

Soziale Strukturen und Prozesse Kausale Inferenz mit Probabilistischen kontextfreien Grammatiken und Bayesschen Netzen Mit Multiagentensystem und Entscheidungsbaum für Rolle Agenten im Dialog nach empirisch gesicherter Handlungsgrammatik
Algorithmisch Rekursive Sequenzanalyse

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 Verkaufsgesprächs auf einem Wochenmarkt
https://github.com/pkoopongithub/algorithmisch-rekursive-sequenzanalyse/blob/main/Aachen_280694_11Uhr.mp3). Die Sequenzanalyse des transkribierten Protokolls (<https://github.com/pkoopongithub/algorithmisch-rekursive-sequenzanalyse/blob/main/oechsle.pdf>) und die Kodierung mit den generierten Kategorien (<https://github.com/pkoopongithub/algorithmisch-rekursive-sequenzanalyse/blob/main/fallstruktur.pdf>) ist dort auch abgelegt.

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; Paul Koop M.A. GRAMMATIKINDUKTION empirisch                                  ;;
;; gesicherter Verkaufsgespraechе                                               ;;
;;                                                                              ;;
;; Die Simulation wurde ursprünglich entwickelt,                                ;;
;; um die Verwendbarkeit von kontextfreien Grammatiken                          ;;
;; fuer die Algorithmisch Rekursive Sequanzanalyse                              ;;
;; zu ueberpruefen                                                              ;;
;; Modellcharakter hat allein der Quelltext.                                  ;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;;      |_      |_      |_      |_      |_      |_      |_      ;;
;;      |_      |_      |_      |_      |_      |_      |_      ;;
;; KBG->VBGKBBd->VBBdKBA->VBAKAE->VAEKAA->VAAKAV-> VAV                ;;
;;                                                                              ;;
;; Die Produktionen --> sind entsprechend ihrer                               ;;
;; emp. Auftrittswahrscheinlichkeit gewichtet                                  ;;
;; DIE GRAMMATIK WIRD AUS DEM KORPUS INDUZIERT                                ;;
;; ein Left-to-the-Right-Modell                                               ;;
;;                                                                              ;;

;;                                                                              ;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;; Transformationsmatrix                                                            ;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;a   b c   d e   f c   d   e   f   g   h   i   j   g   h
    i   j   k   l ;;
```

```

;;0  1 2  3 4  5 2  3  4  5  6  7  8  9  6  7
    8  9  10 11;;
;;
;;  0  1  2  3  4  5  6  7  8  9  10 11
    12 13  ;;
;;0  -  1
    ;;
;;1  -  2
    ;;
;;2  -  -  2
    ;;
;;3  -  -  2
    ;;
;;4  -  -  -  -  2
    ;;
;;5  -  1  2
    ;;
;;6  -  -  2
    ;;
;;7  -  -  2
    ;;
;;8  -  -  -  -  2
    ;;
;;9  -  1  1
    ;;
;;10 -  -  -  -  -  -  -  -  -  -  1
    ;;
;;11
    ;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;
;; Begruessung      := BG
;; Bedarf           := Bd
;; Bedarfsargumentation := BA
;; Abschlusseinwaende := AE
;; Verkaufsabschluss := AA
;; Verabschiedung   := AV
;; Kunde            := vorangestelltes K
;; Verkaeufer       := vorangestelltes V
;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;; Korpus
(define korpus (list 'KBG 'VBG 'KBBd 'VBBd 'KBA 'VBA 'KBBd 'VBBd
'KBA 'VBA 'KAE 'VAE 'KAE 'VAE 'KAA 'VAA 'KAV 'VAV));; 0 - 17

;; Korpus durchlaufen
(define (lesen korpus)

```

```

;; car ausgeben
(display (car korpus))
;; mit cdr weitermachen
(if(not(null? (cdr korpus)))
  (lesen (cdr korpus))
  ;;(else)
)
)

;; Lexikon
(define lexikon (vector 'KBG 'VBG 'KBBd 'VBBd 'KBA 'VBA 'KAE 'VAE
'KAA 'VAA 'KAV 'VAV)) ;; 0 - 12

;; Index fuer Zeichen ausgeben
(define (izeichen zeichen)
  (define wertzeichen 0)
  (do ((i 0 (+ i 1)))
    ((equal? (vector-ref lexikon i) zeichen))
    (set! wertzeichen (+ 1 i))
  )
  ;;index zurueckgeben
  wertzeichen
)

;; transformationsmatrix
(define zeile0 (vector 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0))
(define zeile1 (vector 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0))
(define zeile2 (vector 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0))
(define zeile3 (vector 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0))
(define zeile4 (vector 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0))
(define zeile5 (vector 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0))
(define zeile6 (vector 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0))
(define zeile7 (vector 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0))
(define zeile8 (vector 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0))
(define zeile9 (vector 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0))
(define zeile10 (vector 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0))
(define zeile11 (vector 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0))
(define zeile12 (vector 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0))
(define zeile13 (vector 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0))
(define zeile14 (vector 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0))
(define zeile15 (vector 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0))
(define zeile16 (vector 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0))
(define zeile17 (vector 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0))

(define matrix (vector zeile0 zeile1 zeile2 zeile3 zeile4 zeile5
zeile6 zeile7 zeile8 zeile9 zeile10 zeile11 zeile12 zeile13 zeile14
zeile15 zeile16 zeile17))

```

```

;; Transformationen zaehlen
;; Korpus durchlaufen
(define (transformationenZaehlen korpus)
  ;; car zaehlen
  (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))))))
  ;; mit cdr weitermachen
  (if(not(null? (cdr (cdr korpus))))
    (transformationenZaehlen (cdr korpus))
    ;;(else)
  )
)

;; Transformation aufaddieren

;; Zeilensummen bilden und Prozentwerte bilden

;; Grammatik
(define grammatik (list '- ))

;; aus matrix regeln bilden und regeln in grammatik eingefuege
(define (grammatikerstellen matrix)
  (do ((a 0 (+ a 1)))
    ((= a 12) )(newline)
    (do ((b 0 (+ b 1)))
      ((= b 12))
      (if (< 0 (vector-ref (vector-ref matrix a) b) )
        (display (cons (vector-ref lexikon a) (cons '-> (vector-ref
lexikon b))))
      )
    )
  )
)
)
)
)

```

Zum Erstellen der Grammatik wird die Transformationstabelle erstellt und aus dieser die Grammatik

```

(transformationenZaehlen korpus)
(grammatikerstellen matrix)

```

Die Grammatik wird dann erstellt

```

(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)

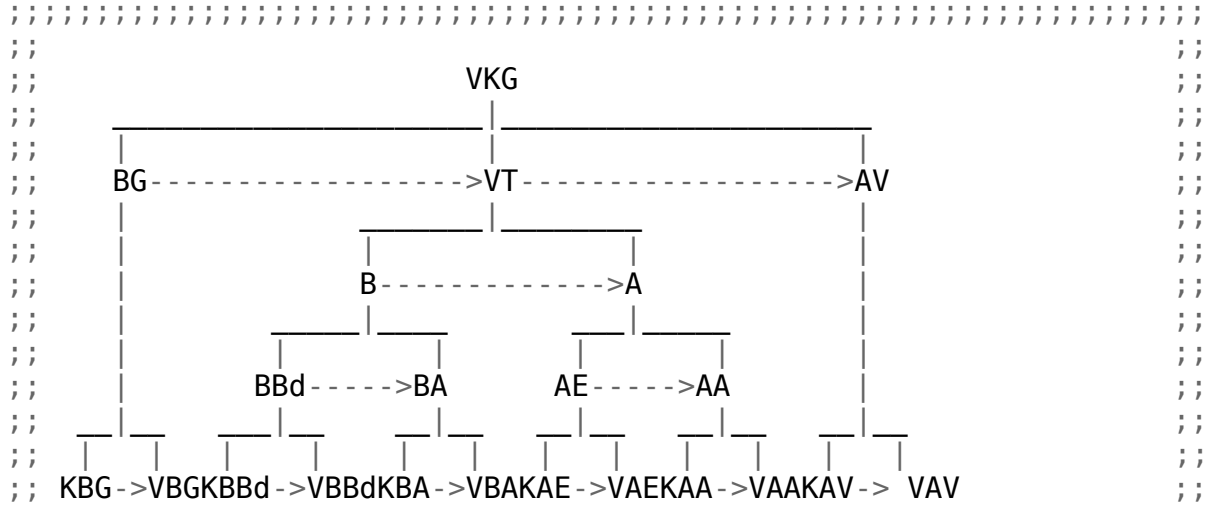
```

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

```

;;;;;
;; Paul Koop M.A. 1994 Sequenzanalyse empirisch ;;
;; gesicherter Verkaufsgespraeche ;;
;; ;;
;; Die Simulation wurde urspruenglich entwickelt, ;;
;; um die Verwendbarkeit von kontextfreien Grammatiken ;;
;; fuer die Algorithmisch Rekursive Sequanzanalyse ;;
;; zu ueberpruefen ;;
;; Modellcharakter hat allein der Quelltext. ;;
;;;;;

```



```

;; Die Produktionen --> sind entsprechend ihrer ;;
;; emp. Auftrittswahrscheinlichkeit gewichtet ;;
;; Waehrend die Kanten des Strukturbaumes ein Top-down-Modell ;;
;; wiedergeben, bilden die Produktionen ;;
;; des Kategoriensystem-Systems (K-System) ;;
;; ein Left-to-the-Right-Modell ;;

```

```

;;
;;;;;

```

```

;;;;;
;; Verkaufsgespraech := VKG ;;

```



```
;; mit empirischen Protokollen ueberprueft ;;
;; - Bei Bedarf werden Korrekturen am K-System vorgenommen ;;
;; und die Simulation wird wiederholt ;;
;; ;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
```

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;
;; Welt 3 Popper ;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
```

```
(setq w3
'(
  (anfang 100 (s vkg)) ;; hier nur Fallstruktur Verkaufsgespraech
  ((s vkg) 100 ende)
)
)
```

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;
;; Kunde teilt Bedarf mit, Verkaefer spiegelt Bedarf Kunde ;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
```

```
(setq bbd
'(
  (kbbd 100 vbdd)
)
)
```

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;
;; wechselseitige Bedarfsargumentation nach Bedarfsmitteilung ;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
```

```
(setq ba
'(
  (kba 100 vba)
)
)
```

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;
;; wechselseitige Einwandsabklaerung ;;

```



```
;;                                                    ;;
;; Verkaufsteil                                       ;;
;; im Anschluss an Begruessung                       ;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
```

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

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;                                                    ;;
;; Begruessung                                       ;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
```

```
(setq bg
 '(
  (kg 100 vbg)
 )
)
```

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;                                                    ;;
;; Verabschiedung                                    ;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
```

```
(setq av
 '(
  (kav 100 vav)
 )
)
```

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;                                                    ;;
;; Verkaufsgespraech                                 ;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
```

```
(setq vkg
 '(
  ((s bg)100(s vt))
  ((s vt)50(s vt))
  ((s vt)100(s av))
 )
)
```

```

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;
;; Algorithmus ueber generativer Struktur
;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

```

```

;; Generiert die Sequenz
(defun gs (st r) ;; Uebergabe Sequenzstelle und Regelliste
(cond

```

```

  ;; gibt nil zurueck, wenn das Sequenzende erreicht ist
  ((equal st nil) nil)

```

```

  ;; gibt terminale Sequenzstelle mit Nachfolgern zurueck
  ((atom st)(cons st(gs(next st r(random 101))r)))

```

```

  ;; gibt expand. nichtterm. Sequenzstelle mit Nachfolger zurueck
  (t (cons(eval st)(gs(next st r(random 101))r)))
)
)

```

```

;; Generiert nachfolgende Sequenzstelle
(defun next (st r z) ;; Sequenzstelle, Regeln und Haeufigkeitsmass
(cond

```

```

  ;; gibt nil zurueck, wenn das Sequenzende erreicht ist
  ((equal r nil)nil)

```

```

  ;; waehlt Nachfolger mit Auftrittsmass h

```

```

  (
    (
      and(<= z(car(cdr(car r))))
      (equal st(car(car r)))
    )
    (car(reverse(car r)))
  )
)

```

```

  ;; in jedem anderen Fall wird Regelliste weiter durchsucht
  (t(next st (cdr r)z))
)
)

```

```

;; waehlt erste Sequenzstelle aus Regelliste
;;vordefinierte funktion first wird ueberschrieben, alternative
umbenennen

```

```

(defun first (list)
(car(car list))

```

```

)
;; startet Simulation fuer eine Fallstruktur
(defun s (list) ;; die Liste mit dem K-System wird uebergeben
(gs(first list)list)
)

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;
;; Ruft den Algorithmus auf / Welt 3 Popper /alt. jew. Fallstrukt.;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;; alternativ (s vkg) / von der Konsole aus (s w3) oder (s vkg)
(s w3)

```

```

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)

```

Ein umfangreicheres und um die Klammern bereinigtes Beispiel:

```

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

```

Das linguistische Korpus in diesem Beispiel: Die Worte des Korpus sind durch Leerzeichen getrennt. Die Worte des Korpus sind Kategorien, die bei einer qualitativen Interpretation des Transkriptes eines Verkaufsgespraches den wechselnden Interakten von Käufer und Verkäufer zugeordnet 1993, 1994 wurden. Die Tondateien, die Transkripte, die Interpretationen und die erstellten Quellcodes (Induktor Scheme, Parser Pascal, Transduktor Lisp sind an dem Ort zum download frei verfügbar, an dem sich diese Jupyter Notebook Datei befindet).

Das Programm liest den Korpus aus einer Datei ein und extrahiert die Terminalsymbole, indem es alle Substrings sucht, die mit "K" oder "V" beginnen und aus mindestens einem Großbuchstaben bestehen. Die vorangestellten "K" oder "V" werden aus den Terminalsymbolen entfernt, um die Nonterminalsymbole zu erhalten. Dann werden die Regelproduktionen erstellt, indem für jedes Nonterminalsymbol alle Terminalsymbole gesammelt werden, die diesem Symbol entsprechen. Schließlich gibt das Programm die Grammatikregeln und das Startsymbol aus.

```

PROGRAM parser (INPUT,OUTPUT);
USES CRT;
(*****
*****)
(* Paul Koop Chart Parser VKG
*)
(*
*)
(*****
*****)

(*-----
---*)
(* Vereinbarungsteil
*)

(*-----
---*)

CONST
  c0      = 0;
  c1      = 1;
  c2      = 2;
  c3      = 3;
  c4      = 4;
  c5      = 5;
  c10     = 10;
  c11     = 11;
  cmax    = 80;
  cwort   = 20;
  CText   :  STRING(.cmax.) = '';
  datei   =  'LEXIKONVKG.ASC';
  blank   =  ' ';

CopyRight
=  'Demo-Parser Chart-Parser Version 1.0(c)1992 by Paul Koop';

TYPE
TKategorien      = ( Leer, VKG, BG, VT, AV, B, A, BBD, BA, AE, AA,
                    KBG, VBG, KBBB, VBBB, KBA, VBA, KAE, VAE,
                    KAA, VAA, KAV, VAV);

PTKategorienListe = ^TKategorienListe;
TKategorienListe  = RECORD
  Kategorie :TKategorien;
  weiter    :PTKategorienListe;
END;

```

```

PTKante      = ^TKante;
PTKantenListe = ^TKantenListe;

TKantenListe = RECORD
    kante:PTKante;
    next :PTKantenListe;
END;

TKante       = RECORD
    Kategorie :TKategorien;
    vor,
    nach,
    zeigt      :PTKante;
    gefunden   :PTKantenListe;
    aktiv      :BOOLEAN;
    nummer     :INTEGER;
    nachkomme  :BOOLEAN;
    CASE Wort:BOOLEAN OF
        TRUE :
            (inhalt:STRING(.cwort.));
        FALSE:
            (gesucht :PTKategorienListe);
    END;

TWurzel      = RECORD
    spalte,
    zeigt     :PTKante;
END;

TEintrag     = RECORD
    A,I      :PTKante;
END;

PTAgenda     = ^TAgenda;
TAgenda      = RECORD
    A,I      :PTKante;
    next,
    back     :PTAgenda;
END;

PTLexElem    = ^TLexElem;
TLexElem     = RECORD
    Kategorie:TKategorien;
    Terminal  :STRING(.cwort.);
    naechstes:PTLexElem;
END;

```

```

TGrammatik = ARRAY (.c1..c10.)
              OF
              ARRAY (.c1..c4.)
              OF TKategorien;
CONST
Grammatik :   TGrammatik =
              (
                (VKG, BG,      VT,   AV),
                (BG,  KBG,     VBG,  Leer),
                (VT,  B,       A,    Leer),
                (AV,  KAV,     VAV,  Leer),
                (B,   BBd,     BA,   Leer),
                (A,   AE,      AA,   Leer),
                (BBd, KBBd,    VBBd, Leer),
                (BA,  KBA,     VBA,  Leer),
                (AE,  KAE,     VAE,  Leer),
                (AA,  KAA,     VAA,  Leer)
              );

```

```

nummer :INTEGER = c0;

```

```

(*-----
---*)
(* Variablen
*)
(*-----
---*)

```

```

VAR
Wurzel,
Pziel      : TWurzel;
Pneu       : PTKante;

Agenda,
PAgenda,
Paar       : PTAgenda;

LexWurzel,
LexAktuell,
LexEintrag : PTLexElem;
Lexikon    : Text;

```

```

(*****
*****
(* FUNKTIONEN

```

```
*)
(*****
*****)
```

```
(*-----
---*)
(* KantenZaehler
*)
```

```
(*-----
---*)
```

```
FUNCTION NimmNummer:INTEGER;
BEGIN
  Nummer := Nummer + c1;
  NimmNummer := Nummer
END;
```

```
(*****
*****)
(* PROZEDUREN
*)
(*****
*****)
```

```
(*-----
---*)
(* LexikonLesen
*)
```

```
(*-----
---*)
```

```
PROCEDURE LiesDasLexikon (VAR f:Text;
                           G:TGrammatik;
                           l:PTLexElem);

VAR
  zaehler :INTEGER;
  z11      : 1..c11;
  z4       : 1.. c4;
  ch       : CHAR;
```

```

st5      : STRING(.c5.);

BEGIN
  ASSIGN(f,datei);
  LexWurzel := NIL;
  RESET(f);
  WHILE NOT EOF(f)
  DO
    BEGIN
      NEW(LexEintrag);
      IF LexWurzel = NIL
      THEN
        BEGIN
          LexWurzel := LexEintrag;
          LexAktuell:= LexWurzel;
          LexEintrag^.naechstes := NIL;
        END
      ELSE
        BEGIN
          LexAktuell^.naechstes := LexEintrag;
          LexEintrag^.naechstes := NIL;
          LexAktuell           := LexAktuell^.naechstes;
        END;
      LexEintrag^.Terminal := '';
      st5 := '';
      FOR Zaehler := c1 to c5
      DO
        BEGIN
          READ(f,ch);
          st5 := st5 + UPCASE(ch)
        END;
      REPEAT
        READ(f,ch);
        LexEintrag^.terminal := LexEintrag^.Terminal + UPCASE(ch);
      UNTIL EOLN(f);
      READLN(f);
      IF st5 = 'KBG**' THEN LexEintrag^.Kategorie := KBG ELSE
      IF st5 = 'VBG**' THEN LexEintrag^.Kategorie := VBG ELSE
      IF st5 = 'KBBD*' THEN LexEintrag^.Kategorie := KBBD ELSE
      IF st5 = 'VBBD*' THEN LexEintrag^.Kategorie := VBBD ELSE
      IF st5 = 'KBA**' THEN LexEintrag^.Kategorie := KBA ELSE
      IF st5 = 'VBA**' THEN LexEintrag^.Kategorie := VBA ELSE
      IF st5 = 'KAE**' THEN LexEintrag^.Kategorie := KAE ELSE
      IF st5 = 'VAE**' THEN LexEintrag^.Kategorie := VAE ELSE
      IF st5 = 'KAA**' THEN LexEintrag^.Kategorie := KAA ELSE
      IF st5 = 'VAA**' THEN LexEintrag^.Kategorie := VAA ELSE
      IF st5 = 'KAV**' THEN LexEintrag^.Kategorie := KAV ELSE
      IF st5 = 'VAV**' THEN LexEintrag^.Kategorie := VAV
    END;
  END;
END;

```



```

(*-----*)
---*)
(* SatzLesen
*)

(*-----*)
---*)

```

```

PROCEDURE LiesDenSatz;
VAR
  satz:      STRING(.cmax.);
  zaehler:   INTEGER;
BEGIN
  CLRSCR;
  WRITELN(CopyRight);
  WRITE('-----> ');
  Wurzel.spalte := NIL;
  Wurzel.zeigt  := NIL;
  READLN(satz);
  FOR zaehler := c1 to LENGTH(satz)
  DO satz(.zaehler.) := UPCASE(satz(.zaehler.));
  Satz := Satz + blank;
  Writeln('-----> ',satz);
  WHILE satz <> ''
  DO
  BEGIN
    NEW(Pneu);
    Pneu^.nummer :=NimmNummer;
    Pneu^.wort   := TRUE;
    NEW(Pneu^.gefunden);
    Pneu^.gefunden^.kante := Pneu;
    pneu^.gefunden^.next := NIL;
    Pneu^.gesucht        := NIL;
    Pneu^.nachkomme     :=FALSE;
    IF Wurzel.zeigt = NIL
    THEN
      BEGIN
        Wurzel.zeigt := pneu;
        Wurzel.spalte:= pneu;
        PZiel.spalte := pneu;
        PZiel.zeigt  := Pneu;
        pneu^.vor    := NIL;
        Pneu^.zeigt  := NIL;
        Pneu^.nach   := NIL;
      END
    ELSE
      BEGIN

```

```

    Wurzel.zeigt^.zeigt := Pneu;
    Pneu^.vor           := Wurzel.zeigt;
    Pneu^.nach         := NIL;
    Pneu^.zeigt        := NIL;
    Wurzel.zeigt       := Wurzel.zeigt^.zeigt;
  END;
  pneu^.aktiv := false;
  pneu^.inhalt := COPY(satz,c1,POS(blank,satz)-c1);
  LexAktuell := LexWurzel;
  WHILE LexAktuell <> NIL
  DO
    BEGIN
      IF LexAktuell^.Terminal = pneu^.inhalt
      Then
        BEGIN
          pneu^.Kategorie := LexAktuell^.Kategorie;
        END;
        LexAktuell := LexAktuell^.naechstes;
      END;
      DELETE(satz,c1,POS(blank,satz));
    END;
  END;
END;

```

```

(*-----*)
---*)
(* Regel3KanteInAgendaEintragen
*)

```

```

(*-----*)
---*)

```

```

PROCEDURE Regel3KanteInAgendaEintragen (Kante:PTKante);
VAR
  Wurzel,
  PZiel :TWurzel;
PROCEDURE NeuesAgendaPaarAnlegen;
BEGIN
  NEW(paar);
  IF Agenda = NIL
  THEN
    BEGIN
      Agenda := Paar;
      Pagenda:= Paar;
      Paar^.next := NIL;
      Paar^.back := NIL;
    END;
  END;
END;

```

```

    END
  ELSE
    BEGIN
      PAgenda^.next := Paar;
      Paar^.next    := NIL;
      Paar^.back    := Pagenda;
      Pagenda       := Pagenda^.next;
    END;
  END;
END;

BEGIN
  IF Kante^.aktiv
  THEN
    BEGIN
      Wurzel.zeigt := Kante^.zeigt;
      WHILE wurzel.zeigt <> NIL
      DO
        BEGIN
          IF NOT(wurzel.zeigt^.aktiv)
          THEN
            BEGIN
              NeuesAgendaPaarAnlegen;
              paar^.A := kante;
              paar^.I := wurzel.zeigt;
            END;
            Wurzel.zeigt := Wurzel.zeigt^.nach
          END
        END
      ELSE
        BEGIN
          PZiel.zeigt := Kante;
          WHILE NOT(PZiel.zeigt^.Wort)
          DO PZiel.zeigt := PZiel.zeigt^.Vor;
          Wurzel.zeigt := PZiel.zeigt;
          Wurzel.Spalte := PZiel.zeigt;
          PZiel.Spalte := PZiel.zeigt;
          WHILE wurzel.spalte <> NIL
          DO
            BEGIN
              WHILE wurzel.zeigt <> NIL
              DO
                BEGIN
                  IF wurzel.zeigt^.aktiv
                  AND (Wurzel.zeigt^.zeigt = PZiel.spalte)
                  THEN
                    BEGIN
                      NeuesAGendaPaarAnlegen;
                      paar^.I := kante;
                      paar^.A := wurzel.zeigt;
                    END;
                END
            END
          END;
        END
      END;
    END
  END;

```



```

VAR
  Wurzel          :TWurzel;
  PHilfe,
  PGesuchteKategorie :PTKategorienListe;
  zaehler,
  zaehler2        :INTEGER;

BEGIN
  Wurzel.zeigt := Kante;
  Wurzel.spalte:= Kante;
  WHILE Wurzel.zeigt^.nach <> NIL
  DO Wurzel.zeigt := Wurzel.zeigt^.nach;
  FOR zaehler := c1 To c11
  DO
    IF (kategorie = Gram(.zaehler,c1.))
    AND (kategorie <> Leer)
    THEN
      BEGIN
        Gram(.zaehler,c1.) := Leer;
        NEW(pneu);
        Wurzel.zeigt^.nach := pneu;
        pneu^.nummer       := NimmNummer;
        pneu^.vor           := Wurzel.zeigt;
        Pneu^.nach         := NIL;
        Pneu^.zeigt        := wurzel.spalte;
        Wurzel.zeigt       := Wurzel.zeigt^.nach;
        pneu^.aktiv        := true;
        pneu^.kategorie     := kategorie;
        Pneu^.Wort          := false;
        Pneu^.gesucht       := NIL;
        Pneu^.gefunden      := NIL;
        Pneu^.nachkomme    := FALSE;
        FOR zaehler2 := c2 TO c4
        DO
          BEGIN
            IF Gram(.zaehler,zaehler2.) <> Leer
            THEN
              BEGIN
                NEW(PGesuchteKategorie);
                PGesuchteKategorie^.weiter:= NIL;
                PGesuchteKategorie^.Kategorie :=
Gram(.zaehler,zaehler2.);
                IF Pneu^.gesucht = NIL
                THEN
                  BEGIN
                    PHilfe          := PGesuchteKategorie;
                    Pneu^.gesucht := PHilfe;
                  END
                ELSE
                  BEGIN

```

```

                PHilfe^.weiter := PGesuchteKategorie;
                PHilfe         := PHilfe^.weiter;
            END
        END
    END;
    Regel3KanteInAgendaEintragen (pneu);
    Regel2EineNeueKanteAnlegen(Wurzel.spalte,
                                pneu^.gesucht^.kategorie,gram);
END;
END;

```

```

(*-----*)
---*)
(* Regel1EineKanteErweitern
*)

```

```

(*-----*)
---*)

```

```

PROCEDURE Regel1EineKanteErweitern(paar:PTAgenda);
VAR
    PneuHilf,Pneugefneu,AHilf :PTKantenListe;
BEGIN

    IF paar^.I^.kategorie = paar^.A^.gesucht^.kategorie
    THEN
        BEGIN
            NEW(pneu);
            pneu^.nummer      := NimmNummer;
            pneu^.kategorie   := Paar^.A^.kategorie;
            (*-----*)
            Pneu^.gefunden := NIL;
            AHilf := Paar^.A^.gefunden;

            WHILE AHilf <> NIL
            DO
                BEGIN
                    NEW(Pneugefneu);
                    IF Pneu^.gefunden = NIL
                    THEN
                        BEGIN
                            Pneu^.gefunden := Pneugefneu;
                            PneuHilf      := Pneu^.gefunden;
                            PneuHilf^.next := NIL;
                        END
                    ELSE

```

```

    BEGIN
        PneuHilf^.next := Pneugefneu;
        PneuHilf       := PneuHilf^.next;
        PneuHilf^.next := NIL;
    END;

    Pneugefneu^.kante := AHilf^.kante;
    AHilf             := AHilf^.next;
END;

NEW(Pneugefneu);
IF Pneu^.gefunden = NIL
    THEN
        BEGIN
            Pneu^.gefunden := Pneugefneu;
            Pneugefneu^.next := NIL;
        END
    ELSE
        BEGIN
            PneuHilf^.next := Pneugefneu;
            PneuHilf       := PneuHilf^.next;
            PneuHilf^.next := NIL;
        END;
        Pneugefneu^.kante := Paar^.I;
    (*-----*)
    Pneu^.wort := FALSE;
    IF Paar^.A^.gesucht^.weiter = NIL
        THEN Pneu^.gesucht := NIL
        ELSE Pneu^.gesucht := Paar^.A^.gesucht^.weiter;
        Pneu^.nachkomme := TRUE;

    IF pneu^.gesucht = NIL
        THEN Pneu^.aktiv := false
        ELSE Pneu^.aktiv := true;

    WHILE Paar^.A^.nach <> NIL
        DO Paar^.A := Paar^.A^.nach;

        Paar^.A^.nach := pneu;
        pneu^.vor     := Paar^.A;
        pneu^.zeigt   := Paar^.I^.zeigt;
        pneu^.nach    := NIL;

        Regel3KanteInAgendaEintragen (pneu);
        IF Pneu^.aktiv
            THEN Regel2EineNeueKanteAnlegen(Pneu^.zeigt,
pneu^.gesucht^.kategorie,Grammatik);
    END;

```

```
END;
```

```
(*-----  
---*)  
(* SatzAnalyse  
*)
```

```
(*-----  
---*)
```

```
PROCEDURE SatzAnalyse;  
BEGIN  
WHILE Agenda <> NIL  
DO  
BEGIN  
NimmAgendaEintrag(Paar);  
RegellEineKanteErweitern(Paar);  
END;
```

```
END;
```

```
(*-----  
---*)  
(* SatzAusgabe  
*)
```

```
(*-----  
---*)
```

```
PROCEDURE GibAlleSatzalternativenAus;  
CONST  
BlankAnz:INTEGER = c2;  
VAR  
PHilf :PTkantenListe;
```

```
PROCEDURE SatzAusgabe(Kante:PTKante;BlankAnz:INTEGER);  
VAR
```

```
Zaehler:INTEGER;  
PHilf :PTkantenListe;  
BEGIN  
FOR Zaehler := c1 TO BlankAnz DO WRITE(blank);
```

```
IF Kante^.kategorie = VKG THEN WRITELN ('VKG ') ELSE  
IF Kante^.kategorie = BG THEN WRITELN ('BG ') ELSE  
IF Kante^.kategorie = VT THEN WRITELN ('VT ') ELSE  
IF Kante^.kategorie = AV THEN WRITE ('AV ') ELSE
```



```

IF Kante^.kategorie = B      THEN WRITELN ('B  ') ELSE
IF Kante^.kategorie = A      THEN WRITE  ('A  ') ELSE
IF Kante^.kategorie = BBD    THEN WRITE  ('BBD ') ELSE
IF Kante^.kategorie = BA     THEN WRITELN ('BA  ') ELSE
IF Kante^.kategorie = AE     THEN WRITE  ('AE  ') ELSE
IF Kante^.kategorie = AA     THEN WRITE  ('AA  ') ELSE
IF Kante^.kategorie = KBG    THEN WRITELN ('KBG ') ELSE
IF Kante^.kategorie = VBG    THEN WRITELN ('VBG ') ELSE
IF Kante^.kategorie = KBBD   THEN WRITELN ('KBBD') ELSE
IF Kante^.kategorie = VBBD   THEN WRITE  ('VBBD') ELSE
IF Kante^.kategorie = KBA    THEN WRITELN ('KBA ') ELSE
IF Kante^.kategorie = VBA    THEN WRITE  ('VBA ') ELSE
IF Kante^.kategorie = KAE    THEN WRITE  ('KAE ') ELSE
IF Kante^.kategorie = VAE    THEN WRITELN ('VAE ') ELSE
IF Kante^.kategorie = KAA    THEN WRITE  ('KAA ') ELSE
IF Kante^.kategorie = VAA    THEN WRITE  ('VAA ') ELSE
IF Kante^.kategorie = KAV    THEN WRITE  ('KAV ') ELSE
IF Kante^.kategorie = VAV    THEN WRITE  ('VAV ');

IF Kante^.wort
THEN
  WRITELN('----> ',Kante^.inhalt)
ELSE
  BEGIN
    PHilf := Kante^.gefunden;
    WHILE PHilf <> NIL
    DO
      BEGIN
        Satzausgabe(PHilf^.kante,Blankanz+c1);
        PHilf := PHilf^.next;
      END
    END
  END;

```

```

BEGIN
  WHILE Wurzel.zeigt^.vor <> NIL
  DO Wurzel.zeigt := Wurzel.zeigt^.vor;

  WHILE Wurzel.zeigt <> NIL
  DO
    BEGIN
      IF (Wurzel.zeigt^.kategorie = VKG)
      AND ((NOT(Wurzel.zeigt^.aktiv))
      AND (wurzel.zeigt^.zeigt = NIL))
      THEN
        BEGIN
          WRITELN('VKG');
          PHilf := Wurzel.zeigt^.gefunden;
          WHILE PHilf <> NIL
          DO

```

```

        BEGIN
            Satzausgabe(PHilf^.kante,Blankanz+c1);
            PHilf := PHilf^.next;
        END
    END;
    Wurzel.zeigt := Wurzel.zeigt^.nach;
END;

END;

```

```

(*-----*)
---*)
(* FreigabeDesBenutztenSpeicherplatzes
*)

(*-----*)
---*)

```

```

PROCEDURE LoescheDieListe;
PROCEDURE LoescheWort(kante :PTKante);
PROCEDURE LoescheSpalte(kante:PTKante);
VAR
    Pgefunden :PTKantenListe;
    Pgesucht :PTKategorienListe;
PROCEDURE LoescheGesucht(p:PTKategorienListe);
BEGIN
    IF p^.weiter <> NIL
    THEN LoescheGesucht(p^.weiter);
    IF P <> NIL THEN DISPOSE(P);
END;
PROCEDURE LoescheGefunden(Kante:PTKante;p:PTKantenListe);
BEGIN
    IF p^.next <> NIL
    THEN LoescheGefunden(Kante,p^.next);
    DISPOSE(P);
END;
BEGIN(*LoescheSpalte*)
    IF Kante^.nach <> NIL
    THEN LoescheSpalte(kante^.nach);
    IF (NOT Kante^.nachkomme) AND ((Kante^.gesucht <> NIL)
    AND (NOT Kante^.wort))
    THEN LoescheGesucht(Kante^.gesucht);
    IF Kante^.gefunden <> NIL
    THEN LoescheGefunden(Kante,Kante^.gefunden);
    DISPOSE(Kante)
END;(*LoescheSpalte*)
BEGIN(*LoescheWort*)
    IF Kante^.zeigt <> NIL
    THEN LoescheWort(Kante^.zeigt);

```

```

    LoescheSpalte(Kante);
    END;(*LoescheWort*)
BEGIN(*LoescheDieListe*)
    WHILE Wurzel.spalte^.vor <> NIL
        DO Wurzel.spalte := Wurzel.spalte^.vor;
        LoescheWort(Wurzel.spalte);
    END;(*LoescheDieListe*)
(*****
***** )
(* HAUPTPROGRAMM DES CHART PARSERS
*)
(*****
***** )

BEGIN
    Agenda := NIL;
    PAgenda := Agenda;
    LiesDasLexikon(Lexikon, Grammatik, LexWurzel);
    LiesDenSatz;
    WHILE Wurzel.spalte^.vor <> NIL
        DO Wurzel.spalte := Wurzel.spalte^.vor;
        Regel2EineNeueKanteAnlegen(Wurzel.spalte, VKG, Grammatik);
        SatzAnalyse;
        GibAlleSatzalternativenAus;
        LoescheDieListe;
(*****
***** )
(* ENDE DES HAUPTPROGRAMMS DES CHART PARSERS
*)
(*****
***** )

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

```

```

import re

# Lesen des Korpus aus einer Datei
#with open("VKGKORPUS.TXT", "r") as f:
#    korpus = f.read()
korpus = "KBG VBG KBBB VBBD KBA VBA KAE VAE KAA VAA KBBB VBBD KBA VBA
KBBB VBBD KBA VBA KBBB VBBD KBA VBA KAE VAE KAA VAA KAV VAV"
# Extrahieren der Terminalsymbole aus dem Korpus
terminals = re.findall(r"[KV][A-Z]+", korpus)

# Entfernen der vorangestellten K- oder V-Zeichen aus den
Terminalsymbolen
non_terminals = list(set([t[1:] for t in terminals]))

# Erzeugen der Regelproduktionen
productions = []
for nt in non_terminals:
    rhs = [t for t in terminals if t[1:] == nt]
    productions.append((nt, rhs))

# 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
BBB -> KBBB | VBBB | KBBB | VBBB | KBBB | VBBB | KBBB | VBBB
Startsymbol: VKG

```

Die Nonterminalsymbole sind hier jeweils die ersten Buchstaben der Terminalsymbole ohne das vorangestellte "K" oder "V". Die Startregel ist 'VK', was bedeutet, dass der Verkäufer (V) die Konversation beginnt und der Käufer (K) antwortet. Beachten Sie, dass die Produktionsregeln in beide Richtungen funktionieren, da die Konversation zwischen Verkäufer und Käufer wechselseitig ist.

```
import re
from collections import defaultdict

corpus = "KBG VBG KBBB VBBD KBA VBA KAE VAE KAA VAA KBBB VBBD KBA VBA
KBBB VBBD KBA VBA KBBB VBBD KBA VBA KAE VAE KAA VAA KAV VAV"

# Erstellen eines Wörterbuchs, um die Anzahl der Vorkommen von
# Terminalsymbolen zu zählen.
vocab = defaultdict(int)
for word in corpus.split():
    vocab[word] += 1

# Entfernen von Präfixen K und V von Terminalsymbolen.
terminals = list(set([re.sub(r'^[KV]', '', w) for w in vocab.keys()]))

# Erstellen der Produktionen für die Grammatik.
productions = []
for w in vocab.keys():
    if re.match(r'^K', w):
        lhs = 'K'
    elif re.match(r'^V', w):
        lhs = 'V'
    else:
        lhs = re.sub(r'^[KV]', '', w)
    rhs = w
    productions.append((lhs, [rhs]))

# Erstellen der Startregel der Grammatik.
start = 'VK'

# Ausgabe der Grammatik.
print(f'Start: {start}')
for lhs, rhs in productions:
    print(f'{lhs} -> {" ".join(rhs)}')

Start: VK
K -> KBG
V -> VBG
K -> KBBB
V -> VBBD
K -> KBA
V -> VBA
K -> KAE
V -> VAE
```

```
K -> KAA
V -> VAA
K -> KAV
V -> VAV
```

Das Programm liest das gegebene Korpus ein und extrahiert die nicht-terminalen Symbole, indem es alle Symbole entfernt, die mit "K" oder "V" beginnen. Dann iteriert es über das Korpus und zählt die Produktionsregeln, indem es für jedes Vorkommen eines nicht-terminalen Symbols den nachfolgenden Terminalsymbol zählt. Schließlich berechnet es die Wahrscheinlichkeiten der Produktionsregeln, indem es die Häufigkeit jedes rechten Seiten eines nicht-terminalen Symbols durch die Gesamtanzahl der Vorkommen des linken Symbols dividiert.

Das Programm gibt dann die induzierte Grammatik aus, wobei die Wahrscheinlichkeiten der Produktionsregeln angezeigt werden.

```
from collections import defaultdict
import random

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

# read in the corpus
corpus = "KBG VBG KBBB VBBD KBA VBA KAE VAE KAA VAA KBBB VBBD KBA VBA
KBBB VBBD KBA VBA KBBB 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]
    total_count = len(productions)
    probabilities = defaultdict(float)
    for rhs in productions:
        probabilities[rhs] += 1.0
    for rhs in probabilities.keys():
        probabilities[rhs] /= total_count
    grammar[lhs] = probabilities

# print the grammar
```

```

print("Grammar:")
for lhs in grammar.keys():
    print(lhs + " ->")
    for rhs in grammar[lhs].keys():
        print(" " + rhs + " : " + str(grammar[lhs][rhs]))

```

```

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

```

Hier ist ein Beispielprogramm in Python, das die gegebene probabilistische Grammatik nutzt, um das gegebene Korpus zu parsen:

```

import random

# Die gegebene probabilistische Grammatik
grammar = {
    'BG': {'VBG': 0.5, 'KBBB': 0.5},
    'BBD': {'VBBD': 0.5, 'KBA': 0.5},
    'BA': {'VBA': 0.5, 'KAE': 0.25, 'KBBB': 0.25},
    'AE': {'VAE': 0.5, 'KAA': 0.5},
    'AA': {'VAA': 0.5, 'KAV': 0.25, 'KBBB': 0.25},
    'AV': {'VAV': 1.0},
}

# Das zu parsende Korpus
corpus = ['KBG', 'VBG', 'KBBG', 'VBBD', 'KAE', 'VBA', 'KAE', 'VAA',
          'KBBG', 'VBBD', 'KBA', 'VBA', 'KBBG', 'VBBD', 'KBA', 'VBA', 'KAE',
          'VAE', 'KAA', 'VAA', 'KAV', 'VAV']

# Initialisiere die Tabelle mit leeren Einträgen
chart = [[{} for i in range(len(corpus) + 1)] for j in
range(len(corpus) + 1)]

```

```

# Fülle die Tabelle mit den Terminalsymbolen und den
Wahrscheinlichkeiten
for i in range(len(corpus)):
    for lhs, rhs_probs in grammar.items():
        for rhs, prob in rhs_probs.items():
            if rhs == corpus[i]:
                chart[i][i+1][lhs] = {'prob': prob, 'prev': None}

# Fülle die Tabelle mit den Nichtterminalsymbolen und den
Wahrscheinlichkeiten
for span in range(2, len(corpus) + 1):
    for start in range(len(corpus) - span + 1):
        end = start + span
        for split in range(start + 1, end):
            for lhs, rhs_probs in grammar.items():
                for rhs, prob in rhs_probs.items():
                    if len(rhs) == 2:
                        left, right = rhs
                        if left in chart[start][split] and right in
chart[split][end]:
                            prod_prob = prob * chart[start][split]
[left]['prob'] * chart[split][end][right]['prob']
                            if lhs not in chart[start][end] or
prod_prob > chart[start][end][lhs]['prob']:
                                chart[start][end][lhs] = {'prob':
prod_prob, 'prev': (split, left, right)}

# Ausgabe des Parsing-Baums
def print_tree(start, end, symbol):
    if symbol in chart[start][end]:
        if chart[start][end][symbol]['prev'] is None:
            return [symbol]
        split, left, right = chart[start][end][symbol]['prev']
        return [symbol, print_tree(start, split, left),
print_tree(split, end, right)]
    else:
        return []

# Parse den Satz und gib den resultierenden Parse-Baum aus
parse_tree = print_tree(0, len(corpus), 'BG')
print(parse_tree)

```

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.

Hier ist ein Beispielprogramm, das den Agenten K startet und das Terminalzeichen "KBG" setzt. Der Agent V generiert dann das nächste Terminalzeichen basierend auf der übergebenen Grammatik und dem letzten Terminalzeichen "KBG". Dies wird dann in einer Schleife fortgesetzt, bis eine maximale Anzahl von Terminalzeichen erreicht ist.

```
import random

# Grammatik als probabilistisches Bayessches Netz definieren
grammar = {
    "BG": {"VBG": 0.5, "KBBB": 0.5},
    "BBD": {"VBBB": 0.5, "KBA": 0.5},
    "BA": {"VBA": 0.5, "KAE": 0.25, "KBBB": 0.25},
    "AE": {"VAE": 0.5, "KAA": 0.5},
    "AA": {"VAA": 0.5, "KBBB": 0.25, "KAV": 0.25},
    "AV": {"VAV": 1.0}
}

# Funktion zur Generierung des nächsten Terminalzeichens
def generate_next_terminal(grammar, last_terminal):
    # Wähle die Produktion basierend auf dem letzten Terminalzeichen
    # und der Grammatik aus
    productions = grammar[last_terminal]
    production = random.choices(list(productions.keys()),
                                list(productions.values()))[0]
    return production

# Maximale Anzahl von Terminalzeichen
max_length = 10

# Startzeichen für Agent K
last_terminal = "KBG"
```

```

# Schleife für Interaktion zwischen Agent K und Agent V
for i in range(max_length):
    # Agent K generiert das nächste Terminalzeichen basierend auf der
    # Grammatik und dem letzten gesetzten Zeichen
    next_terminal = generate_next_terminal(grammar, last_terminal)
    print("Agent K: ", last_terminal)

    # Agent V generiert das nächste Terminalzeichen basierend auf der
    # Grammatik und dem letzten gesetzten Zeichen
    last_terminal = generate_next_terminal(grammar, next_terminal)
    print("Agent V: ", next_terminal)

# Letztes Terminalzeichen ausgeben, das von Agent K generiert wurde
print("Letztes Terminalzeichen: ", last_terminal)

```

Agent K: KBG Agent V: KBBDD Agent K: KBBDD Agent V: KAE Agent K: KAE Agent V: VAE
Agent K: VAE Agent V: KAA Agent K: KAA Agent V: VAA Agent K: VAA Letztes
Terminalzeichen: VAA

Es ist möglich, das Beispielprogramm entsprechend zu erweitern, um die genannten Eigenschaften der Agenten und die Rollenverteilung zu berücksichtigen. Ein Entscheidungsbaum legt erst die Rollen der Agenten fest. Dann Handeln die Agenten nach der Handlungsgrammatik. Hier ist eine einfache erweiterte Version des Programms:

```

import random

# Die gegebene probabilistische Grammatik
grammar = {
    'BG': {'VBG': 0.5, 'KBBDD': 0.5},
    'BBD': {'VBBD': 0.5, 'KBA': 0.5},
    'BA': {'VBA': 0.5, 'KAE': 0.25, 'KBBDD': 0.25},
    'AE': {'VAE': 0.5, 'KAA': 0.5},
    'AA': {'VAA': 0.5, 'KAV': 0.25, 'KBBDD': 0.25},
    'AV': {'VAV': 1.0},
}

# Zufällige Belegung von Ware und Zahlungsmittel bei den Agenten
agent_k_ware = random.uniform(0, 100)
agent_k_zahlungsmittel = 100 - agent_k_ware
agent_v_ware = random.uniform(0, 100)
agent_v_zahlungsmittel = 100 - agent_v_ware

# Entscheidung über die Rollenverteilung basierend auf Ware und
# Zahlungsmittel
if agent_k_ware > agent_v_ware:
    agent_k_role = 'Käufer'
    agent_v_role = 'Verkäufer'
else:
    agent_k_role = 'Verkäufer'

```

```

agent_v_role = 'Käufer'

# Ausgabe der Rollenverteilung und der Belegung von Ware und
Zahlungsmittel
print("Agent K: Rolle =", agent_k_role, "| Ware =", agent_k_ware, "|
Zahlungsmittel =", agent_k_zahlungsmittel)
print("Agent V: Rolle =", agent_v_role, "| Ware =", agent_v_ware, "|
Zahlungsmittel =", agent_v_zahlungsmittel)
print()

# Agent K startet den Dialog mit dem Terminalzeichen 'KBG'
last_terminal = 'KBG'

# Maximale Anzahl von Terminalzeichen im Dialog
max_terminals = 10

# Dialog-Schleife
for i in range(max_terminals):
    # Agent K generiert das nächste Terminalzeichen basierend auf der
    # Grammatik und dem letzten Terminalzeichen
    next_terminal =
    random.choices(list(grammar[last_terminal].keys()),
weights=list(grammar[last_terminal].values()))[0]

    # Agent V generiert das nächste Terminalzeichen basierend auf der
    # Grammatik und dem letzten Terminalzeichen
    next_terminal =
    random.choices(list(grammar[last_terminal].keys()),
weights=list(grammar[last_terminal].values()))[0]

    # Aktualisierung des letzten Terminalzeichens
    last_terminal = next_terminal

    # Ausgabe des aktuellen Terminalzeichens
    print("Agent K:", next_terminal)

    # Break, wenn das Terminalzeichen 'VAV' erreicht ist
    if next_terminal == 'VAV':
        break

```

```

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