Linguistische Arbeiten 406

Herausgegeben von Hans Altmann, Peter Blumenthal, Herbert E. Brekle, Gerhard Helbig, Hans Jürgen Heringer, Heinz Vater und Richard Wiese

Detlef Peter Zaun

Künstliche neuronale Netze und Computerlinguistik

Max Niemeyer Verlag
Tübingen 1999

Die Deutsche Bibliothek – CIP-Einheitsaufnahme

Zaun, Detlef P.: Künstliche neuronale Netze und Computerlinguistik / Detlef Peter Zaun. – Tübingen : Niemeyer, 1999
(Linguistische Arbeiten ; 406)
Zugl.: Köln, Univ., Diss., 1996

ISBN 3-484-30406-5 ISSN 0344-6727

Gedruckt auf alterungsbeständigem Papier.
Druck: Weihert-Druck GmbH, Darmstadt
Buchbinder: Nädele Verlags- und Industriebuchbinderei, Nehren

Inhaltsverzeichnis

Inhaltsverzeichnis `neuron-s_code.ps`

Vorwort

Bezogen auf die – zumindest in der Vergangenheit – verschiedenartigen Verarbeitungsmethoden der symbolorientierten Computerlinguistik und den Modellen künstlicher neuronaler Netze deutet der Titel dieses Buches an, daß es möglicherweise Berührungspunkte zwischen beiden Disziplinen geben kann. Solche Berührungspunkte, die oft auch „Reibungspunkte" sind, möchte die vorliegenden Arbeit in das Zentrum der Diskussion stellen, nicht aber ohne mit dem NEURON–S Simulator und dem Verfahren der selektiven Propagierung einen eigenen Beitrag zur Computerlinguistik und zu künstlichen neuronalen Netzen zu leisten.

Die Entscheidung, den NEURON–S Programmcode separat in maschineller Form über das Internet frei zugänglich zu machen, erfolgte in erster Linie aus folgenden Gründen: der Programmcode kann mit Hilfe von Zusatzprogrammen direkt in verarbeitbaren Code übersetzt werden und ist nur für diejenigen interessant, die an einer praktischen Arbeit mit künstlichen neuronalen Netzen sowie deren Implementation interessiert sind. Zudem hätte der Programmcode den Umfang des vorliegenden Buches deutlich vergrößert.

An dieser Stelle möchte ich mich bei Prof. Dr. Heinz Vater bedanken, der die Erstellung der vorliegenden und im Sommer 1996 als Dissertation von der Philosophischen Fakultät der Universität zu Köln angenommenen Arbeit angeregt und ihre Entstehung mit Toleranz, Hilfe und Kritik begleitet hat sowie erster Referent war. Tag des Rigorosums war der 6.7.1996. Bedanken möchte ich mich auch bei Prof. Dr. Jürgen Rolshoven, der sich dazu bereit erklärt hat, als zweiter Referent eine Begutachtung vorzunehmen.

Mein größter Dank gebührt meiner Frau, Dr. Anja Kristina Radojewski-Zaun, die mich bei der Abfassung dieser Arbeit stets unterstützt hat.

Zons, den 27.07.1998 Detlef Peter Zaun

1 Einleitung

1.1 Unzufriedenheit

Betrachtet man die Erfolge der künstlichen Intelligenz (KI) und der Computerlinguistik gemessen an der Leistungsfähigkeit heutiger Rechenanlagen, so scheint es, daß die Forschung im Hinblick auf die traditionelle, symbolische Verarbeitung in einer Sackgasse angelangt ist. Im Bereich syntaktischer Analyseprogramme ist beispielsweise der von Earley (1970) entwickelte Chart–Parser immer noch der meist verwendete Algorithmus. Neuere Entwicklungen auf dem Gebiet der automatischen Analyse natürlicher Sprachen sind häufig nur als Modifikationen alter Modelle anzusehen[1]. In den auf symbolischer Verarbeitung basierenden semantischen Modellen ist die Situation ähnlich. Die in der Computerlinguistik dominierenden Ansätze sind semantische Netze, die eine Codierung mit indizierten Kanten und symbolischen Knoten verwenden. Eine Darstellung solcher, in Anlehnung an häufig gebrauchte Kantenindizierungen benannten „is–a" Netze, findet sich beispielsweise bei Norman/Rumelhart (1975), Findler (1979) und Ritchie/Hanna (1983). Obwohl semantische Netze oft intuitiv erfaßbar sind und systematische Zusammenhänge auf den ersten Blick klar wiederzugeben scheinen, mangelt es den meisten Systemen an konzeptioneller Klarheit. Ritchie/Thompson (1984:375) stellen hierzu fest:

> To propose a theory of meaning based on semantic nets, one should specify (among other things):
>
> - What organisational links are available
> - How the various types of links and nodes can be connected
> - What operations can performed on these combinations of links and nodes
>
> In the past, most proposals for semantic net systems have not made such details clear (relying on a few sample diagrams and the reader's intuitions), and many confusions have crept in concerning the use of network notation.

Diese Kritik trifft ebenfalls auf Systeme zu, die eine Prädikatenlogik verwenden, da sich nach Simmons/Bruce (1971) alle linguistischen Codierungen in Netzwerkform direkt in Formate eines Prädikatenkalküls überführen lassen (vgl. auch Wilks 1977:202).

Sieht man von dem bei Lenat (1990) beschriebenen Cyc Projekt ab, das sich die ehrgeizige Aufgabe gestellt hat, den größten Teil menschlichen Alltagswissens in Form von Sätzen der Prädikatenlogik und Inferenzmechanismen zu codieren, ist in der Forschung zur KI keinem dieser Modelle ein wirklicher Durchbruch gelungen. Der KI–Forscher H. Dreyfus bemerkte in einem Interview mit dem NDR–Fernsehen:

> Aber wenn Cyc scheitert, ist die KI wirklich am Ende. Das ist wirklich das letzte Projekt, das noch irgendeine Hoffnung bietet. Die KI ist dann so viele Jahrhunderte tot, wie es dauert, dieses mysteriöse Ding namens Gehirn zu verstehen.
>
> (zitiert nach Schult 1994:99)

[1] Eine Übersicht gängiger symbolverarbeitender Parser bietet Hellwig (1989).

Obwohl sich die Computerlinguistik von einer reinen Hilfswissenschaft zu einem integralen und theoriebildenden Bestandteil der KI entwickelt hat[2], sind sprachverstehende und intelligente Computersysteme noch nicht in Sicht. Vor dem Hintergrund symbolischer Ansätze scheint in der KI und Computerlinguistik eine Situation eingetreten zu sein, die an die Situation in der Forschung zur maschinellen Übersetzung nach Erscheinen des berüchtigten ALPAC-Reports im Jahre 1966 erinnert[3]. Viele Forschungsvorhaben wurden daraufhin eingestellt, da auf absehbare Zeit keine erfolgversprechenden Perspektiven in Sicht waren, was dazu führte, daß die Entwicklung in den folgenden Jahren stagnierte.

Als ein Beispiel für die Unzufriedenheit mit den Ergebnissen symbolischer Verarbeitung in der KI und der Computerlinguistik soll hier der Bericht von Dyer (1991:383) stehen:

> Although my students and I worked diligently at designing and building such complex symbol processing systems, by 1986 I was becoming increasingly unhappy with the tremendous amount of knowledge engineering that such systems required, and unsatisfied with their resulting fragility. I began to question whether symbol processing (based on hand-coded knowledge constructs and processing strategies, such as logical predicates, frames, scripts, demons, unification, discrimination trees, and so on) was the only available research path to modeling the mind. While symbol processing technology allows one to build sophisticated architectures, capable of exhibiting complex behavior in limited task/domains, it seems very difficult to theorize as to how such complex structures and processes could have arisen in the first place in such systems, through learning and experience. In addition, hand-coded structures seemed to lack an inherent 'richness' in representation. At the time I suspected that the inability to automatically acquire and represent the complexity of experience is a major reason for the resulting fragility of these human-engineered systems.

In diesem Zusammenhang stellt sich die Frage, auf welcher Ebene der Verarbeitung nach Gründen für die Unzulänglichkeit traditioneller Systeme zu suchen ist. Waltz/Pollack (1985:69f.) sehen den Grund in den Beschränkungen, die der KI und der Computerlinguistik durch das Modell der seriellen und symbolmanipulierenden Von-Neumann-Architektur[4] auferlegt werden:

> Computer scientists, like cognitive scientists, tend to be limited by the conceptual framework of serial processing, the 30-year-old framework of the "von Neumann" machine,

[2] Noch im Jahre 1977 konstatierte Eisenberg (1977:4): „Auch wenn es eine ganze Reihe von Berührungspunkten gibt (...), ist die Beschränkung der Linguistik zunächst auf Teilbereiche des 'Wissens über die Sprache' und dann auf dieses Wissen insgesamt aber dennoch eines der schwerwiegendsten Hindernisse zur Kooperation mit der künstlichen Intelligenz (...)." Eine solche Beschränkung auf 'Wissen über Sprache' läßt sich heute im allgemeinen nicht mehr beobachten, da auch in die Linguistik Begriffe wie Kontext und Weltwissen in die Theoriebildung eingegangen sind. Schank (1990:13) fordert daher: „AI (Artificial Intelligence) should, in principle, be a contribution to a great many fields of study. AI has already made contributions to psychology, linguistics, and philosophy as well as other fields. In reality, AI is, potentially, the algorithmic study of processes in every field of inquiry."

[3] Der ALPAC-Bericht (vgl. ALPAC 1966) stellte die Unzulänglichkeit maschineller Sprachverarbeitung am Beispiel maschineller Übersetzung fest und konstatierte einen mangelnden Bedarf an solchen Programmen.

[4] Vgl. S.10 dieser Arbeit.

> with its Central Processing Unit connected to its passive array of memory by a small bundle of wires.

Diese Hypothese scheint die Ursache auf die Ebene der physikalischen Realisierung von Computerprogrammen zu reduzieren und Programmiersprachen, Algorithmen, Theorien über kognitive Strukturen und Grammatikmodelle der Computerlinguistik auszuklammern. Da die einzelnen Ebenen jedoch stark miteinander interagieren, ist eine differenziertere Betrachtungsweise geboten.

Aufgabe von Kapitel 2 ist daher eine eingehendere Untersuchung des klassischen, symbolischen Ansatzes in der KI und Computerlinguistik, sowie die diesem Ansatz zugrundeliegende Theorie und Systemarchitektur. Im Vordergrund der Betrachtung soll dabei die Chomsky–Hierarchie der Sprachen stehen, ein formaler Rahmen, der für die Entwicklung computerlinguistischer Modelle von großer Bedeutung ist. Bezüglich der mit den Sprachen der Chomsky–Hierarchie korrespondierenden Grammatikmodelle soll besonders auf die generative Grammatik in der Tradition von Chomsky (1957) näher eingegangen werden, die die Basis für viele formale und natürlichsprachliche Grammatiken gelegt hat. Die Government Binding Theorie (vgl. Chomsky 1986) läßt sich in ihren Grundzügen direkt aus den frühen Phrasenstrukturgrammatiken Chomskys ableiten. Selbst Ansätze wie die Lexikalisch–Funktionale Grammatik (vgl. Kaplan/Bresnan 1982) oder die Head–driven Phrase Structure Grammar (vgl. Pollard 1984) tragen für die Verarbeitung entscheidende Komponenten des Modells von 1957 in sich. Da sich eine seriöse Beschäftigung mit alternativen Verarbeitungsmodellen stets in argumentativer Auseinandersetzung mit den klassischen Ansätzen befinden sollte, muß der kritischen Erörterung ebendieser traditionellen Modelle ein entsprechender Umfang eingeräumt werden. Insbesondere gilt dies, wenn es sich beim Gegenstand der Kontroverse nicht um kleinere notationelle Veränderungen oder die Formulierung neuer Regelmodule handelt, sondern grundsätzliche Aspekte der Verarbeitung und Repräsentation natürlicher Sprachen angesprochen werden.

1.2 Neue Perspektiven

Zitiert man die eben angeführten Unmutsäußerungen von Dyer (1991:383) weiter, so scheint es einen Ausweg aus dem Dilemma zu geben. Er bemerkt:

> Around that time, the two volume PDP books came out (Rumelhart and McClelland, 1986). Like many, I had preordered these books and upon their arrival "devoured" them with great excitement, seeing in them the potential solution to all my problems. Here was an alternative paradigm and technology, holding the promise of automatic acquisition of knowledge, where knowledge consists of distributed representations, and where processing involves spreading activation (versus the unification-based symbol processing mechanisms I had been using).

Die beiden Aufsatzsammlungen von Rumelhart/McClelland (1986), *Parallel Distributed Processing* (= PDP) – *Explorations in the Microstructure of Cognition*, begründeten ein neues Interesse und einen ungeahnten Boom in der Forschung zu künstlichen

neuronalen Netzwerken. Im allgemeinen werden die beiden Bände daher heute als die „Bibel des neueren Konnektionismus" bezeichnet. Die Architektur und Arbeitsweise solcher künstlichen neuronalen Netze kann als Abstraktion über biologische neuronale Netze aufgefaßt werden. Der neuere Konnektionismus beschäftigt sich u.a. mit der Anwendung solcher künstlichen neuronalen Netze bei der Simulation kognitiver Fähigkeiten und scheint somit ein neues Paradigma für die KI und Computerlinguistik zu bilden.

Kapitel 3 behandelt zunächst die Grundprinzipien der Informationsverarbeitung in biologischen neuronalen Netzen, von denen die Arbeitsweise künstlicher neuronaler Netzwerke abgeleitet ist. Im Vordergrund soll zunächst ein Systemvergleich zwischen den Eigenschaften biologischer neuronaler Netze und der klassischen Von-Neumann-Architektur stehen, wobei ein zentraler Aspekt die Fähigkeit zur Sprachverarbeitung ist. Der Aufbau des menschlichen Gehirns, die Frage nach der Lokalisierung von Sprachzentren und die damit verbundene These der Autonomie der Syntax sollen ebenfalls diskutiert werden.

Der Mechanismus der Erregung von Nervenzellen und die Weiterleitung von Aktionsimpulsen an nachgeschaltete Zellen ist ebenso Gegenstand der Untersuchung wie der Prozeß der Signalweiterleitung und der Gewichtung von Aktionsimpulsen innerhalb der Synapsen, den Schaltstellen zwischen den Nervenzellen. Da das „Wissen" eines biologischen neuronalen Netzes neben der Netzwerktopologie im wesentlichen durch diese synaptischen Verbindungen determiniert wird, sind die in der Synapse ablaufenden Prozesse von größtem Interesse für die Konstruktion von daraus abgeleiteten künstlichen neuronalen Netzwerken. Aus diesem Grund soll die Arbeitsweise der Synapsen eingehender betrachtet werden.

Kapitel 4 ist den Prinzipien der Informationsverarbeitung in künstlichen neuronalen Netzwerken – d.h. dem neueren Konnektionismus – gewidmet. Neben einer Darstellung der Arbeitsweise und einer Klassifizierung künstlicher neuronaler Netze soll außerdem die Klärung des Begriffs „neuerer Konnektionismus" im Vordergrund stehen, der impliziert, daß es einen „älteren Konnektionismus" gegeben hat. Der Grundmechnismus der Aktivierungsausbreitung in künstlichen neuronalen Netzen sowie die laterale Hemmung einzelner Verarbeitungselemente bestimmt im wesentlichen das Reaktionsverhalten solcher Netze auf eingegebene Daten. Wichtige Parameter hierbei sind, neben der Netzwerktopologie, die Gewichtungen der einzelnen Verbindungen der Verarbeitungselemente untereinander. Da neben einer manuellen Justierung solcher Gewichte auch automatisierte Adaptionsverfahren existieren, sollen die Vor- und Nachteile solcher Verfahren beleuchtet werden.

Ein weiterer Punkt betrifft die unterschiedlichen Netzwerktopologien und Klassen künstlicher neuronaler Netzwerke. Einige der bekanntesten Modelle sollen deshalb hier vorgestellt werden. Darüberhinaus stellt dieses Kapitel mögliche Kombinationen von Netzwerken mit traditionellen regelbasierten Systemen in hybriden Modellen zur Diskussion.

Grundlegende Aspekte bei der Modellierung von Sprachverarbeitsprozessen werden in Kapitel 5 angesprochen. In diesem Zusammenhang sollen einige Modelle, die bei der Entwicklung sprachverarbeitender konnektionistischer Netzwerke eine wichtige Rolle

spielen, näher betrachtet werden. Dabei ist in erster Linie das „Past–Tense" Modell von Rumelhart/McClelland (1986b) zu nennen, das eine Simulation des Spracherwerbsprozesses bei Kindern in bezug auf den Erwerb des Past Tense im Englischen vornimmt.

Die Zuweisung von Rollen an Konstituenten eines Eingabesatzes steht im Mittelpunkt des von McClelland/Kawamoto (1986) entwickelten „Sentence Processing" Modells. Über eine Spezifizierung der Argumentstrukturen von Verben sowie eine Verknüpfung der Argumente mit semantischen Microfeatures ist das Netzwerk in der Lage, Rollen korrekt zuweisen zu können.

Ein weiterer interessanter Ansatz ist das „Massively Parallel Parsing" nach Waltz/Pollack (1985). Ziel dieses Modells ist es, unter Zuhilfenahme von Kontextinformationen, welche durch Mengen vom Microfeatures ausgedrückt werden, eine Desambiguierung mehrdeutiger Elemente der Eingabekette vorzunehmen und somit eine eindeutige Interpretation eines Satzes zu liefern. Im Gegensatz zu den Modellen von Rumelhart/McClelland (1986b) und McClelland/Kawamoto (1986) handelt es sich hierbei um ein lose gekoppeltes und rekurrentes Netzwerk mit streng lokalen Repräsentationen.

Neuere Ansätze, wie die sequentiellen Netzwerke nach Elman (1990) oder rekursive auto–assoziative Speicher nach Pollack (1988), werden ebenfalls in Kapitel 5 diskutiert, da sie möglicherweise neue Aspekte der Verarbeitung natürlicher Sprachen durch künstliche neuronale Netzwerke bieten.

In Kapitel 6 erfolgt die Darstellung des im Rahmen dieser Arbeit entwickelten NEURON-S Simulators. Dabei handelt es sich um eine Entwicklungs- und Simulationsumgebung für künstliche neuronale Netzwerke, die über entsprechende Zusatzdateien weitgehend frei zu konfigurieren ist. Über externe Dateien lassen sich in der Programmiersprache Cambridge Lisp Aktivierungs-, Propagierungs- und Lernfunktionen definieren und in den laufenden Simulator compilieren. Die Konstruktion von Netzwerken sowie eine Steuerung und Auswertung der Simulation kann dann durch NEURON-S erfolgen.

In diesem Zusammenhang wurde außerdem ein neues Propagierungsverfahren für künstliche neuronale Netze entwickelt, das im folgenden als *„selektive Propagierung"* bezeichnet wird. Durch dieses neue Verfahren ist es möglich, den Verarbeitungsaufwand für die Simulation künstlicher neuronaler Netze erheblich zu reduzieren. Eine Vorstellung und Diskussion der selektiven Propagierung ist ebenfalls Gegenstand dieses Kapitels.

Außerdem sollen anhand unterschiedlicher Fallbeispiele die Möglichkeiten der Codierung sprachlichen Wissens demonstriert werden. So behandelt das erste Beispiel die Deklination von Personalpronomina im Deutschen, das zweite die Codierung von Wortformen und das dritte die gegenseitige Beeinflussung solcher Wortformen untereinander. Dabei soll dargestellt werden, wie eine zeitlich versetzte Darbietung der Elemente einer Eingabekette die jeweiligen Lesarten dieser Elemente determiniert und sich durch diese Vorauswahl die Interpretation in eine bestimmte Richtung lenken läßt.

Für die Darstellung des Programmcodes des NEURON-S Simulators wurde in der ersten Form der vorliegenden Arbeit ein separater zweiter Band verwendet. Diese

Form der Präsentation sollte im wesentlichen die Handhabung der einzelnen Teile dieser Arbeit erleichtern und ließ sich auch bezüglich der unterschiedlichen Inhalte der beiden Bände begründen. Da Programmcode aber besser direkt in maschinenlesbarer Form präsentiert werden sollte, wurde der gesamte Inhalt des vormalig sog. „zweiten Bandes“ in Form einer Postscript–Datei unter der folgenden Adresse im Internet abgelegt:

`ftp://ftp.uni-koeln.de/neural-nets/papers/neuron-s_code.ps.gz`

Die Datei wurde hiefür mit dem Programm `gzip` komprimiert, das Postscript–Format wurde mit `dvips` aus einer TEX–DVI Quelle erzeugt.

2 Computer und Sprache

2.1 Von Automaten und Computern

2.1.1 Die Turingmaschine

Der Aufbau und die Funktionsweise heutiger digitaler Rechenanlagen, allgemein als „Computer" bezeichnet, basiert im wesentlichen auf den theoretischen Überlegungen Alan Turings und den Entwürfen John von Neumanns. Sieht man von den zahlreichen Ideen und Realisierungen mechanischer Rechenautomaten ab, die es seit Leonardo da Vinci und Blaise Pascal immer wieder gegeben hat, so kann der „Urahn" heutiger Computer in dem abstrakten Formalismus der Turingmaschine gesehen werden[5]. Turing (1936) entwarf hierbei einen theoretischen Automaten, dessen schematischen Aufbau Abbildung 2.1 zeigt. Eine Turingmaschine setzt sich demnach aus einem Schreib-

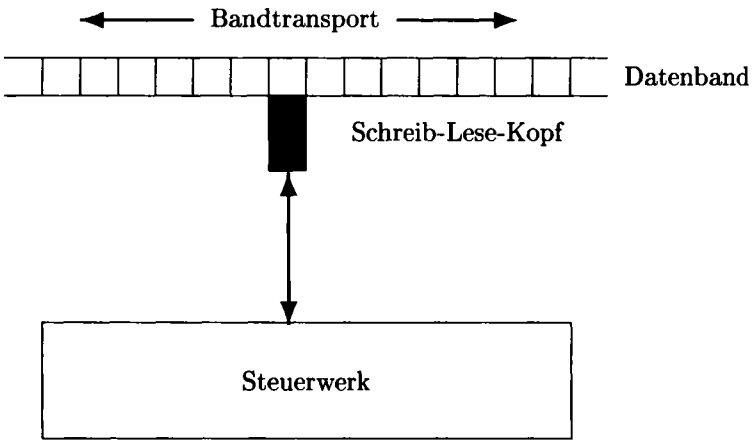

Abb. 2.1: Aufbau einer Turingmaschine

Lese-Kopf, einem Datenband mit einer Menge von Feldern und einem Steuerwerk für die Bandbewegung und die Applikation von Regeln zusammen. Formal kann sie als ein Quadrupel $\{K, \sum, s, \delta\}$ angesehen werden (vgl. Partee/ter Meulen/Wall 1990:510), wobei

K eine endliche Menge von Zuständen ist, die als Untermenge die Menge der Haltezustände F enthält ($F \subseteq K$),

[5] Zu einer exakten Darstellung und umfassenden Erörterung von Turingmaschinen vgl. Minsky 1967:117ff., Hopcroft/Ullman 1969:80ff. und Partee/ter Meulen/Wall 1990:507ff.

$\sum$ eine endliche Menge von Symbolen (Alphabet) und ein Leersymbol #[6],

s ein Startsymbol aus K ($s \in K$) und

δ eine Übergangsfunktion, die die Bandbewegung regelt und für die $K \times \sum \rightarrow K \times (\sum \cup \{L, R\})$ gilt.

Dieser Automat ist in der Lage, aus einer endlichen Anzahl von Symbolen und Steueranweisungen eine nur durch die Bandlänge beschränkte, aber theoretisch unendliche Sequenz von Symbolen zu generieren oder zu erkennen. Dazu verfährt der Automat wie folgt: Beginnend mit einem aktuellen Zustand s (bzw. q_0) des Steuerwerks wird ein neues Symbol auf das Band geschrieben. Dann wird das Band nach rechts oder links bewegt und ein neues Symbol vom Band gelesen. Aus diesem Zeichen und dem aktuellen Zustand des Steuerwerks wird mittels einer Übergangsfunktion δ ein neues Symbol generiert und an die Stelle des zuletzt gelesenen Zeichens auf das Band geschrieben. Anschließend nimmt das Steuerwerk einen neuen Zustand ein und bewegt das Band nach rechts oder nach links. Dieser Vorgang wird solange fortgesetzt, bis das Steuerwerk einen Haltezustand F erreicht.

Problematisch ist in diesem Zusammenhang, daß es algorithmisch nicht entscheidbar ist, ob eine Turingmaschine jemals terminiert, d.h. daß sie einen Zustand F einnehmen kann und somit eine Eingabekette x akzeptiert[7]. Ausgehend von einem Alphabet $\sum$ und den Regeln der Übergangsfunktion δ kann eine Turingmaschine T (falls sie terminiert) eine Eingabekette x zwar syntaktisch bewerten, hingegen können keinerlei Aussagen über deren Semantik gemacht werden. Daher ist algorithmisch weder entscheidbar, ob ein spezieller Wert Resultat eines Programms[8] sein kann, noch ob syntaktisch verschiedene Programme den gleichen Wert berechnen.

Charakteristisch für eine Turingmaschine ist also, daß es sich bei ihr um einen sequentiell arbeitenden Automaten handelt, der mit einer endlichen Anzahl von Regeln, Symbolen und Zuständen in der Lage ist, theoretisch unendliche Ketten zu generieren bzw. zu erkennen. Hinsichtlich rekursiver Fähigkeiten gilt, daß wenn x von einer Turingmaschine akzeptiert oder erzeugt wird, x eine rekursiv aufzählbare Menge ist und es einen Algorithmus zur Erkennung bzw. Generierung aller Elemente von x geben muß (vgl. Wall 1972:280). Hinsichtlich des Verhältnisses zwischen Turingmaschinen und Sprachen der Chomsky-Hierarchie (vgl. Partee/ter Meulen/Wall 1990:561ff.) kann im Vorgriff auf die zum Thema Sprache folgenden Ausführungen

---

[6] Wall (1972:276) charakterisiert eine Turingmaschine als $\{K, \sum, \#, \delta, q_0, F\}$ mit $q_0 = s$ und der Einschränkung, daß # nicht Element aus $\sum$ ist. Der Hintergrund hierfür ist, daß das Leersymbol zur Menge der erlaubten Bandsymbole gehört, nicht aber zur Menge der Eingabesymbole. Partee/ter Meulen/Wall (1990:511) sprechen in diesem Zusammenhang von: „$\{K, \sum, s, \delta\}$ and $\sum_1$, a subset of $\sum$ which does not contain #...". Hopcroft/Ullman (1969:81) dagegen verwenden $\{K, \Gamma, \sum, \delta, q_0, F\}$, wobei Γ eine endliche Menge der erlaubten Bandsymbole mit dem Leersymbol # ist, $\sum$ hingegen eine Untermenge von Γ ohne #.

[7] Zum sogenannten „Halteproblem" vgl. Partee/ter Meulen/Wall 1990:522ff., Minsky 1967:146ff., Hopcroft/Ullman 1969:80ff. und Wall 1972:287.

[8] Mit der Bezeichnung Programm sind hier hauptsächlich die möglichen Zustände K, das Alphabet $\sum$ und die Übergangsregeln δ des Steuerwerks gemeint.

festgehalten werden, daß wenn eine Sprache L von einer Typ–0–Grammatik erzeugt wird, L auch von einer Turingmaschine erkannt wird, bzw. wenn L durch eine Turingmaschine erkannt wurde, muß L von einer Typ–0–Grammatik generiert worden sein (vgl. Hopcroft/Ullman 1969:111f.).

2.1.2 Computer

Heutige Computer entsprechen in ihrer Konzeption im wesentlichen dem Entwurf der Turingmaschine. Die wichtigsten Bestandteile eines Computers sind hierbei Speicher (Memory) und Hauptprozessor (Central Processing Unit = CPU)[9]. Wie in Abbildung 2.2 zu sehen ist, bestehen zwischen dem Prozessor und dem Speicher Verbindungen,

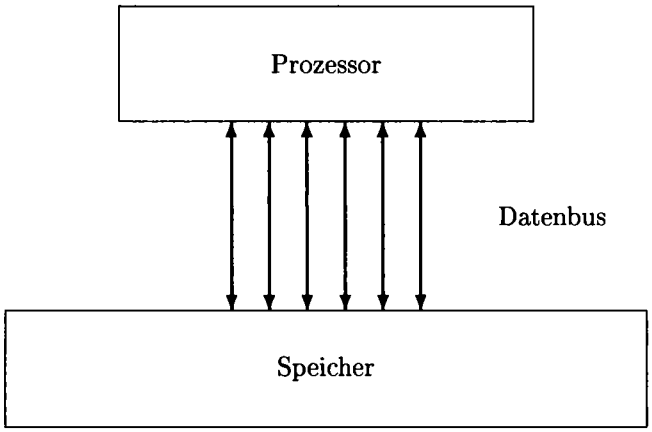

Abb. 2.2: Architektur eines Computers

über die ein Austausch von Informationen in beide Richtungen erfolgen kann. Die Verbindung zwischen Speicher und Prozessor, über die der Datenaustausch erfolgt, wird Bus, Systembus oder Datenbus genannt. Bei den Daten handelt es sich um codierte Objekte, die auf unterster Ebene als diskrete, binäre Einheiten realisiert sind. Eine solche binäre Einheit, die als Zustand 0 oder 1 einnehmen kann, wird als Bit bezeichnet. Im Speicher sind Bits im allgemeinen in Gruppen zu 8 Bit (= Byte), 16 Bit (= Word), 32 Bit (= Longword) oder höher organisiert. In Abhängigkeit zur Größe dieser Gruppen ist der gesamte Speicher in Zellen eingeteilt, die jeweils mit einer eindeutigen Adresse versehen und über diese zu identifizieren sind. Auf abstrakter Ebene sind die Speicherzellen mit den Zellen des Bandes einer Turingmaschine vergleichbar.

[9] Neben Speicher und Hauptprozessor sind noch viele weitere Komponenten, wie z.B. Ein– und Ausgabeeinheiten, Treiberbausteine, Taktgeneratoren usw., zum Betrieb eines Computers notwendig. Speicher werden zudem noch nach Read–Only–Memory (= ROM) und Random–Access–Memory (= RAM) unterteilt. Weitere feinere Abstufungen nach SRAM, DRAM, VRAM etc. sind ebenfalls möglich, doch sind all diese Komponenten für den grundsätzlichen Aufbau eines Computers zunächst irrelevant.

Der Prozessor stellt eine arithmetisch–logische Einheit dar, die über eine bestimmte Anzahl von Funktionen verfügt. Diese Funktionen sind durch festverdrahtete logische Schaltungen realisiert und variieren je nach Prozessortyp. Der vom Prozessor aus erfolgende Zugriff auf Speicherzellen ist immer eine sequentielle Operation. Ein Datenobjekt wird aus einer Speicherstelle gelesen, im Prozessor nach Regeln verarbeitet und in den Speicher zurückgeschrieben, wobei entweder der alte Wert überschrieben wird oder eine andere Speicherzelle den neuen Wert erhält. Die Auswahl der Speicheradressen korrespondiert im wesentlichen mit der Bandbewegung einer Turingmaschine. Wichtig ist hierbei, daß immer nur ein Datenobjekt zu einem Zeitpunkt verarbeitet werden kann.

2.1.3 Programmierung

Geht man davon aus, daß das Programm einer Turingmaschine durch das Alphabet, den aktuellen Zustand und die Regeln des Steuerwerks bestimmt wird, stellt sich die Frage, wie neben dem festverdrahteten Befehlssatz des Prozessors Programme in einem Computer realisiert werden können. Wirth (1983:20f.) bemerkt hierzu:

> Aus der elementaren Forderung, daß ein Computer fähig sein soll, einem gegebenen Programm zu folgen, ergibt sich, daß dieses Programm dem Computer zugänglich sein muß. Wo befindet sich deshalb das Programm zweckmäßigerweise? Es war die geniale (und heute beinahe trivial anmutende) Idee von John von Neumann, das Programm ebenfalls im Speicher unterzubringen. Derselbe Speicher enthält also sowohl die zu bearbeitenden Daten als auch das zu befolgende Programm.

Vor diesem Hintergrund wird die Gesamtarchitektur eines Computers allgemein als *Von-Neumann-Architektur* bezeichnet. Kennzeichen dieser Architektur ist, ebenso wie die Arbeitsweise der Turingmaschine, das Prinzip der sequentiellen und rekursiven Verarbeitung von diskreten symbolischen Objekten durch eine Menge von Steueranweisungen. Beschreiben solche Anweisungen einen Prozeß und dessen Teilhandlungen vollständig, wird die Gesamtheit der hierzu notwendigen Anweisungen *Programm* genannt (vgl. Wirth 1983:14). War es in den fünfziger Jahren noch üblich, Programme an dem Instruktionssatz eines bestimmten Prozessors auszurichten und Befehle in Form binärer, oktaler oder hexadezimaler Folgen zu codieren, so werden Programme heute mit Hilfe „höherer Programmiersprachen" formuliert. Solche Sprachen wie z.B. Fortran, Lisp oder C gestatten es dem Entwickler, Programme in einer von dem aktuell verwendeten Prozessor unabhängigen, problemorientierten und auf die menschlichen Gewohnheiten ausgerichteten Weise zu erstellen. Merkmale dieser symbolischen und regelorientierten Programmierung sind beispielsweise `wenn-dann` Konditionen, `while` oder `for` Schleifen sowie die Bereitstellung von Datenstrukturen wie Zeichenketten, Listen oder Klassen[10].

[10] Klassen, ebenso wie z.B. Methoden und Instanzen, werden von objekt–orientierten Programmiersystemen (= OOP) wie Smalltalk oder C++, bzw. von Systemen mit objekt–orientierten Erweiterungen (z.B. Lisp) verwendet. Objekt–orientierte Systeme sind keine parallel und verteilt arbeitenden Modelle, da sie spezialisierte und komplexe Aufgaben durchführen (vgl. Kemke 1988:3) und keine uniformen Verarbeitungselemente bereitstellen. Sie können daher als Spezialfall sequentieller, symbolischer und regelorientierter Verarbeitung angesehen werden.

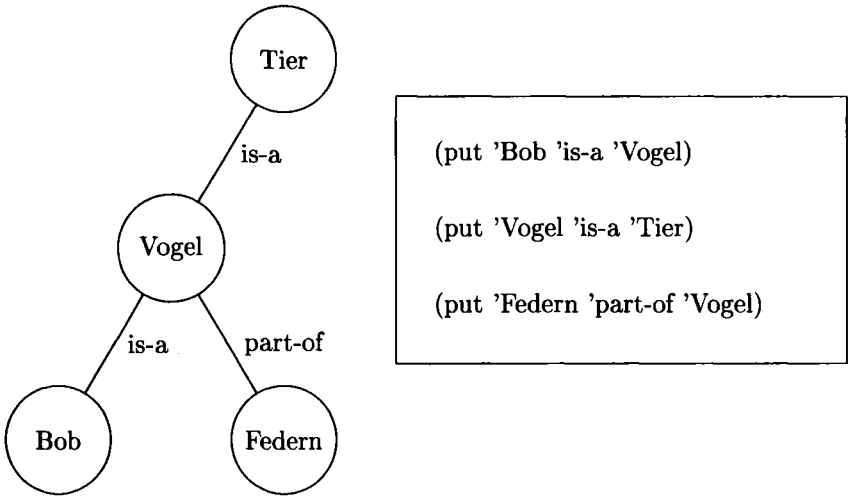

Abb. 2.3: Codierung eines is–a Netzes

Ein Beispiel für die symbolische und regelbasierte Art der Codierung ist in Abbildung 2.3 zu sehen. Sie zeigt ein einfaches hierarchisch organisiertes Netzwerk aus symbolischen Knoten und indizierten Kanten sowie eine dazu äquivalente Beschreibung durch Eigenschaftslisten in der Programmiersprache Lisp[11].

Mittels symbolischer Codierung ist es nun problemlos möglich, eine Sequenz von Anweisungen zusammenzustellen, die Antwort auf die Frage gibt, ob *Federn* ein Teil von *Bob* ist.

```
(cond ((equal (get 'Bob 'is-a)
              (get 'Federn 'part-of)) 'ja)
      (t 'nein))
```

Diese Folge kann in natürlicher Sprache wie folgt beschrieben werden: Wenn (= `cond`) die beiden Terme, die jeweils die is-a Eigenschaft von Bob (= `(get 'Bob 'is-a)`) und part-of Eigenschaft von Federn (= `(get 'Federn 'part-of)`) ermitteln, gleich sind (= `equal`), dann gib `ja` aus, ansonsten (= `t`) gib `nein` aus.

Technisch gesehen werden die Lisp–Anweisungen durch einen Interpreter oder Compiler in einen sogenannten Maschinencode überführt, der vom Prozessor ausgeführt werden kann (vgl. Zaun 1992:6f.). Interpretersprachen führen diesen Code sofort aus und liefern ein Ergebnis zurück. Compilersprachen hingegen nehmen zunächst nur eine Übersetzung vor und generieren damit ein separat vorliegendes, ausführbares

[11] Zu der Programmiersprache Lisp vgl. Winston/Horn 1984. Zu den verwendeten Eigenschaftslisten Zaun 1992:28ff. Mit der Funktion `put` wird einem Symbol (wie z.B. Bob) unter einem beliebigen Indikator (z.B. is-a) ein Wert (z.B. Vogel) zugewiesen. Mit der Funktion `get` kann dann bei Angabe des Symbols und des Indikators der jeweilige Wert abgefragt werden.

Programm in Maschinencode. Die in der KI und Computerlinguistik meistverwendete Programmiersprache Lisp gestattet es zudem, einzelne Funktionen zu compilieren und im Interpretermodus zu verwenden.

2.1.4 Äquivalenzen

Hinsichtlich der Architektur eines informationsverarbeitenden Systems stellt sich nun die Frage, wie diese beschaffen sein muß, um „intelligentes Verhalten" hervorbringen zu können, bzw. bezüglich der Computerlinguistik, die hier im Mittelpunkt des Interesses steht, natürliche Sprache verarbeiten und/oder generieren kann. Für die bisher vorgestellten Systeme, die Turingmaschine und den Computer, ist das gemeinsame Prinzip das der sequentiellen Verarbeitung von Symbolen. Durch diese Arbeitsweise, welche rein mechanisch, d.h. ohne Zuhilfenahme einer Semantik oder einer über bloße Mechanik hinausgehenden Komponente vollzogen wird, kann ein allgemeines Verfahren zur Erkennung bzw. Generierung einer Kette x definiert werden. (vgl. Wall 1972:279f.). Ein solches Verfahren, bei dem sichergestellt werden kann, daß es terminiert, wird als Algorithmus bezeichnet. Betrachtet man noch einmal die Aussage von S.8, daß, wenn x von einer Turingmaschine akzeptiert oder erzeugt wird und x eine rekursiv aufzählbare Menge ist, es einen Algorithmus zur Erzeugung bzw. Generierung aller Elemente von x geben muß, so wird deutlich, daß ein solcher Algorithmus auf alle Automaten übertragbar ist, die zu einer Turingmaschine äquivalent sind. Diese These ist allgemein als die *Church–Turing These* bekannt[12]. Hopcroft/Ullman (1969:80) formulieren dies wie folgt:

> However, from the definition of a Turing machine, it will be readily apparent that any computation that can be described by means of a Turing machine can be mechanically carried out. Thus the definition is not too broad. It can also be shown that any computation that can be performed on a modern–day digital computer can be described by means of a Turing machine.

Dorffner (1991:7) verdeutlicht diese Aussage, indem er explizit sagt:

> Jede berechenbare Funktion kann von einer Turing–Maschine berechnet werden, somit auch von jeder Maschine, die einer Turingmaschine äquivalent ist.
> Ein von Neumann Computer, mit theoretisch beliebig viel Speicherraum, ist einer Turingmaschine äquivalent.

Letztendlich führt diese Behauptung zu der Annahme, daß die physikalische Beschaffenheit eines informationsverarbeitenden Systems völlig irrelevant für die Belange der KI und der Computerlinguistik ist. Wichtig ist nur, daß das System den Prinzipien der sequentiellen rekursiven Symbolverarbeitung folgt und diese auf einer beliebigen physikalischen Struktur realisieren kann. Dabei ist es unwichtig, auf welcher Grundlage ein solches System arbeitet. Newell (1973:44f.) bemerkt:

[12] Minsky (1967:108) bemerkt hierzu: „Turing goes on to defend the following proposition, now often called *Turing's thesis*: Any process which could naturally be called an effective procedure can be realized by a Turing machine. This proposition, in its most general form, is usually called *Church's thesis*, after the work of Alonzo Church relating the intuitive notion of effectiveness to formal logical process."

> Mind is an information-processing system.
> Minds (information-processing systems) are realized in physical systems.
> (...)
> Neurophysiology is almost totally irrelevant to the nature of mind.
> (...)
> Electronics is almost totally irrelevant to the nature of artificial intelligence.

Diese Sichtweise impliziert, daß auch die menschliche Kognition vollständig durch symbolmanipulierende Verfahren beschreibbar ist, bzw. auf diesen beruht.

> The view that man is an information processor means that his behavior can be seen as the result of a system consisting of memories containing discrete symbols and symbolic expressions (i.e., occurrences of symbols), and processes that manipulate these symbols. The central notion is that of the symbol, which is taken to mean essentially what it does in computer science, an entity with a certain functional property, to wit: that when a process has a token of a symbol it has access to information about what that symbol designates (encoded in symbolic expressions). The processes that can be performed on symbols are their creation (and, possibly, destruction), the obtaining of designated information, the creation of symbolic expressions, and the manipulation of these symbolic expressions by insertion, deletion, replacement, and reordering.
>
> (Newell 1973:27)

Newell/Simon (1976) formulierten hieraus die sogenannte *Physical Symbol Systems Hypothesis*:

> Ein 'Physical Symbol System' hat die notwendigen und hinreichenden Voraussetzungen für ein intelligentes System.
> Def.: Ein 'Physical Symbol System' ist ein System, das aus Symbolen und Symbolstrukturen besteht, die physikalisch realisiert sein müssen.
>
> (zitiert nach Dorffner 1991:3f.)

Es stellt sich jedoch die Frage, ob es sich bei der menschlichen Kognition tatsächlich um ein physikalisches System handelt, auf das ein wie auch immer gearteter symbolverarbeitender Mechanismus aufgesetzt ist. Trifft diese Annahme zu, so kann auch behauptet werden, daß sich der menschliche Geist wie ein Computerprogramm verhält, das auf einer neuronalen Architektur implementiert ist. In den Kognitionswissenschaften wird dies als *Computermetapher* bezeichnet (vgl. Schwarz 1992:15f.). Geist verhält sich danach zum Gehirn wie die Software zur Computerhardware. Die Computermetapher scheint allerdings auszuklammern, daß die Hardware eines Computers durch Vorgabe hochkomplexer Logikschaltungen bereits strukturiert ist und damit über „Wissen" verfügt. Zudem kann auf Software vollständig verzichtet werden, wenn Prozesse durch feste Verschaltungen auf einem Chip realisiert werden. In diesem Fall gibt es keine formale Trennung zwischen Software und Hardware mehr. Das Wissen eines solchen Systems liegt somit in der physikalischen Verschaltung der Hardware. Die Sichtweise der Computermetapher impliziert hingegen, daß es sich bei dem menschliche Geist, ebenso wie bei der Turingmaschine und dem Computer, um ein sequentielles und symbolmanipulierendes System handelt. Folglich basiert auch die Fähigkeit zur Sprachverarbeitung auf diesen Prinzipien und kann aufgrund der Church–Turing These und der Physical Symbol System Hypothesis auf Computern nachgebildet werden.