. Ob dies das Grundmuster für menschliche Argumentation sein muss, sei im Momentdahingestellt.
Schließlich, und das ist in unserem Zusammenhang das wichtigste Argument: das dialogische Modell bietet schlechte Anschlussmöglichkeiten an Felder wie Logikschaltungen, Logikprogrammierung oder das Feld einer erweiterten Logik der Handlungen. Entweder gibt es in diesen gar keine Entsprechung zum dialogischen Modell oder dieses ist, wie im Fall de Handlungen, nur auf eine Teilmenge anwendbar.

Diese Nachteile lassen sich vermeiden, wenn man eine konstruktive Logik in ganz ähnlicher Weise als ein System der Regelung von Begründungsverpflichtungen aufbaut, sie aber nicht an den Zweipersonendialog bindet, sondern als kollaborative Aufgabenlösung organisiert. Die Partikelregeln blieben im Prinzip die gleichen. Sie stellten eine Verteilung von Begründungsverpflichtungen dar. Im Falle der Negation und der Subjunktion würden Begründungspflichten delegiert und einer anderen vorgelagerten Ebene zugewiesen. Die Bedingung einer Bedingung, ebenso wie die Negation einer Negation erscheinen aber natürlich nicht auf der Ausgangsebene, sondern auf einer dritten, so dass ihr Status von vornherein nicht verunklart wird. Die Rangfolge der Verpflichtungen ist wiederum durch Rahmenregeln festzulegen. Der Grundgedanke dieses Ansatzes hat Ähnlichkeit mit Kolmogorovs Aufgabenlogik. Die grafische Darstellung verliert gegenüber der dialogischen Logik an Geschlossenheit, da sich nicht mehr alles schön auf zwei Spalten aufteilen lässt. Stattdessen benötigt man sehr bald Baumstrukturen. Damit befindet man sich aber in guter Gesellschaft, nämlich der von Logikprogrammierung, Suchbäumen und means-end-analysis. Es ist unübersehbar, dass der Aufgabenansatz zu allen diesen Gebieten größere Nähe hinsichtlich der Art der Konzeptualisierung aufweist.
Durchaus in engem Zusammenhang damit eröffnet sich die Chance auf eine weitere Erweiterung, nämlich auf die einer Logik der Handlungen. Gemeint

ist damit nicht eine Handlungslogik im Sinne einer Logik der Aufforderungs-, Gebots-, Verbotssätze, sondern die Erweiterung einer Logik komplexer Begründungshandlungen auf komplexe Handlungen allgemein.

Ausleitung

Von der aufgabenorienten Begründung konstruktiver Logik lässt sich sogar eine Brücke schlagen zu dem Beispiel, von dem unser Überlegungen den Ausgang nahmen: den logischen Schaltwerke. Analog dazu, wie Handlungen durch entsprechende Junktoren zu komplexen Handlungen verknüpft werden können, kann man sich auch die Verknüpfung der mikrohandlungsförmigen Stromflüsse zu elektronischen Schaltwerke interpretieren, während von einem dialogbasierten Logikmodell keine Brücke zu ihnen ersichtlich ist. Die Rede von Handlungsförmigkeit in diesem Zusammenhang mag Stirnrunzeln auslösen. Es gibt jedoch zunächst eine relative Berechtigung dafür, in der Welt der Programmierung in einer Handlungsmetaphorik von Befehlen und ihrer Ausführung zu sprechen. Berechnungen oder logische Folgerungen können ja auch von Menschen ausgeführt werden und dann beschreiben die Programme in der Tat einen Handlungszusammenhang. Dies für den Computer beizubehalten, bedeutet nicht, ihm irgendwelche menschlichen Eigenschaften zuzuschreiben. Ihm wird damit allerdings eine gewisse Handlungsförmigkeit zugeschrieben, das heißt die strukturelle Eigenheit, Handlungen und Objekte in ihn abbilden und manipulieren und die Ergebnisse entsprechend interpretieren zu können. Dazu müssen Computer entsprechend entworfen und gebaut sein und insbesondere muss es genug Möglichkeiten (Programme) geben, Objekte und Handlungen der Menschenwelt auf die Hardware des Computers zu projizieren und die Ergebnisse zurück zu interpretieren. Objekte werden dabei in Bits umgesetzt und die Handlungen in die durch Logik gesteuerten Stromflüsse. Eben diese Steuerbarkeit, dass sie nämlich nach Bedingungen ein- und ausgeschaltet werden können, gibt ihnen – keinen Handlungscharakter, aber Handlungsförmigkeit.