Algorithmisch Rekursive Sequenzanalyse Algorithmischer Strukturalismus: Formalisierung genetischer Strukturalismus: Ein Versuch, dazu beizutragen, den genetischen Strukturalismus falsifizierbar zu machen

Paul Koop M.A.
post@paul-koop.org

June 21, 2023

Abstract

Es wird eine Methode zur Analyse von diskreten endlichen Zeichenketten vorgestellt. Die postmoderne Sozialphilosophie wird zurückgewiesen. Zugestimmt wird einer naturalistischen Soziologie mir falsifizierbaren Modellen für Handlungssysteme. Vorgestellt wird die Algorithmisch rekursive Sequenzanalyse (Aachen 1994) mit der Definition einer formalen Sprache für soziale Handlungen, einem Grammatikinduktor (Scheme), einem Parser (Pascal) und einem Grammatiktransduktor (Lisp).

Die Algorithmisch Rekursive Sequenzanalyse (Aachen 1994) ist eine Methode zur Analyse endlicher diskreter Zeichenketten. Ndiaye, Alassane (Rollenübernahme als Benutzermodellierungsmethode : globale Antizipation in einem transmutierbaren Dialogsystem 1998) und Krauße, C. C., Krueger,F.R. (Unbekannte Signale 2002) veröffentlichten äquivalente Methoden. Genial ist, etwas Einfaches einfach zu denken. Seit Anfang des 21. Jahrhunderts wird die Konstruktion von Grammatiken aus gegebenen empirischen Zeichenketten in der Computerlinguistik unter dem Stichwort Grammatikinduktion diskutiert (Alpaydin, E. 2008: Maschinelles Lernen, Shen, Chunze 2013: Effiziente Grammatikinduktion, Dehmer (2005) Strukturelle Analyse, Krempel 2016: Netze, Karten, Irrgärten). Mit sequitur definieren Nevill-Manning und Witten (Nevill-Manning Witten 1999: Identifying Hierarchical Structure in Sequences: A linear-time algorithm 1999) eine Grammar Induktion zur Komprimierung von Zeichenketten. Graphen, Grammatiken und Transformationsregeln sind natürlich erst der Anfang. Denn eine Sequenzanalyse ist erst abgeschlossen, wenn, wie bei der algorithmisch rekursiven Sequenzanalyse mindestens eine Grammatik angegeben werden kann für die ein Parser die Sequenz als wohlgeformt identifiziert, mit der ein Transduktor künstliche Protokolle erzeugen kann, die

äquivalent zur untersuchten empirschen Sequenz sind und zu der ein Induktor mindestens eine äquivalente Grammatik erzeugen kann. Gold (1967) formulierte das Problem in Antwort auf Chomsky (1965). Der algorithmische Strukturalismus ist widerspruchsfrei, empirisch bewährt, galileisch, naturalistisch, darwinisch und ein Ärgernis für tiefenhermeneutische, konstruktivistische, postmodernistische und (post)strukturalistische Sozialphilosophen. Ich freue mich über Erben, die die Arbeit fortsetzen oder sich inspirieren lassen. Eine soziale Handlung ist ein Ereignis im Möglichkeitsraum aller sozialen Handlungen. Der Sinn einer sozialen Handlung ist die Menge aller möglichen Anschlusshandlungen und ihrer Auftrittswahrscheinlichkeit. Der Sinn muss nicht deutend verstanden werden, sondern kann empirisch rekonstruiert werden. Die Rekonstruktion kann durch Bewährungsversuche an empirischen Protokollen bewährt oder falsifiziert werden. Seit Mitte der 1970er bis heute finden irrationalistische oder antirationalistische Ideen unter akademischen Soziologen in Amerika, Frankreich, Großbritannien und Deutschland zunehmend Verbreitung. Die Ideen werden als Dekonstruktionismus, Tiefenhermeneutik, Wissenssoziologie, Sozialkonstruktivismus, Konstruktivismus oder Wissenschafts- und Technologieforschung bezeichnet. Der Oberbegriff für diese Bewegungen ist (Post)strukturalismus oder Postmodernismus. Alle Formen des Postmodernismus sind antiwissenschaftlich, antiphilosophisch, antistrukturalistisch, antinaturalistisch, antigalileisch, antidarwinisch und allgemein antirational. Die Sicht der Wissenschaft als eine Suche nach Wahrheiten (oder annähernden Wahrheiten) über die Welt wird abgelehnt. Die natürliche Welt spielt eine kleine oder gar keine Rolle bei der Konstruktion wissenschaftlichen Wissens. Die Wissenschaft ist nur eine andere soziale Praxis, die Erzählungen und Mythen hervorbringt, die nicht mehr Gültigkeit haben als die Mythen vorwissenschaftlicher Epochen. Man kann den Gegenstand der Sozialwissenschaften so beobachten, wie die Astronomie ihren Gegenstand beobachtet. Wenn sich der Gegenstand der Sozialwissenschaften dem direkten Zugang oder Laborexperiment so entzieht, wie Himmelsobjekte (Gerichtsverhandlung, Verkaufsgespräch, Vorstandssitzung, et cetera), bleibt nur, ihn interpretationsfrei rein physikalisch zu beobachten und die Beobachtungen rein physikalisch zu protokollieren. Die Protokolle könnte man dann natürlich auch ohne Rückbindung an Physik, Chemie, Biologie, Evolutionsbiologie, Zoologie, Primatenforschung und Lifescience interpretieren. Diese überprüfungsfreie Interpretation nennt man bei der Himmelsbeobachtung dann Astrologie. In den Sozialwissenschaften nennt man auch diese überprüfungsfreie Interpretation Soziologie. Beispiele sind Konstruktivismus (Luhmann), systemische Heilslehren, Postmodernismus, Poststrukturalismus, oder Theorie kommunikativen Handelns (Habermas). Regelbasierte Agentenmodelle arbeiten deshalb bisher mit heuristischen Regelsystemen. Diese Regelsysteme sind nicht empirisch bewährt. Wie auch in der Astrologie könnte man damit natürlich auch in der Soziologie Computermodelle erstellen, die ebenso wie astrologische Modelle wenig empirischen Erklärungsgehalt hätten. Einige nennen das Sozionik. Man kann aber auch die Protokolle unter Beachtung von Physik, Chemie, Biologie, Evolutionsbiologie, Zoologie, Primatenforschung und Lifescience interpretieren und auf empirische Gültigkeit überprüfen. Die

Beobachtung von Himmelsobjekten nennt man dann Astronomie. In den Sozialwissenschaften könnte man von Sozionomie oder Soziomatik sprechen. Das eigentlich ist Soziologie. Heraus kämen dabei keine grossen Weltanschauungen, sondern wie in der Astronomie Modelle mit begrenzter Reichweite, die empirisch überprüfbar sind und an Evolutionsbiologie, Zoologie, Primatenforschung und Lifescience anschlussfähig sind. Diese Modelle (Differentialgleichungen, formale Sprachen, Zellularautomaten, etc) liessen die Deduktion empirisch überprüfbarer Hypothesen zu, wären also falsifizierbar. Eine solche Sozionomie oder Soziomatik gibt es noch nicht. Ich würde formale Sprachen als Modellsprachen für empirisch bewährte Regelsysteme bevorzugen. Denn Regelsysteme für Gerichtsverhandlungen oder Verkaufsgespräche z.B. (Modelle mit begrenzter Reichweite, Multiagentensysteme, zelluläre Automaten) lassen sich eher mit formalen Sprachen als mit Differentialgleichungen modellieren. Der Algorithmische Strukturalismus ist ein Versuch, dazu beizutragen, den genetischen Strukturalismus (ohne Auslassung und ohne Hinzufügung) in eine falsifizierbare Form zu übersetzen und empirisch bewährte Regelsysteme zu ermöglichen. Die Algorithmisch Rekursive Sequenzanalyse ist der erste systematische Versuch, einer naturalistischen und informatischen Ausformulierung des genetischen Strukturalismus als memetisches und evolutionäres Modell. Die Methodologie der Algorithmisch Rekursiven Sequenzanalyse ist der Algorithmische Strukturalismus. Der Algorithmische Strukturalismus ist eine Formalisierung des genetischen Strukturalismus.Der genetische Strukturalismus (Oevermann) unterstellt einen intentionsfreien, apsychischen Möglichkeitsraum algorithmischer Regeln, die die Pragmatik wohlgeformter Ereignisketten textförmig strukturieren (Chomsky, McCarthy, Papert, Solomon, Lévi-Strauss, de Saussure, Austin, Searle). Der Algorithmische Strukturalismus ist der Versuch den genetischen Strukturalismus falsifizierbar zu machen. Der Algorithmische Strukturalismus ist galileisch und an Habermas und Luhmann so wenig anschlußfähig, wie Galilei an Aristoteles. Natürlich kann man sich bemühen, an Luhmann oder Habermas anschlussfähig zu bleiben und Luhmann oder Habermas zu algorithmisieren. Algorithmisieren kann man alle Artefakte, zum Beispiel die Astrologie oder das Schachspiel. Und man kann normative Agenten verteilter künstlicher Intelligenz, Zelluläre Automaten, neuronale Netze und andere Modelle mit heuristischen Protokollsprachen und Regeln modellieren. Das ist zweifellos theoretisch wertvoll. So wird es keinen soziologischen Theoriefortschritt geben. Gesucht ist eine neue Soziologie, die die Replikation, Variation und Selektion sozialer Replikatoren, gespeichert in Artefakten und neuralen Mustern, modelliert. Diese neue Soziologie wird an Habermas oder Luhmann ebenso wenig anschlussfähig sein wie Galilei an Aristoteles. Und ihre basalen Sätze werden so einfach sein wie die newtonschen Gesetze. So wie Newton die Begriffe Bewegung, Beschleunigung, Kraft, Körper und Masse operational definierte, so wird diese Theorie die sozialen Replikatoren, ihre materiellen Substrate, ihre Replikation, Variation und Selektion algorithmisch und operational definieren und sequenzanalytisch sichern. Soziale Strukturen sind sprachlich codiert und basieren auf einem digitalen Code. Gesucht sind syntaktische Strukturen einer Kultur codierenden Sprache. Aber dies wird keine philosophische Sprache sein,

sondern eine Sprache, die Gesellschaft codiert und erschafft. Diese Sprache codiert die Replikation, Variation und Selektion kultureller Replikatoren. Auf dieser Basis werden dann normative Agenten verteilter künstlicher Intelligenz, Zelluläre Automaten, neuronale Netze und andere Modelle andere als heuristische Protokollsprachen und Regelsysteme nutzen können, um die Evolution kultureller Replikatoren zu simulieren. Thematisch bewegt sich der Algorithmische Strukturalismus im Grenzgebiet zwischen Informatik und Soziologie. Die Algorithmische Strukturalismus unterstellt, dass die soziale Wirklichkeit selbst (Wetware, Welt 2) nicht kalkülfähig ist. Die soziale Wirklichkeit hinterlässt bei ihrer Reproduktion und Transformation rein physikalisch semantisch unspezifische Spuren (Protokolle, Hardware, Welt 1). Diese Spuren können als Texte (diskrete endliche Zeichenketten, Software, Welt 3) verstanden werden. Es wird dann gezeigt, dass eine Approximation der Transformationregeln der sozialen Wirklichkeit (latente Sinnstrukturen, Regeln im Sinne von Algorithmen) durch Konstruktion formaler Sprachen (Welt 3, Software) möglich ist. Diese Methode ist die Algorithmisch Rekursive Sequenzanalyse. Diese linguistische Struktur ist der Motor der memetischen Reproduktion kultureller Replikatoren. Diese algorithmisch rekursive Struktur ist natürlich nicht (sic!) an Habermas und Luhmann anschlußfähig. Galilei ist ja auch nicht an Aristoteles anschlußfähig! Durch Lesartenproduktion und Lesartenfalsifikation wird Sequenzstelle für Sequenzstelle informell das Regelsystem erzeugt. Das informelle Regelsystem wird in ein K-System übersetzt. Mit dem K-System wird dann eine Simulation duchgeführt. Das Ergebnis der Simulation, eine terminale endliche Zeichenkette, wird mit der empirisch gesicherten Spur statistisch verglichen. Das bedeutet nicht, dass Subjekte in irgendeinem Sinne von Bedeutung Regeln im Sinne von Algorithmen folgen. Die soziale Wirklichkeit ist unmittelbar nur sich selbst zugänglich. Völlig unzugänglich sind die inneren Zustände der Subjekte. Aussagen über diese inneren Zustände von Subjekten sind Derivate aus den gefundenen latenten Sinnstrukturen, Regeln im Sinne von Algorithmen. Bevor eine Annahme über den inneren Zustand eines Subjektes formuliert werden kann, müssen zuerst diese latenten Sinnstrukturen, Regeln im Sinne von Algorithmen, als Möglichkeitsraum von Sinn und Bedeutung gültig bestimmt werden. Sinn meint nicht ein ethisch gutes, ästhetisch schönes oder empathisch nachvollzogenes Leben, sondern einen intelligiblen Zusammenhang, Regeln im Sinne von Algorithmen. Die latenten Sinnstrukturen, Regeln im Sinne von Algorithmen, erzeugen diachronisch eine Kette von Selektionsknoten (Parameter I), wobei sie synchronisch zum Zeitpunkt t aus dem Selektionsknoten t den Selektionsknoten t+1 erzeugen (Parameter II). Dem entspricht eine kontextfreie formale Sprache (K-Systeme), die aus dem Selektionsknoten zum Zeitpunkt t durch Anwendung von Produktionsregeln den Selektionsknoten t+1 erzeugt. Dabei ist jeder Selektionsknoten ein Zeiger auf rekursiv ineinander verschachtelte K-Systeme. So kann wie mit einem Mikroskop in die Fallstruktur hineingezoomt werden. Die Menge der K-Systeme bilden eine Case Structure Modelling Language "CSML". Die Approximation lässt sich beliebig nahe an die Transformation der sozialen Wirklichkeit annähern. Dabei werden den Produktionen Maße zugeordnet, die ihrer empirischen gesicherten Pragmatik/Semantik entsprechen.

Sie bilden topologisch ein rekursives Transitionsnetz diskreter nichtmetrischer Ereignismengen über denen ein algorithmisches Regelsystem arbeitet. K- Systeme K sind formal durch ein Alphabet

$$A = \{a_1, a_2, \ldots, a_n\},$$

alle Worte über dem Alphabet

$$A^*,$$

Produktionsregeln

$$p,$$

dem Auftrittsmaß

$$h$$

(Pragmatik/Semantik)und einer axiomatischen ersten Zeichenkette

$$k_0 \in A^*$$

definiert:

K-System:

$$\begin{aligned} K &= (A, P, k_0) \\ A &= \{a_1, a_2, \ldots, a_n\} \\ P &:= A \to A \\ p_{a_i} &\in P \\ p_{a_i} &: A \times H \times A \\ H &= \{h \in N \mid 0 \leq h \leq 100\} \\ k_0 &\in A^* \\ k_i &\in A \quad (i \geq 1) \end{aligned}$$

Das Auftrittsmaß

$$h$$

läßt sich spieltheoretisch expandieren (vgl. Diekmann). Vom Axiom

$$k_0$$

ausgehend, erzeugt ein K-System eine Zeichenkette

$$k_0 k_1 k_2 ...$$

indem die Produktionsregel p auf das Zeichen i einer Kette angewendet wird:

$$\begin{aligned} \mathrm{a}_{i+1} &:= p_{a_i}(a_i) \\ k_i &:= a_{i-2} a_{i-1} a_i \\ k_{i+1} &:= a_{i-2} a_{i-1} a_i p_{a_i}(a_i) \end{aligned}$$

Ein strenges Maß für die Zuverlässigkeit der Zuordnung der Interakte zu den Kategorien (vorläufige Formative da im Prinzip ad infinitum zu approximieren) ist die Anzahl der von allen Interpreten übereinstimmend vorgenommene Zuordnung (vgl MAYRING l99O,S.94ff, LISCH/KRIZ1978,S.84ff). Diese Zahl muss dann noch durch Relativierung um die Anzahl der Interpreten normalisiert werden. Dieser Koeffizient ist dann definiert mit:

$$R_{\text{ars}} = 0.59, \quad p = 0.05$$

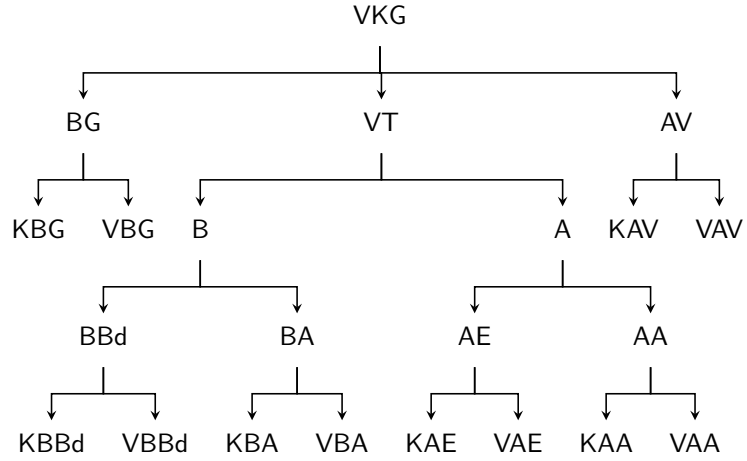
Zwischen 1993 und 1996 habe ich ein K-System für Verkaufsgespräche auf Wochenmärkten rekonstruiert und empirisch abgesichert (Koop, P. 1993, 1994, 1995, 1996 siehe Anhang).

Die Regeln lassen sich als Kontextfreie Grammatik darstellen.

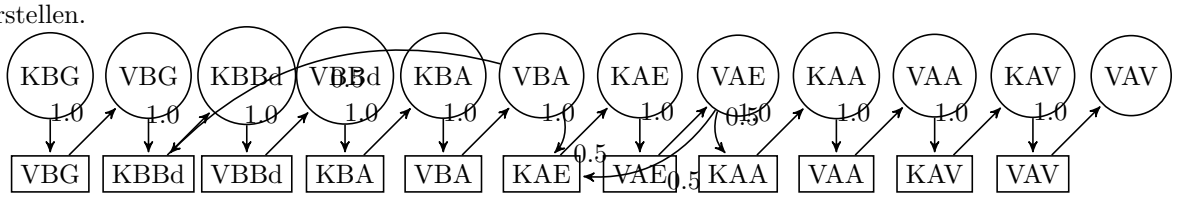
Produktionsregeln:

$$\begin{aligned}
\text{VKG} &\rightarrow \text{BG VT AV} \\
\text{BG} &\rightarrow \text{KBG VBG} \\
\text{VT} &\rightarrow \text{B A} \\
\text{B} &\rightarrow \text{BBd BA} \\
\text{BBd} &\rightarrow \text{KBBd VBBd} \\
\text{BA} &\rightarrow \text{KBA VBA} \\
\text{A} &\rightarrow \text{AE AA} \\
\text{AE} &\rightarrow \text{KAE VAE} \\
\text{AA} &\rightarrow \text{KAA VAA} \\
\text{AV} &\rightarrow \text{KAV VAV}
\end{aligned}$$

Die Grammatik läßt sich als Strukturbaum darstellen.



Das Korpus aus Terminalzeichen läßt sich als Graph (z.B. Petri-Netz) darstellen.



Die Zeichen der Zeichenkette sind ohne vordefinierte Bedeutung. Theoretisch relevant ist allein die Syntax ihrer Verknüpfung. Sie definiert die Fallstruktur. Die semantische Interpretation der Zeichen ist alleine eine Interpretationsleistung eines menschlichen Lesers. Im Prinzip ist auch eine visuelle Interpretation (die animiert werden kann) etwa zur automatischen Synthese von Filmsequenzen möglich.:

Ein menschlicher Leser kann die Zeichen interpretieren:

Verkaufsgespräche	VKG
Verkaufstätigkeit	VT
Bedarfsteil	B
Abschlußteil	A
Begrüßung	BG
Bedarf	Bd
Bedarfsargumentation	BA
Abschlußeinwände	AE
Verkaufsabschluss	AA
Verabschiedung	AV
vorangestelltes K	Kunde
vorangestelltes V	Verkäufer

Soziale Strukturen und Prozesse hinterlassen rein physikalisch und semantisch unspezifische Spuren, die als Protokolle ihrer Reproduktion und Transformation gelesen werden können. So gelesen sind die Protokolle Texte, diskrete endliche Zeichenkette. Die Regeln der Reproduktion und Transformation können als probabilistische, kontextfreie Grammatiken oder als Bayessche Netze rekonstruiert werden. Die Rekonstruktion steht dann für eine kausale Inferenz der Transformationsregeln der sozialen Strukturen und Prozesse. In dem hier vorliegenden Beispiel ist das Protokoll eine Tonbandaufnahme eines Verkaufsgespräches auf einem Wochenmarkt: (https://github.com/pkoopongithub/ algorithmisch-rekursive-sequenzanalyse/). Die Sequenzanalyse des transkribierten Protokolls und die Kodierung mit den generierten Kategorien ist dort auch abgelegt. Die Interpretation und die Kodierung mit den Terminalzeichen ist auch in einem Anhang zu diesem Rext in einer separaten Datei abgelegt.

```
1
2  ;; Korpus
3     (define korpus (list 'KBG 'VBG 'KBBd 'VBBd 'KBA '
          VBA 'KBBd 'VBBd 'KBA 'VBA 'KAE 'VAE 'KAE 'VAE '
          KAA 'VAA 'KAV 'VAV));; 0 - 17
4
5
6
7  ;; Korpus durchlaufen
8      (define (lesen korpus)
9       ;; car ausgeben
10     (display (car korpus))
11      ;; mit cdr weitermachen
12     (if(not(null? (cdr korpus)))
13       (lesen (cdr korpus))
14       ;;(else)
15      )
16    )
17
18 ;; Lexikon
19    (define lexikon (vector 'KBG 'VBG 'KBBd 'VBBd 'KBA
          'VBA 'KAE 'VAE 'KAA 'VAA 'KAV 'VAV)) ;; 0 - 12
20
21
22
23 ;; Index fuer Zeichen ausgeben
24     (define (izeichen zeichen)
25      (define wertizeichen 0)
26      (do ((i 0 (+ i 1)))
27       ( (equal? (vector-ref lexikon i) zeichen))
28       (set! wertizeichen (+ 1 i))
29      )
```

```
30      ;;index zurueckgeben
31      wertizeichen
32    )
33
34  ;; transformationsmatrix
35    (define zeile0 (vector 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
          0 0 0 0))
36    (define zeile1 (vector 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
          0 0 0 0))
37    (define zeile2 (vector 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
          0 0 0 0))
38    (define zeile3 (vector 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
          0 0 0 0))
39    (define zeile4 (vector 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
          0 0 0 0))
40    (define zeile5 (vector 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
          0 0 0 0))
41    (define zeile6 (vector 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
          0 0 0 0))
42    (define zeile7 (vector 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
          0 0 0 0))
43    (define zeile8 (vector 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
          0 0 0 0))
44    (define zeile9 (vector 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
          0 0 0 0))
45    (define zeile10 (vector 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
          0 0 0 0))
46    (define zeile11 (vector 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
          0 0 0 0))
47    (define zeile12 (vector 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
          0 0 0 0))
48    (define zeile13 (vector 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
          0 0 0 0))
49    (define zeile14 (vector 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
          0 0 0 0))
50    (define zeile15 (vector 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
          0 0 0 0))
51    (define zeile16 (vector 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
          0 0 0 0))
52    (define zeile17 (vector 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
          0 0 0 0))
53
54    (define matrix (vector zeile0 zeile1 zeile2 zeile3
          zeile4 zeile5 zeile6 zeile7 zeile8 zeile9
          zeile10 zeile11 zeile12 zeile13 zeile14 zeile15
          zeile16 zeile17))
```

```
55
56
57 ;; Transformationen zaehlen
58 ;; Korpus durchlaufen
59    (define (transformationenZaehlen korpus)
60      ;; car zaehlen
61       (vector-set! (vector-ref matrix (izeichen (car
             korpus))) (izeichen (car(cdr korpus))) (+ 1 (
             vector-ref  (vector-ref matrix (izeichen (car
              korpus))) (izeichen (car(cdr korpus))))))
62      ;; mit cdr weitermachen
63       (if(not(null? (cdr (cdr korpus))))
64        (transformationenZaehlen (cdr korpus))
65        ;;(else)
66       )
67    )
68
69
70 ;; Transformation aufaddieren
71
72 ;; Zeilensummen bilden und Prozentwerte bilden
73
74
75 ;; Grammatik
76    (define grammatik (list '- ))
77
78 ;; aus matrix regeln bilden und regeln in grammatik
       einfuegene
79    (define (grammatikerstellen matrix)
80     (do ((a 0 (+ a 1)))
81         ((= a 12) )(newline)
82       (do ((b 0 (+ b 1)))
83           ((= b 12))
84         (if (< 0 (vector-ref  (vector-ref matrix a) b)
               )
85          (display (cons (vector-ref lexikon a) (cons
               '-> (vector-ref lexikon b))))
86          )
87       )
88     )
89    )
90
91
92   ;; matrix ausgeben
93    (define (matrixausgeben matrix)
94     (do ((a 0 (+ a 1)))
```

```
95          ((= a 12) ) (newline)
96        (do ((b 0 (+ b 1)))
97            ((= b 12))
98          (display (vector-ref  (vector-ref matrix a) b)
               )
99        )
100     )
101    )
```

```
 (transformationenZaehlen korpus)
(grammatikerstellen matrix) (KBG -> .  VBG) (VBG
-> .  KBBd) (KBBd -> .  VBBd) (VBBd -> .  KBA) (KBA
-> .  VBA) (VBA -> .  KBBd)(VBA -> .  KAE) (KAE -> .
VAE) (VAE -> .  KAE)(VAE -> .  KAA) (KAA -> .  VAA)
(VAA -> .  KAV) (KAV -> .  VAV)
```

Figure 1: ASCII-Output des Konsolenprogramms

Mit dieser Grammatik und den empirischen Auftrittswahrscheinlichkeiten lässt sich dann ein Transduktor erstellen, der Protokolle simuliert.

```
1  \begin{verbatim}
2
3
4
5  (setq w3
6  '(
7   (anfang 100 (s vkg)) ;; hier nur Fallstruktur
       Verkaufsgespraeche
8   ((s vkg) 100 ende)
9   )
10 )
11
12
13
14 (setq bbd
15 '(
16  (kbbd 100 vbbd)
17  )
18 )
19
20
21 (setq ba
22 '(
23  (kba 100 vba)
24  )
```

```
)


(setq ae
'(
(kae 100 vae)
 )
)


(setq aa
'(
 (kaa 100 vaa)
 )
)


(setq b
'(
 ((s bbd) 100 (s ba))
 )
)


(setq a
'(
 ((s ae)50(s ae))
 ((s ae)100(s aa))
 )
)


(setq vt
'(
 ((s b)50(s b))
 ((s b)100(s a))
 )
)


(setq bg
```

```
71  '(
72   (kbg 100 vbg)
73   )
74  )
75
76
77
78  (setq av
79  '(
80   (kav 100 vav)
81   )
82  )
83
84
85
86  (setq vkg
87  '(
88   ((s bg)100(s vt))
89   ((s vt)50(s vt))
90   ((s vt)100(s av))
91   )
92  )
93
94
95
96
97  ;; Generiert die Sequenz
98  (defun gs (st r);; Uebergabe Sequenzstelle und
        Regelliste
99  (cond
100
101   ;; gibt nil zur ck , wenn das Sequenzende ereicht
        ist
102   ((equal st nil) nil)
103
104   ;; gibt terminale Sequenzstelle mit Nachfolgern
        zurueck
105   ((atom st)(cons st(gs(next st r(random 101))r)))
106
107   ;; gibt expand. nichtterm. Sequenzstelle mit
        Nachfolger zurueck
108   (t (cons(eval st)(gs(next st r(random 101))r)))
109 )
110 )
111
112 ;; Generiert nachfolgende Sequenzstelle
```

```
113 (defun next (st r z);; Sequenzstelle, Regeln und
       Haeufigkeitsmass
114 (cond
115
116   ;; gibt nil zurueck, wenn das Sequenzende erreicht
         ist
117   ((equal r nil)nil)
118
119   ;; waehlt Nachfolger mit Auftrittsmass h
120   (
121     (
122        and(<= z(car(cdr(car r))))
123        (equal st(car(car r)))
124     )
125    (car(reverse(car r)))
126   )
127
128   ;; in jedem anderen Fall wird Regelliste weiter
         durchsucht
129   (t(next st (cdr r)z))
130 )
131 )
132
133 ;; waehlt erste Sequenzstelle aus Regelliste
134 ;;vordefinierte funktion first wird ueberschrieben,
       alternative umbenennen
135 (defun first (list)
136 (car(car list))
137 )
138
139 ;; startet Simulation fuer eine Fallstruktur
140 (defun s (list) ;; die Liste mit dem K-System wird
       uebergeben
141 (gs(first list)list)
142 )
143
144
145 ;; alternativ (s vkg) / von der Konsole aus (s w3)
       oder (s vkg)
146 (s w3)
```

Ein umfangreicheres und um die Klammern bereinigtes Beispiel:
Das Korpus kann jetzt auch, da die Grammatik gegeben ist, geparst werden.

```
1 PROGRAM parser (INPUT,OUTPUT);
2 USES CRT;
```

```
CL-USER 20 > (s w3) (ANFANG ((KBG VBG) (((KBBD VBBD)
(KBA VBA)) ((KAE VAE) (KAA VAA))) (((KBBD VBBD)
(KBA VBA)) ((KAE VAE) (KAA VAA))) (((KBBD VBBD) (KBA
VBA)) ((KBBD VBBD) (KBA VBA)) ((KAE VAE) (KAA VAA)))
(((KBBD VBBD) (KBA VBA)) ((KBBD VBBD) (KBA VBA))
((KBBD VBBD) (KBA VBA)) ((KAE VAE) (KAA VAA))) (KAV
VAV)) ENDE)
```

Figure 2: ASCII-Output des Konsolenprogramms

```
KBG VBG KBBD VBBD KBA VBA KAE VAE KAA VAA KBBD VBBD
KBA VBA KBBD VBBD KBA VBA KBBD VBBD KBA VBA KAE VAE
KAA VAA KAV VAV KBG VBG KBBD VBBD KBA VBA KAE VAE
KAE VAE KAE VAE KAE VAE KAA VAA KBBD VBBD KBA VBA
KAE VAE KAE VAE KAA VAA KBBD VBBD KBA VBA KAE VAE
KAA VAA KBBD VBBD KBA VBA KBBD VBBD KBA VBA KAE VAE
KAA VAA KAV VAV KBG VBG KBBD KBA VBA KBBD VBBD KBA
VBA KAE VAE KAE VAE KAA VAA KBBD VBBD KBA VBA KBBD
KBA VBA KBBD VBBD KBA VBA KBBD VBBD KBA KAE VAE KAA
VAA KBBD VBBD KBA VBA KAE VAE KAE VAE VAE KAA VAA
KAV VAV
```

Figure 3: ASCII-Output des Konsolenprogramms

```
3
4
5   CONST
6     c0                =      0;
7     c1                =      1;
8     c2                =      2;
9     c3                =      3;
10    c4                =      4;
11    c5                =      5;
12   c10                =     10;
13   c11                =     11;
14   cmax               =     80;
15   cwort              =     20;
16   CText              :     STRING(.cmax.) = '';
17   datei              =     'LEXIKONVKG.ASC';
18   blank              =     '␣';
19
20   CopyRight
21   =     'Demo-Parser␣Chart-Parser␣Version␣1.0(c)1992␣
        by␣Paul␣Koop';
22
23  TYPE
24   TKategorien        = ( Leer, VKG, BG, VT, AV, B, A,
        BBD, BA, AE, AA,
25                          KBG, VBG, KBBD, VBBD, KBA,
                               VBA, KAE, VAE,
26                          KAA, VAA, KAV, VAV);
27
28
29   PTKategorienListe = ^TKategorienListe;
30   TKategorienListe  = RECORD
31                         Kategorie :TKategorien;
32                         weiter    :PTKategorienListe;
33                       END;
34
35   PTKante           = ^TKante;
36   PTKantenListe     = ^TKantenListe;
37
38   TKantenListe      = RECORD
39                         kante:PTKante;
40                         next :PTKantenListe;
41                       END;
42
43   TKante            = RECORD
44                         Kategorie :TKategorien;
45                         vor,
```

```
46                          nach,
47                          zeigt     :PTKante;
48                          gefunden  :PTKantenListe;
49                          aktiv     :BOOLEAN;
50                          nummer    :INTEGER;
51                          nachkomme :BOOLEAN;
52                          CASE  Wort:BOOLEAN OF
53                           TRUE :
54                               (inhalt:STRING(.cwort.);)
                                    ;
55                           FALSE:
56                               (gesucht :
                                    PTKategorienListe;);
57                          END;
58
59
60    TWurzel    = RECORD
61                    spalte,
62                    zeigt     :PTKante;
63                  END;
64
65    TEintrag    = RECORD
66                  A,I   :PTKante;
67                  END;
68
69    PTAgenda    = ^TAgenda;
70    TAgenda     = RECORD
71                   A,I  :PTKante;
72                   next,
73                   back : PTAgenda;
74                  END;
75
76    PTLexElem   = ^TLexElem;
77    TLexElem    = RECORD
78                   Kategorie: TKategorien;
79                   Terminal : STRING(.cwort.);
80                   naechstes: PTLexElem;
81                  END;
82
83    TGrammatik  = ARRAY (.c1..c10.)
84                  OF
85                  ARRAY (.c1..c4.)
86                  OF TKategorien;
87   CONST
88    Grammatik :       TGrammatik =
89                (
```

```
90                (VKG, BG,      VT,    AV),
91                (BG,  KBG,     VBG,   Leer),
92                (VT,  B,       A,     Leer),
93                (AV,  KAV,     VAV,   Leer),
94                (B,   BBd,     BA,    Leer),
95                (A,   AE,      AA,    Leer),
96                (BBd, KBBd,    VBBd,  Leer),
97                (BA,  KBA,     VBA,   Leer),
98                (AE,  KAE,     VAE,   Leer),
99                (AA,  KAA,     VAA,   Leer)
100              );
101
102  nummer :INTEGER = c0;
103
104
105
106  VAR
107   Wurzel,
108   Pziel       : TWurzel;
109   Pneu        : PTKante;
110
111   Agenda,
112   PAgenda,
113   Paar        : PTAgenda;
114
115   LexWurzel,
116   LexAktuell,
117   LexEintrag  : PTLexElem;
118   Lexikon     : Text;
119
120
121
122  FUNCTION NimmNummer:INTEGER;
123   BEGIN
124    Nummer := Nummer + c1;
125    NimmNummer := Nummer
126   END;
127
128
129
130
131  PROCEDURE LiesDasLexikon (VAR f:Text;
132                            G:TGrammatik;
133                            l:PTLexElem);
134    VAR
135     zaehler :INTEGER;
```

```
136       z11      : 1..c11;
137       z4       : 1.. c4;
138       ch       :   CHAR;
139       st5      : STRING(.c5.);
140
141     BEGIN
142      ASSIGN(f,datei);
143      LexWurzel := NIL;
144      RESET(f);
145      WHILE NOT EOF(f)
146       DO
147        BEGIN
148         NEW(LexEintrag);
149         IF LexWurzel = NIL
150          THEN
151           BEGIN
152            LexWurzel := LexEintrag;
153            LexAktuell:= LexWurzel;
154            LexEintragˆ.naechstes := NIL;
155           END
156          ELSE
157           BEGIN
158            LexAktuellˆ.naechstes := LexEintrag;
159            LexEIntragˆ.naechstes := NIL;
160            LexAktuell            := LexAktuellˆ.
                  naechstes;
161           END;
162         LexEintragˆ.Terminal := '';
163         st5 := '';
164         FOR Zaehler := c1 to c5
165          DO
166           BEGIN
167            READ(f,ch);
168            st5 := st5 + UPCASE(ch)
169           END;
170         REPEAT
171          READ(f,ch);
172          LexEintragˆ.terminal := LexEintragˆ.Terminal +
                UPCASE(ch);
173         UNTIL EOLN(f);
174         READLN(f);
175         IF st5 = 'KBG**' THEN  LexEintragˆ.Kategorie :=
               KBG    ELSE
176         IF st5 = 'VBG**' THEN  LexEintragˆ.Kategorie :=
               VBG    ELSE
177         IF st5 = 'KBBD*' THEN  LexEintragˆ.Kategorie :=
```

```
            KBBD   ELSE
178       IF st5 = 'VBBD*' THEN  LexEintrag^.Kategorie :=
            VBBD   ELSE
179       IF st5 = 'KBA**' THEN  LexEintrag^.Kategorie :=
            KBA    ELSE
180       IF st5 = 'VBA**' THEN  LexEintrag^.Kategorie :=
            VBA    ELSE
181       IF st5 = 'KAE**' THEN  LexEintrag^.Kategorie :=
            KAE    ELSE
182       IF st5 = 'VAE**' THEN  LexEintrag^.Kategorie :=
            VAE    ELSE
183       IF st5 = 'KAA**' THEN  LexEintrag^.Kategorie :=
            KAA    ELSE
184       IF st5 = 'VAA**' THEN  LexEintrag^.Kategorie :=
            VAA    ELSE
185       IF st5 = 'KAV**' THEN  LexEintrag^.Kategorie :=
            KAV    ELSE
186       IF st5 = 'VAV**' THEN  LexEintrag^.Kategorie :=
            VAV
187      END;
188    END;
189
190
191   PROCEDURE LiesDenSatz;
192    VAR
193     satz:         STRING(.cmax.);
194     zaehler:      INTEGER;
195    BEGIN
196     CLRSCR;
197     WRITELN(CopyRight);
198     WRITE('----->␣');
199     Wurzel.spalte := NIL;
200     Wurzel.zeigt  := NIL;
201     READLN(satz);
202     FOR zaehler := c1 to LENGTH(satz)
203      DO satz(.zaehler.) := UPCASE(satz(.zaehler.));
204     Satz := Satz + blank;
205     Writeln('----->␣',satz);
206     WHILE satz <> ''
207     DO
208     BEGIN
209        NEW(Pneu);
210        Pneu^.nummer   :=NimmNummer;
211        Pneu^.wort     := TRUE;
212        NEW(Pneu^.gefunden);
213        Pneu^.gefunden^.kante := Pneu;
```

```
214         pneu^.gefunden^.next  := NIL;
215         Pneu^.gesucht         := NIL;
216         Pneu^.nachkomme       :=FALSE;
217         IF Wurzel.zeigt = NIL
218          THEN
219           BEGIN
220             Wurzel.zeigt := pneu;
221             Wurzel.spalte:= pneu;
222             PZiel.spalte := pneu;
223             PZiel.zeigt  := Pneu;
224             pneu^.vor    := NIL;
225             Pneu^.zeigt  := NIL;
226             Pneu^.nach   := NIL;
227           END
228          ELSE
229           BEGIN
230            Wurzel.zeigt^.zeigt := Pneu;
231            Pneu^.vor           := Wurzel.zeigt;
232            Pneu^.nach          := NIL;
233            Pneu^.zeigt         := NIL;
234            Wurzel.zeigt        := Wurzel.zeigt^.zeigt;
235           END;
236         pneu^.aktiv   := false;
237         pneu^.inhalt  := COPY(satz,c1,POS(blank,satz)-
               c1);
238         LexAktuell    := LexWurzel;
239         WHILE LexAktuell <> NIL
240          DO
241           BEGIN
242            IF LexAktuell^.Terminal = pneu^.inhalt
243             Then
244              BEGIN
245               pneu^.Kategorie := LexAktuell^.Kategorie;
246              END;
247            LexAktuell := LexAktuell^.naechstes;
248           END;
249         DELETE(satz,c1,POS(blank,satz));
250       END;
251    END;
252
253
254
255
256
257   PROCEDURE Regel3KanteInAgendaEintragen (Kante:
         PTKante);
```

```
258   VAR
259    Wurzel,
260    PZiel  :TWurzel;
261   PROCEDURE NeuesAgendaPaarAnlegen;
262    BEGIN
263     NEW(paar);
264     IF Agenda = NIL
265      THEN
266       BEGIN
267        Agenda := Paar;
268        Pagenda:= Paar;
269        Paar^.next := NIL;
270        Paar^.back := NIL;
271       END
272      ELSE
273       BEGIN
274        PAgenda^.next := Paar;
275        Paar^.next    := NIL;
276        Paar^.back    := Pagenda;
277        Pagenda       := Pagenda^.next;
278      END;
279    END;
280
281   BEGIN
282    IF Kante^.aktiv
283     THEN
284      BEGIN
285       Wurzel.zeigt := Kante^.zeigt;
286       WHILE wurzel.zeigt <> NIL
287        DO
288        BEGIN
289         IF NOT(wurzel.zeigt^.aktiv)
290          THEN
291           BEGIN
292            NeuesAgendaPaarAnlegen;
293            paar^.A := kante;
294            paar^.I := wurzel.zeigt;
295           END;
296        Wurzel.zeigt  := Wurzel.zeigt^.nach
297        END
298      END
299     ELSE
300     BEGIN
301       PZiel.zeigt  := Kante;
302       WHILE NOT(PZiel.zeigt^.Wort)
303        DO PZiel.Zeigt := PZiel.Zeigt^.Vor;
```

```
304         Wurzel.Zeigt    := PZiel.Zeigt;
305         Wurzel.Spalte   := PZiel.Zeigt;
306         PZiel.Spalte    := Pziel.zeigt;
307         WHILE wurzel.spalte <> NIL
308          DO
309          BEGIN
310           WHILE wurzel.zeigt <> NIL
311           DO
312           BEGIN
313            IF wurzel.zeigt^.aktiv
314             AND (Wurzel.zeigt^.zeigt = PZiel.spalte)
315             THEN
316              BEGIN
317               NeuesAGendaPaarAnlegen;
318               paar^.I := kante;
319               paar^.A := wurzel.zeigt;
320              END;
321             Wurzel.zeigt  := Wurzel.zeigt^.nach
322           END;
323           wurzel.spalte  := wurzel.spalte^.vor;
324           wurzel.zeigt   := wurzel.spalte;
325          END
326         END
327        END;
328
329
330    PROCEDURE NimmAgendaEintrag(VAR PEintrag:PTAgenda);
331     BEGIN
332        IF PAgenda = Agenda
333        THEN
334         BEGIN
335          PEintrag := Agenda;
336          PAgenda  := NIL;
337          Agenda   := NIL;
338         END
339        ELSE
340         BEGIN
341          PAGENDA       := PAGENDA^.back;
342          PEintrag      := PAgenda^.next;
343          PAGENDA^.next := NIL;
344         END;
345     END;
346
347
348
349
```

```
350
351   PROCEDURE Regel2EineNeueKanteAnlegen( Kante     :
          PTKante;
352                                         Kategorie :
                                               TKategorien
                                               ;
353                                         Gram      :
                                               TGrammatik
                                               );
354    VAR
355      Wurzel             :TWurzel;
356      PHilfe,
357      PGesuchteKategorie :PTKategorienListe;
358      zaehler,
359      zaehler2           :INTEGER;
360
361    BEGIN
362    Wurzel.zeigt := Kante;
363    Wurzel.spalte:= Kante;
364    WHILE Wurzel.zeigt^.nach <> NIL
365     DO Wurzel.zeigt := Wurzel.zeigt^.nach;
366     FOR zaehler := c1 To c11
367      DO
368       IF  (kategorie = Gram(.zaehler,c1.))
369       AND (kategorie <> Leer)
370        THEN
371        BEGIN
372         Gram(.zaehler,c1.) := Leer;
373         NEW(pneu);
374         Wurzel.zeigt^.nach := pneu;
375         pneu^.nummer       := NimmNummer;
376         pneu^.vor          := Wurzel.zeigt;
377         Pneu^.nach         := NIL;
378         Pneu^.zeigt        := wurzel.spalte;
379         Wurzel.zeigt       := Wurzel.zeigt^.nach;
380         pneu^.aktiv        := true;
381         pneu^.kategorie    := kategorie;
382         Pneu^.Wort         := false;
383         Pneu^.gesucht      := NIL;
384         Pneu^.gefunden     := NIL;
385         Pneu^.nachkomme    := FALSE;
386         FOR zaehler2 := c2 TO c4
387          DO
388          BEGIN
389           IF Gram(.zaehler,zaehler2.) <> Leer
390            THEN
```

```
391             BEGIN
392              NEW(PGesuchteKategorie);
393              PGesuchteKategorie^.weiter:= NIL;
394              PGesuchteKategorie^.Kategorie := Gram(.
                    zaehler,zaehler2.);
395              IF Pneu^.gesucht = NIL
396               THEN
397                BEGIN
398                 PHilfe         := PGesuchteKategorie;
399                 Pneu^.gesucht  := PHilfe;
400                END
401               ELSE
402                BEGIN
403                 PHilfe^.weiter := PGesuchteKategorie;
404                 PHilfe         := PHilfe^.weiter;
405                END
406             END
407          END;
408         Regel3KanteInAgendaEintragen (pneu);
409         Regel2EineNeueKanteAnlegen(Wurzel.spalte,
410                                    pneu^.gesucht^.
                                       kategorie,gram);
411       END;
412    END;
413
414
415
416
417    PROCEDURE Regel1EineKanteErweitern(paar:PTAgenda);
418     VAR
419      PneuHilf,Pneugefneu,AHilf :PTKantenListe;
420     BEGIN
421
422     IF paar^.I^.kategorie = paar^.A^.gesucht^.kategorie
423      THEN
424       BEGIN
425        NEW(pneu);
426        pneu^.nummer      := NimmNummer;
427        pneu^.kategorie   := Paar^.A^.kategorie;
428
429        Pneu^.gefunden := NIL;
430        AHilf := Paar^.A^.gefunden;
431
432        WHILE AHilf <> NIL
433         DO
434         BEGIN
```

```
435           NEW(Pneugefneu);
436           IF Pneu^.gefunden = NIL
437            THEN
438             BEGIN
439              Pneu^.gefunden := Pneugefneu;
440              PneuHilf       := Pneu^.gefunden;
441              PneuHilf^.next := NIL;
442             END
443            ELSE
444             BEGIN
445              PneuHilf^.next   := Pneugefneu;
446              PneuHilf         := PneuHilf^.next;
447              PneuHilf^.next   := NIL;
448             END;
449
450           Pneugefneu^.kante     := AHilf^.kante;
451           AHilf                 := AHilf^.next;
452         END;
453
454         NEW(Pneugefneu);
455         IF Pneu^.gefunden = NIL
456          THEN
457           BEGIN
458            Pneu^.gefunden := Pneugefneu;
459            Pneugefneu^.next := NIL;
460           END
461          ELSE
462           BEGIN
463             PneuHilf^.next   := Pneugefneu;
464             PneuHilf         := PneuHilf^.next;
465             PneuHilf^.next   := NIL;
466           END;
467         Pneugefneu^.kante    := Paar^.I;
468
469         Pneu^.wort             := FALSE;
470         IF Paar^.A^.gesucht^.weiter = NIL
471          THEN Pneu^.gesucht   := NIL
472          ELSE Pneu^.gesucht   := Paar^.A^.gesucht^.
                weiter;
473         Pneu^.nachkomme := TRUE;
474
475        IF pneu^.gesucht   = NIL
476         THEN Pneu^.aktiv := false
477         ELSE Pneu^.aktiv := true;
478
479        WHILE Paar^.A^.nach <> NIL
```

```
480         DO Paar^.A        := Paar^.A^.nach;
481
482        Paar^.A^.nach      := pneu;
483        pneu^.vor          := Paar^.A;
484        pneu^.zeigt        := Paar^.I^.zeigt;
485        pneu^.nach         := NIL;
486
487        Regel3KanteInAgendaEintragen (pneu);
488        IF Pneu^.aktiv
489         THEN Regel2EineNeueKanteAnlegen(Pneu^.zeigt,
490                                         pneu^.gesucht^.
                                               kategorie,
                                               Grammatik);
491       END;
492
493
494     END;
495
496     PROCEDURE SatzAnalyse;
497      BEGIN
498      WHILE Agenda <> NIL
499      DO
500       BEGIN
501        NimmAgendaEintrag(Paar);
502        Regel1EineKanteErweitern(Paar);
503       END;
504
505      END;
506
507     PROCEDURE GibAlleSatzalternativenAus;
508      CONST
509       BlankAnz:INTEGER = c2;
510      VAR
511       PHilf    :PTkantenListe;
512
513      PROCEDURE SatzAusgabe(Kante:PTKante;BlankAnz:
            INTEGER);
514       VAR
515
516       Zaehler:INTEGER;
517       PHilf  :PTKantenListe;
518       BEGIN
519        FOR Zaehler := c1 TO BlankAnz DO WRITE(blank);
520
521        IF Kante^.kategorie = VKG     THEN WRITELN ('VKG
              ␣') ELSE
```

```
522       IF Kante^.kategorie = BG      THEN WRITELN ('BG␣
              ␣') ELSE
523       IF Kante^.kategorie = VT      THEN WRITELN ('VT␣
              ␣') ELSE
524       IF Kante^.kategorie = AV      THEN WRITE   ('AV␣
              ␣') ELSE
525       IF Kante^.kategorie = B       THEN WRITELN ('B␣␣␣
              ␣') ELSE
526       IF Kante^.kategorie = A       THEN WRITE   ('A␣␣␣
              ␣') ELSE
527       IF Kante^.kategorie = BBD     THEN WRITE   ('BBD
              ␣') ELSE
528       IF Kante^.kategorie = BA      THEN WRITELN ('BA␣
              ␣') ELSE
529       IF Kante^.kategorie = AE      THEN WRITE   ('AE␣
              ␣') ELSE
530       IF Kante^.kategorie = AA      THEN WRITE   ('AA␣
              ␣') ELSE
531       IF Kante^.kategorie = KBG     THEN WRITELN ('KBG
              ␣') ELSE
532       IF Kante^.kategorie = VBG     THEN WRITELN ('VBG
              ␣') ELSE
533       IF Kante^.kategorie = KBBD    THEN WRITELN ('
              KBBD') ELSE
534       IF Kante^.kategorie = VBBD    THEN WRITE   ('
              VBBD') ELSE
535       IF Kante^.kategorie = KBA     THEN WRITELN ('KBA
              ␣') ELSE
536       IF Kante^.kategorie = VBA     THEN WRITE   ('VBA
              ␣') ELSE
537       IF Kante^.kategorie = KAE     THEN WRITE   ('KAE
              ␣') ELSE
538       IF Kante^.kategorie = VAE     THEN WRITELN ('VAE
              ␣') ELSE
539       IF Kante^.kategorie = KAA     THEN WRITE   ('KAA
              ␣') ELSE
540       IF Kante^.kategorie = VAA     THEN WRITE   ('VAA
              ␣') ELSE
541       IF Kante^.kategorie = KAV     THEN WRITE   ('KAV
              ␣') ELSE
542       IF Kante^.kategorie = VAV     THEN WRITE   ('VAV
              ␣');
543
544       IF Kante^.wort
545        THEN
546         WRITELN('---->␣',Kante^.inhalt)
```

```
547         ELSE
548          BEGIN
549          PHilf := Kante^.gefunden;
550          WHILE PHilf <> NIL
551           DO
552            BEGIN
553             Satzausgabe(PHilf^.kante,Blankanz+c1);
554             PHilf := Philf^.next;
555            END
556          END
557      END;
558
559      BEGIN
560        WHILE Wurzel.zeigt^.vor <> NIL
561         DO Wurzel.zeigt := Wurzel.zeigt^.vor;
562
563        WHILE Wurzel.zeigt <> NIL
564        DO
565        BEGIN
566         IF (Wurzel.zeigt^.kategorie = VKG)
567           AND ((NOT(Wurzel.zeigt^.aktiv))
568           AND (wurzel.zeigt^.zeigt = NIL))
569           THEN
570            BEGIN
571             WRITELN('VKG');
572             PHilf := Wurzel.zeigt^.gefunden;
573             WHILE PHilf <> NIL
574              DO
575               BEGIN
576                Satzausgabe(PHilf^.kante,Blankanz+c1);
577                PHilf := Philf^.next;
578               END
579            END;
580        Wurzel.zeigt := Wurzel.zeigt^.nach;
581        END;
582
583      END;
584
585    PROCEDURE LoescheDieListe;
586     PROCEDURE LoescheWort(kante :PTKante);
587      PROCEDURE LoescheSpalte(kante:PTKante);
588       VAR
589        Pgefunden :PTKantenListe;
590        Pgesucht  :PTKategorienListe;
591       PROCEDURE LoescheGesucht(p:PTKategorienListe);
592        BEGIN
```

```
593         IF p^.weiter <> NIL
594          THEN LoescheGesucht(p^.weiter);
595         IF P <> NIL THEN DISPOSE(P);
596        END;
597       PROCEDURE LoescheGefunden(Kante:PTKante;p:
              PTKantenListe);
598        BEGIN
599         IF p^.next <> NIL
600          THEN LoescheGefunden(Kante,p^.next);
601         DISPOSE(P);
602        END;
603       BEGIN(*LoescheSpalte*)
604        IF Kante^.nach <> NIL
605         THEN LoescheSpalte(kante^.nach);
606        IF (NOT Kante^.nachkomme) AND ((Kante^.gesucht
              <> NIL)
607         AND (NOT Kante^.wort))
608         THEN LoescheGesucht(Kante^.gesucht);
609        IF Kante^.gefunden <> NIL
610         THEN LoescheGefunden(Kante,Kante^.gefunden);
611        DISPOSE(Kante)
612       END;(*LoescheSpalte*)
613      BEGIN(*LoescheWort*)
614       IF Kante^.zeigt <> NIL
615        THEN LoescheWort(Kante^.zeigt);
616      LoescheSpalte(Kante);
617      END;(*LoescheWort*)
618     BEGIN(*LoescheDieListe*)
619      WHILE Wurzel.spalte^.vor <> NIL
620       DO Wurzel.spalte := Wurzel.spalte^.vor;
621      LoescheWort(Wurzel.spalte);
622     END;(*LoescheDieListe*)
623
624    BEGIN
625     Agenda := NIL;
626     PAgenda := Agenda;
627     LiesDasLexikon(Lexikon,Grammatik,LexWurzel);
628     LiesDenSatz;
629     WHILE Wurzel.spalte^.vor <> NIL
630      DO Wurzel.spalte := Wurzel.spalte^.vor;
631     Regel2EineNeueKanteAnlegen(Wurzel.spalte,VKG,
           Grammatik);
632     SatzAnalyse;
633     GibAlleSatzalternativenAus;
634     LoescheDieListe;
635
```

```
636
637   END.
```

```
Demo-Parser Chart-Parser Version 1.0(c)1992 by Paul Koop -
- - - - > KBG VBG KBBD KBA VBA KAE VAE KAA VAA
KAV VAV - - - - - > KBG VBG KBBD KBA VBA KAE VAE
KAA VAA KAV VAV VKG BG KBG - - - - > KBG VBG - - -
- > VBG VT B BBD KBBD - - - - > KBBD VBBD - - - - >
VBBD BA KBA - - - - >. KBA VBA - - - - > VBA A AE KAE
- - - - > KAE VAE - - - - > VAE AA KAA - - - - > KAA VAA
- - - - > VAA AV KAV - - - - > KAV VAV - - - - > VAV
Demo-Parser Chart-Parser Version 1.0(c)1992 by Paul
Koop - - - - - - > KBG VBG KBBD KBA VBA KAE VAE KAA
VAA KAV VAV - - - - - - > KBG VBG KBBD KBA VBA KAE VAE
KAA VAA KAV VAV VKG BG KBG - - - - - > KBG VBG - - -
- > VBG VT B BBD KBBD - - - - - > KBBD VBBD - - - - - >
VBBD BA KBA - - - - - >.  KBA VBA - - - - - > VBA A AE
KAE - - - - - > KAE VAE - - - - - > VAE AA KAA - - - - - >
KAA VAA - - - - - > VAA AV KAV - - - - - > KAV VAV - - -
- - > VAV
```

Figure 4: ASCII-Output des Konsolenprogramms

```
1  import re
2
3  # Lesen des Korpus aus einer Datei
4  #with open("VKGKORPUS.TXT", "r") as f:
5  #    korpus = f.read()
6  korpus = "KBG␣VBG␣KBBD␣VBBD␣KBA␣VBA␣KAE␣VAE␣KAA␣VAA␣
       KBBD␣VBBD␣KBA␣VBA␣KBBD␣VBBD␣KBA␣VBA␣KBBD␣VBBD␣KBA␣
       VBA␣KAE␣VAE␣KAA␣VAA␣KAV␣VAV"
7  # Extrahieren der Terminalsymbole aus dem Korpus
8  terminals = re.findall(r"[KV][A-Z]+", korpus)
9
10 # Entfernen der vorangestellten K- oder V-Zeichen aus
       den Terminalsymbolen
11 non_terminals = list(set([t[1:] for t in terminals]))
12
13 # Erzeugen der Regelproduktionen
14 productions = []
15 for nt in non_terminals:
16     rhs = [t for t in terminals if t[1:] == nt]
17     productions.append((nt, rhs))
18
```

```
# Ausgabe der Grammatikregeln
print("Regeln:")
for nt, rhs in productions:
    print(nt + " -> " + " | ".join(rhs))

# Ausgabe der Startsymbol
print("Startsymbol: VKG")
```

```
 Regeln:  AV -> KAV | VAV BG -> KBG | VBG AA -> KAA
| VAA | KAA | VAA AE -> KAE | VAE | KAE | VAE BA ->
KBA | VBA | KBA | VBA | KBA | VBA | KBA | VBA BBD
-> KBBD | VBBD | KBBD | VBBD | KBBD | VBBD | KBBD |
VBBD Startsymbol:  VKG
```

Figure 5: ASCII-Output des Konsolenprogramms

Aus dem Korpus läßt sich auch eine probabilistische kontextfreie Grammatik mit gewichteten Produktionen induzieren:

```
from collections import defaultdict
import random

# define the grammar production rules
grammar = defaultdict(list)

# read in the corpus
corpus = "KBG VBG KBBD VBBD KBA VBA KAE VAE KAA VAA
    KBBD VBBD KBA VBA KBBD VBBD KBA VBA KBBD VBBD KBA
    VBA KAE VAE KAA VAA KAV VAV".split()

# get the non-terminal symbols
nonterminals = set([symbol[1:] for symbol in corpus if
     symbol.startswith("K") or symbol.startswith("V")])

# iterate over the corpus and count the production
    rules
for i in range(1, len(corpus)):
    curr_symbol = corpus[i]
    prev_symbol = corpus[i-1]
    if prev_symbol.startswith("K") or prev_symbol.
        startswith("V"):
         grammar[prev_symbol[1:]].append(curr_symbol)

# calculate the probabilities for the production rules
for lhs in grammar.keys():
    productions = grammar[lhs]
```

```
23      total_count = len(productions)
24      probabilities = defaultdict(float)
25      for rhs in productions:
26          probabilities[rhs] += 1.0
27      for rhs in probabilities.keys():
28          probabilities[rhs] /= total_count
29      grammar[lhs] = probabilities
30
31  # print the grammar
32  print("Grammar:")
33  for lhs in grammar.keys():
34      print(lhs + "␣->")
35      for rhs in grammar[lhs].keys():
36          print("␣␣" + rhs + "␣:␣" + str(grammar[lhs][
                rhs]))
```

```
 Grammar:  BG -> VBG : 0.5 KBBD : 0.5 BBD -> VBBD
: 0.5 KBA : 0.5 BA -> VBA : 0.5 KAE : 0.25 KBBD :
0.25 AE -> VAE : 0.5 KAA : 0.5 AA -> VAA : 0.5 KBBD
: 0.25 KAV : 0.25 AV -> VAV : 1.0
```

Figure 6: ASCII-Output des Konsolenprogramms

Eine probabilistische Grammatik kann als Bayessches Netz interpretiert werden. In einem Bayesschen Netz werden die Abhängigkeiten zwischen den Variablen durch gerichtete Kanten modelliert, während die Wahrscheinlichkeiten der einzelnen Variablen und Kanten durch Wahrscheinlichkeitsverteilungen dargestellt werden. In einer probabilistischen Grammatik werden die Produktionsregeln als Variablen und die Terme und Nichtterminale als Zustände modelliert. Jede Produktion hat eine bestimmte Wahrscheinlichkeit, die durch eine Wahrscheinlichkeitsverteilung dargestellt werden kann. Die Wahrscheinlichkeit, einen bestimmten Satz zu generieren, kann dann durch die Produktionsregeln und deren Wahrscheinlichkeiten berechnet werden. Die Zustände in der probabilistischen Grammatik können als Knoten im Bayesschen Netz interpretiert werden, während die Produktionsregeln als gerichtete Kanten dargestellt werden können. Die Wahrscheinlichkeiten der Produktionsregeln können dann als Kantenbedingungen modelliert werden. Durch die Berechnung der posterior Wahrscheinlichkeit kann dann eine probabilistische Vorhersage getroffen werden, welcher Satz am wahrscheinlichsten ist, gegeben die Beobachtungen. Das Korpus kann als Protokoll der wechselseitigen Interaktion zweier Softwareagenten eines Multiagentensystems verstanden werden. Die Agenten dieses Multiagentensystems haben Zugriff auf das letzte generierte Terminalzeichen und die probabilistische Grammatik, die als Bayerisches Netz interpretiert werden kann. Dieses Wissen nutzen sie zur Generierung des nächsten Terminalzeichens. Ein Agent K generiert die Käufer-Terminalzeichen. Ein Agent V generiert die Verkäufer-Terminalzeichen.

Das Korpus kann als Protokoll der wechselseitigen Interaktion zweier Softwareagenten eines Multiagentensystems verstanden werden. Die Agenten dieses Multiagentensystems haben Zugriff auf das letzte generierte Terminalzeichen und die probabilistische Grammatik, die als Bayerisches Netz interpretiert werden kann. Dieses Wissen nutzen sie zur Generierung des nächsten Terminalzeichens. Ein Agent K generiert die Käufer-Terminalzeichen. Ein Agent V generiert die Verkäufer-Terminalzeichen. Das Korpus kann als Protokoll der wechselseitigen Interaktion zweier Softwareagenten eines Multiagentensystems verstanden werden. Die Agenten dieses Multiagentensystems haben Zugriff auf das letzte generierte Terminalzeichen und die probabilistische Grammatik, die als Bayerisches Netz interpretiert werden kann. Dieses Wissen nutzen sie zur Generierung des nächsten Terminalzeichens. Ein Agent K generiert die Käufer-Terminalzeichen. Ein Agent V generiert die Verkäufer-Terminalzeichen.

```
1  import random
2
3  # Die gegebene probabilistische Grammatik
4  grammar = {
5      'BG': {'VBG': 0.5, 'KBBD': 0.5},
6      'BBD': {'VBBD': 0.5, 'KBA': 0.5},
7      'BA': {'VBA': 0.5, 'KAE': 0.25, 'KBBD': 0.25},
8      'AE': {'VAE': 0.5, 'KAA': 0.5},
9      'AA': {'VAA': 0.5, 'KAV': 0.25, 'KBBD': 0.25},
10     'AV': {'VAV': 1.0},
11 }
12
13 # Zuf llige Belegung von Ware und Zahlungsmittel bei
       den Agenten
14 agent_k_ware = random.uniform(0, 100)
15 agent_k_zahlungsmittel = 100 - agent_k_ware
16 agent_v_ware = random.uniform(0, 100)
17 agent_v_zahlungsmittel = 100 - agent_v_ware
18
19 # Entscheidung  ber  die Rollenverteilung basierend
       auf Ware und Zahlungsmittel
20 if agent_k_ware > agent_v_ware:
21     agent_k_role = 'K ufer'
22     agent_v_role = 'Verk ufer'
23 else:
24     agent_k_role = 'Verk ufer'
25     agent_v_role = 'K ufer'
26
27 # Ausgabe der Rollenverteilung und der Belegung von
       Ware und Zahlungsmittel
28 print("Agent␣K:␣Rolle␣=", agent_k_role, "|␣Ware␣=",
       agent_k_ware, "|␣Zahlungsmittel␣=",
```

```
    agent_k_zahlungsmittel)
29  print("Agent␣V:␣Rolle␣=", agent_v_role, "|␣Ware␣=",
        agent_v_ware, "|␣Zahlungsmittel␣=",
        agent_v_zahlungsmittel)
30  print()
31
32  # Agent K startet den Dialog mit dem Terminalzeichen '
        KBG'
33  last_terminal = 'KBG'
34
35  # Maximale Anzahl von Terminalzeichen im Dialog
36  max_terminals = 10
37
38  # Dialog-Schleife
39  for i in range(max_terminals):
40      # Agent K generiert das n chste Terminalzeichen
            basierend auf der Grammatik und dem letzten
            Terminalzeichen
41      next_terminal = random.choices(list(grammar[
            last_terminal].keys()), weights=list(grammar[
            last_terminal].values()))[0]
42
43      # Agent V generiert das n chste Terminalzeichen
            basierend auf der Grammatik und dem letzten
            Terminalzeichen
44      next_terminal = random.choices(list(grammar[
            last_terminal].keys()), weights=list(grammar[
            last_terminal].values()))[0]
45
46      # Aktualisierung des letzten Terminalzeichens
47      last_terminal = next_terminal
48
49      # Ausgabe des aktuellen Terminalzeichens
50      print("Agent␣K:", next_terminal)
51
52      # Break, wenn das Terminalzeichen 'VAV' erreicht
            ist
53      if next_terminal == 'VAV':
54          break
```

Agent K: KBBD Agent V: VBBD Agent K: KBA Agent V: VAE Agent K: KBBD Agent V: VBBD Agent K: KBA Agent V: VBBD Agent K: KBA Agent V: VAE Agent K: KAA Agent V: VAA Agent K: KBBD Agent V: VBBD Agent K: KBA Agent V: VAE Agent K: KAA Agent V: VAA Agent K: KAA Agent V: VAA Agent K: KAA Agent V: VAA Agent K: KAV Agent V: VAV Agent K: Rolle = Verkäufer | Ware = 60.935380690830155 | Zahlungsmittel = 39.064619309169845 Agent V: Rolle = Käufer | Ware = 46.51117771417693 | Zahlungsmittel = 53.48882228582307
Agent K: KBBD Agent V: VBBD Agent K: KBA Agent V: VAE Agent K: KBBD Agent V: VBBD Agent K: KBA Agent V: VBBD Agent K: KBA Agent V: VAE Agent K: KAA Agent V: VAA Agent K: KBBD Agent V: VBBD Agent K: KBA Agent V: VAE Agent K: KAA Agent V: VAA Agent K: KAA Agent V: VAA Agent K: KAA Agent V: VAA Agent K: KAV Agent V: VAV

Figure 7: ASCII-Output des Konsolenprogramms

Literatur

- Alpaydin, E.: *Maschinelles Lernen*, 2008
- Chomsky, N.: *Aspects of the Theory of Syntax*, 1965
- Dehmer, Matthias: *Strukturelle Analyse Web-basierter Dokumente*, 2005
- Diekmann, A.: *Spieltheorie: Einführung, Beispiele, Experimente*, 2009
- Gold, E. Mark: *Limiting Recursion, The Journal of Symbolic Logic 30: 28–48*, 1965
- Gold, E. Mark: *Language Identification in the Limit, Information and Control 10: 447–474*, 1967
- Koop, P.: *Über die Entscheidbarkeit der GTG*, 1994
- Koop, P.: *Rekursive Strukturen und Prozesse*, 1995
- Koop, P.: *K-Systeme: Das Projekt ARS*, 1994
- Koop, P.: *Algorithmisch Rekursive Sequenzanalyse*, 1996
- Koop, P.: *Oevermann, Chomsky, Searle*, 1994
- Koop, P.: https://github.com/pkoopongithub/ algorithmisch-rekursive-sequenzanalyse/)

- Krauße, C. C., & Krueger, F. R.: *Unbekannte Signale, Spektrum Dossier 2/2002*
- Krempel, Rasmus: *Netze, Karten, Irrgärten: Graphenbasierte explorative Ansätze zur Datenanalyse und Anwendungsentwicklung in den Geisteswissenschaften*, 2016
- Lisch, R., Kriz, J.: *Grundlagen und Modelle der Inhaltsanalyse*, 1978
- Mayring, P.: *Einführung in die qualitative Sozialforschung*, 1990
- Ndiaye, Alassane: *Rollenübernahme als Benutzermodellierungsmethode: globale Antizipation in einem transmutierbaren Dialogsystem*, 1998
- Nevill-Manning Witten: *Identifying Hierarchical Structure in Sequences: A linear-time algorithm*, 1999
- Oevermann, U.: *Die objektive Hermeneutik als unverzichtbare methodologische Grundlage für die Analyse von Subjektivität. Zugleich eine Kritik an der Tiefenhermeneutik, in: Jung, Th., Müller-Dohm, St. (Hg): »Wirklichkeit« im Deutungsprozess: Verstehen und Methoden in den Kultur- und Sozialwissenschaften*, Frankfurt 1993
- Shen, Chunze: *EDSI - Effiziente Grammatikinduktion*, 2013