Paul Koop 2023

Qualitative Sozialforschung und Große Sprachmodelle

Anhand eines protokollierten und kategorisierten Verkaufsgesprächs werde ich demonstrieren,
dass große Sprachmodelle zwar Interaktionen simulieren, jedoch nicht erklären können.
Zunächst werde ich einen generativen Transformer entwickeln und diesen mit den kategorisierten Protokollen trainieren.
Dabei wird deutlich, dass das Modell zwar in der Lage ist, einen Dialog überzeugend nachzuahmen,
jedoch keine fundierten Erklärungen zu liefern vermag.

Im Anschluss werde ich für die gleichen Daten einen Induktor, einen Transduktor und einen Parser erstellen.
Dies wird veranschaulichen, dass die mit diesen Methoden erzeugten Dialoggrammatiken im Gegensatz zum großen Sprachmodell
und zum generativen vortrainierten Transformer Erklärungswert besitzen.

Abschließend werde ich die entwickelte Grammatik in ein Multiagentensystem integrieren, das in der Lage ist,
Verkaufsgespräche zu simulieren und diese entsprechend zu erklären.

Die qualitative Sozialforschung hat den Kognitivismus verschlafen.
So wurde verpasst, die Rekonstruktion latenter Sinnstrukturen
durch die Konstruktion generativer Regeln im Sinne von Algorithmen abzusichern.
Für valide erhobene Kategoriensysteme (vg. Mayring) lassen sich algorithmische Regeln
eines endlichen Automaten angeben
(vg. Koop, Paul.: ARS, Grammar-Induction, Parser, Grammar-Transduction).

Jetzt parasitieren Posthumanismus, Poststrukturalismus und Transhumanismus die Opake KI.
Und parasitieren sie diese nicht, so sind sie wechselseitige Symbionten.

Karl Popper wird dann durch Harry Potter ersetzt und
qualitative Sozialforschung und Grosse Sprachmodelle werden zu wenig erklärenden,
aber beeindruckendem Cargo-Kult einer nichts erklärenden und alles verschleiernden Postmoderne.

Für die Algorithmisch rekursive Sequenzanalyse wurde gezeigt,
dass für das Protokoll einer Handlungssequenz
mindestens eine Grammatik angegeben werden kann
(Induktor in Scheme, Parser in Pascal, Transduktor in Lisp, vgl Koop,

P.).

ARS ist ein qualitatives Verfahren,
das latente Regeln protokollierter Handlungssequenzen
widerlegbar rekonstruieren kann.

Ein Großes Sprachmodell lässt sich so nachprogrammieren, dass es die ermittelten Kategorien einer qualitativen Inhaltsanalyse (vgl. Mayring)
rekonstruieren kann.

Der Erklärungswert eines solchen Modells ist aber vernachlässigbar, weil gerade eben nichts erklärt wird.

Um das zu zeigen, wird im Folgenden
die Nachprogrammierung eines Großen Sprachmodells beschrieben.

Aus dem Korpus der Kodierungen eines transkribierten Protokolls kann mit einem tiefen Sprachmodell
eine Simulation eines Verkaufsgespräches gefahren werden.
Der Algorithmus des tiefen Sprachmodell steht dann für die generative Struktur.
Gute Einführungen bieten:

Steinwender, J., Schwaiger, R.:
Neuronale Netze programmieren mit Python
2. Auflage 2020
ISBN 978-3-8362-7452-4

Trask, A. W.:
Neuronale Netze und Deep Learning kapieren
Der einfache Praxiseinstieg mit Beispielen in Python
1. Auflage 2020
ISBN 978-3-7475-0017-0

Hirschle, J.:
Deep Natural Language Processing
1. Auflage 2022
ISBN 978-3-446-47363-8

Die Datenstrukturen in diesem Text sind aus dem oben genannten Titel von A. W. Trask nachprogrammiert.
Daraus ist dann das tiefe Sprachmodell für Verkaufsgespäche abgeleitet.

Neuronale Netze sind mehrdimensionale, meist zweidimensionale Datenfelder rationaler Zahlen.
Eine verborgene Schicht aus voraussagenden Gewichten gewichtet die

Daten der Eingabeschicht, propagiert die Ergebnisse zur nächsten Schicht und so fort,
bis eine offene Ausgabeschicht sie dann ausgibt.

In der Trainingsphase werden die Gewichte zurückpropagiert, bei Grossen Sprachmodellen mit rekurrenten Netzwerken mit Aufmerksamkeit auf dem protokollierten Kontext.

In den dem Verständnis dienenden Beispielen wird versucht, die Spielergebnisse einer Mannschaft durch Gewichtung der Zehenzahl, der bisher gewonnenen Spiele und der Anzahl an Fans, die zukünftigen Geweinnchancen zu ermitteln.

Nur ein Eingabedatum, hier die Zehenzahl:

```
# Das Netzwerk
gewicht = 0.1
def neurales_netzwerk (eingabe, gewicht):
    ausgabe = eingabe * gewicht
    return ausgabe

# Anwendung des Netzwerkes
anzahl_der_zehen = [8.5, 9.5, 10, 9]
eingabe = anzahl_der_zehen[0]
ausgabe = neurales_netzwerk (eingabe, gewicht)
print(ausgabe)
```

```
0.8500000000000001
```

Jetzt mit drei Eingabedaten (Zehenzahl, bisherige Gewinne, Anzahl Fans):

```
def propagierungsfunktion(a,b):
    assert(len(a) == len(b))
    ausgabe = 0
    for i in range(len(a)):
        ausgabe += (a[i] * b[i])
    return ausgabe

gewicht = [0.1, 0.2, 0]

def neurales_netzwerk(eingabe, gewicht):
    ausgabe = propagierungsfunktion(eingabe,gewicht)
    return ausgabe


zehen =  [8.5, 9.5, 9.9, 9.0]
gewinnrate = [0.65, 0.8, 0.8, 0.9]
fans = [1.2, 1.3, 0.5, 1.0]
```

```
eingabe = [zehen[0],gewinnrate[0],fans[0]]
ausgabe = neurales_netzwerk(eingabe,gewicht)

print(ausgabe)
```

```
0.9800000000000001
```

Jetzt mit der Bibliothek numy (Datenfelder, Vektoren, Matrizen):

```
import numpy as ny
gewicht = ny.array([0.1, 0.2, 0])
def neurales_netzwerk(eingabe, gewicht):
    ausgabe = eingabe.dot(gewicht)
    return ausgabe

zehen      = ny.array([8.5, 9.5, 9.9, 9.0])
gewinnrate = ny.array([0.65, 0.8, 0.8, 0.9])
fans       = ny.array([1.2, 1.3, 0.5, 1.0])


eingabe = ny.array([zehen[0],gewinnrate[0],fans[0]])
ausgabe = neurales_netzwerk(eingabe,gewicht)

print(ausgabe)
```

```
0.9800000000000001
```

Die Gewichte lassen sich so lange anpassen,
bis der Fehler minimiert ist.

```
# Prinzipielles Beispiel
gewicht = 0.5
eingabe = 0.5
erwuenschte_vorhersage = 0.8

schrittweite = 0.001

for iteration in range(1101):

    vorhersage = eingabe * gewicht
    fehler = (vorhersage - erwuenschte_vorhersage) ** 2

    print("Fehler:" + str(fehler) + " Vorhersage:" + str(vorhersage))

    hoehere_vorhersage = eingabe * (gewicht + schrittweite)
    tieferer_fehler = (gewünschte_vorhersage - hoehere_orhersage) ** 2

    hoehere_vorhersage = eingabe * (gewicht - schrittweite)
    tiefere_fehler = (erwuenschte_vorhersage - tiefere_vorhersage) **
2

    if(tieferer_fehler <  hoeherer_fehler):
```

```
        gewicht = gewicht - schrittweite

    if(tieferer_fehler >  hoeherer_fehler):
        gewicht = gewicht + schrittweite

# Trask, A. W.:
# Neuronale Netze und Deep Learning kapieren
# Der einfache Praxiseinstieg mit Beispielen in Python
# 1. Auflage 2020
# ISBN 978-3-7475-0017-0

import numpy as np

# Objektklasse Datenfeld
class Tensor (object):

    def __init__(self,data,
                 autograd=False,
                 creators=None,
                 creation_op=None,
                 id=None):

        self.data = np.array(data)
        self.autograd = autograd
        self.grad = None

        if(id is None):
            self.id = np.random.randint(0,1000000000)
        else:
            self.id = id

        self.creators = creators
        self.creation_op = creation_op
        self.children = {}

        if(creators is not None):
            for c in creators:
                if(self.id not in c.children):
                    c.children[self.id] = 1
                else:
                    c.children[self.id] += 1

    def all_children_grads_accounted_for(self):
        for id,cnt in self.children.items():
            if(cnt != 0):
                return False
        return True

    def backward(self,grad=None, grad_origin=None):
        if(self.autograd):
```

```
            if(grad is None):
                grad = Tensor(np.ones_like(self.data))

            if(grad_origin is not None):
                if(self.children[grad_origin.id] == 0):
                    return
                    print(self.id)
                    print(self.creation_op)
                    print(len(self.creators))
                    for c in self.creators:
                        print(c.creation_op)
                    raise Exception("cannot backprop more than once")
                else:
                    self.children[grad_origin.id] -= 1

            if(self.grad is None):
                self.grad = grad
            else:
                self.grad += grad


            assert grad.autograd == False


            if(self.creators is not None and
               (self.all_children_grads_accounted_for() or
                grad_origin is None)):

                if(self.creation_op == "add"):
                    self.creators[0].backward(self.grad, self)
                    self.creators[1].backward(self.grad, self)

                if(self.creation_op == "sub"):
                    self.creators[0].backward(Tensor(self.grad.data),
self)

self.creators[1].backward(Tensor(self.grad.__neg__().data), self)

                if(self.creation_op == "mul"):
                    new = self.grad * self.creators[1]
                    self.creators[0].backward(new , self)
                    new = self.grad * self.creators[0]
                    self.creators[1].backward(new, self)


                if(self.creation_op == "mm"):
                    c0 = self.creators[0]
                    c1 = self.creators[1]
                    new = self.grad.mm(c1.transpose())
```

```
                    c0.backward(new)
                    new = self.grad.transpose().mm(c0).transpose()
                    c1.backward(new)

                if(self.creation_op == "transpose"):
                    self.creators[0].backward(self.grad.transpose())

                if("sum" in self.creation_op):
                    dim = int(self.creation_op.split("_")[1])
                    self.creators[0].backward(self.grad.expand(dim,

self.creators[0].data.shape[dim]))

                if("expand" in self.creation_op):
                    dim = int(self.creation_op.split("_")[1])
                    self.creators[0].backward(self.grad.sum(dim))

                if(self.creation_op == "neg"):
                    self.creators[0].backward(self.grad.__neg__())

                if(self.creation_op == "sigmoid"):
                    ones = Tensor(np.ones_like(self.grad.data))
                    self.creators[0].backward(self.grad * (self *
(ones - self)))

                if(self.creation_op == "tanh"):
                    ones = Tensor(np.ones_like(self.grad.data))
                    self.creators[0].backward(self.grad * (ones -
(self * self)))

                if(self.creation_op == "index_select"):
                    new_grad = np.zeros_like(self.creators[0].data)
                    indices_ =
self.index_select_indices.data.flatten()
                    grad_ = grad.data.reshape(len(indices_), -1)
                    for i in range(len(indices_)):
                        new_grad[indices_[i]] += grad_[i]
                    self.creators[0].backward(Tensor(new_grad))

                if(self.creation_op == "cross_entropy"):
                    dx = self.softmax_output - self.target_dist
                    self.creators[0].backward(Tensor(dx))

    def __add__(self, other):
        if(self.autograd and other.autograd):
            return Tensor(self.data + other.data,
                          autograd=True,
                          creators=[self,other],
                          creation_op="add")
```

```
        return Tensor(self.data + other.data)

    def __neg__(self):
        if(self.autograd):
            return Tensor(self.data * -1,
                          autograd=True,
                          creators=[self],
                          creation_op="neg")
        return Tensor(self.data * -1)

    def __sub__(self, other):
        if(self.autograd and other.autograd):
            return Tensor(self.data - other.data,
                          autograd=True,
                          creators=[self,other],
                          creation_op="sub")
        return Tensor(self.data - other.data)

    def __mul__(self, other):
        if(self.autograd and other.autograd):
            return Tensor(self.data * other.data,
                          autograd=True,
                          creators=[self,other],
                          creation_op="mul")
        return Tensor(self.data * other.data)

    def sum(self, dim):
        if(self.autograd):
            return Tensor(self.data.sum(dim),
                          autograd=True,
                          creators=[self],
                          creation_op="sum_"+str(dim))
        return Tensor(self.data.sum(dim))

    def expand(self, dim,copies):

        trans_cmd = list(range(0,len(self.data.shape)))
        trans_cmd.insert(dim,len(self.data.shape))
        new_data =
self.data.repeat(copies).reshape(list(self.data.shape) +
[copies]).transpose(trans_cmd)

        if(self.autograd):
            return Tensor(new_data,
                          autograd=True,
                          creators=[self],
                          creation_op="expand_"+str(dim))
        return Tensor(new_data)

    def transpose(self):
```

```
        if(self.autograd):
            return Tensor(self.data.transpose(),
                          autograd=True,
                          creators=[self],
                          creation_op="transpose")

        return Tensor(self.data.transpose())

    def mm(self, x):
        if(self.autograd):
            return Tensor(self.data.dot(x.data),
                          autograd=True,
                          creators=[self,x],
                          creation_op="mm")
        return Tensor(self.data.dot(x.data))

    def sigmoid(self):
        if(self.autograd):
            return Tensor(1 / (1 + np.exp(-self.data)),
                          autograd=True,
                          creators=[self],
                          creation_op="sigmoid")
        return Tensor(1 / (1 + np.exp(-self.data)))

    def tanh(self):
        if(self.autograd):
            return Tensor(np.tanh(self.data),
                          autograd=True,
                          creators=[self],
                          creation_op="tanh")
        return Tensor(np.tanh(self.data))

    def index_select(self, indices):

        if(self.autograd):
            new = Tensor(self.data[indices.data],
                         autograd=True,
                         creators=[self],
                         creation_op="index_select")
            new.index_select_indices = indices
            return new
        return Tensor(self.data[indices.data])

    def softmax(self):
        temp = np.exp(self.data)
        softmax_output = temp / np.sum(temp,
                                       axis=len(self.data.shape)-1,
                                       keepdims=True)
        return softmax_output
```

```
    def cross_entropy(self, target_indices):

        temp = np.exp(self.data)
        softmax_output = temp / np.sum(temp,
                                       axis=len(self.data.shape)-1,
                                       keepdims=True)

        t = target_indices.data.flatten()
        p = softmax_output.reshape(len(t),-1)
        target_dist = np.eye(p.shape[1])[t]
        loss = -(np.log(p) * (target_dist)).sum(1).mean()

        if(self.autograd):
            out = Tensor(loss,
                         autograd=True,
                         creators=[self],
                         creation_op="cross_entropy")
            out.softmax_output = softmax_output
            out.target_dist = target_dist
            return out

        return Tensor(loss)


    def __repr__(self):
        return str(self.data.__repr__())

    def __str__(self):
        return str(self.data.__str__())

class Layer(object):

    def __init__(self):
        self.parameters = list()

    def get_parameters(self):
        return self.parameters


class SGD(object):

    def __init__(self, parameters, alpha=0.1):
        self.parameters = parameters
        self.alpha = alpha

    def zero(self):
        for p in self.parameters:
            p.grad.data *= 0
```

```
    def step(self, zero=True):

        for p in self.parameters:

            p.data -= p.grad.data * self.alpha

            if(zero):
                p.grad.data *= 0


class Linear(Layer):

    def __init__(self, n_inputs, n_outputs, bias=True):
        super().__init__()

        self.use_bias = bias

        W = np.random.randn(n_inputs, n_outputs) *
np.sqrt(2.0/(n_inputs))
        self.weight = Tensor(W, autograd=True)
        if(self.use_bias):
            self.bias = Tensor(np.zeros(n_outputs), autograd=True)

        self.parameters.append(self.weight)

        if(self.use_bias):
            self.parameters.append(self.bias)

    def forward(self, input):
        if(self.use_bias):
            return input.mm(self.weight)
+self.bias.expand(0,len(input.data))
        return input.mm(self.weight)


class Sequential(Layer):

    def __init__(self, layers=list()):
        super().__init__()

        self.layers = layers

    def add(self, layer):
        self.layers.append(layer)

    def forward(self, input):
        for layer in self.layers:
            input = layer.forward(input)
        return input
```

```
    def get_parameters(self):
        params = list()
        for l in self.layers:
            params += l.get_parameters()
        return params


class Embedding(Layer):

    def __init__(self, vocab_size, dim):
        super().__init__()

        self.vocab_size = vocab_size
        self.dim = dim

        # this random initialiation style is just a convention from
word2vec
        self.weight = Tensor((np.random.rand(vocab_size, dim) - 0.5) /
dim, autograd=True)

        self.parameters.append(self.weight)

    def forward(self, input):
        return self.weight.index_select(input)


class Tanh(Layer):
    def __init__(self):
        super().__init__()

    def forward(self, input):
        return input.tanh()


class Sigmoid(Layer):
    def __init__(self):
        super().__init__()

    def forward(self, input):
        return input.sigmoid()


class CrossEntropyLoss(object):

    def __init__(self):
        super().__init__()

    def forward(self, input, target):
        return input.cross_entropy(target)
```

```
# Sprachmodell Long Short Term Memory
class LSTMCell(Layer):

    def __init__(self, n_inputs, n_hidden, n_output):
        super().__init__()

        self.n_inputs = n_inputs
        self.n_hidden = n_hidden
        self.n_output = n_output

        self.xf = Linear(n_inputs, n_hidden)
        self.xi = Linear(n_inputs, n_hidden)
        self.xo = Linear(n_inputs, n_hidden)
        self.xc = Linear(n_inputs, n_hidden)

        self.hf = Linear(n_hidden, n_hidden, bias=False)
        self.hi = Linear(n_hidden, n_hidden, bias=False)
        self.ho = Linear(n_hidden, n_hidden, bias=False)
        self.hc = Linear(n_hidden, n_hidden, bias=False)

        self.w_ho = Linear(n_hidden, n_output, bias=False)

        self.parameters += self.xf.get_parameters()
        self.parameters += self.xi.get_parameters()
        self.parameters += self.xo.get_parameters()
        self.parameters += self.xc.get_parameters()

        self.parameters += self.hf.get_parameters()
        self.parameters += self.hi.get_parameters()
        self.parameters += self.ho.get_parameters()
        self.parameters += self.hc.get_parameters()

        self.parameters += self.w_ho.get_parameters()

    def forward(self, input, hidden):

        prev_hidden = hidden[0]
        prev_cell = hidden[1]

        f = (self.xf.forward(input) +
self.hf.forward(prev_hidden)).sigmoid()
        i = (self.xi.forward(input) +
self.hi.forward(prev_hidden)).sigmoid()
        o = (self.xo.forward(input) +
self.ho.forward(prev_hidden)).sigmoid()
        g = (self.xc.forward(input) +
self.hc.forward(prev_hidden)).tanh()
        c = (f * prev_cell) + (i * g)
```

```
        h = o * c.tanh()

        output = self.w_ho.forward(h)
        return output, (h, c)

    def init_hidden(self, batch_size=1):
        init_hidden = Tensor(np.zeros((batch_size,self.n_hidden)),
autograd=True)
        init_cell = Tensor(np.zeros((batch_size,self.n_hidden)),
autograd=True)
        init_hidden.data[:,0] += 1
        init_cell.data[:,0] += 1
        return (init_hidden, init_cell)

import sys,random,math
from collections import Counter
import numpy as np
import sys

np.random.seed(0)

# Einlesen des VKG KORPUS
f = open('VKGKORPUS.TXT','r')
raw = f.read()
f.close()



vocab = list(set(raw))
word2index = {}
for i,word in enumerate(vocab):
    word2index[word]=i
indices = np.array(list(map(lambda x:word2index[x], raw)))

embed = Embedding(vocab_size=len(vocab),dim=512)
model = LSTMCell(n_inputs=512, n_hidden=512, n_output=len(vocab))
model.w_ho.weight.data *= 0

criterion = CrossEntropyLoss()
optim = SGD(parameters=model.get_parameters() +
embed.get_parameters(), alpha=0.05)

def generate_sample(n=30, init_char=' '):
    s = ""
    hidden = model.init_hidden(batch_size=1)
    input = Tensor(np.array([word2index[init_char]]))
    for i in range(n):
        rnn_input = embed.forward(input)
        output, hidden = model.forward(input=rnn_input, hidden=hidden)
#         output.data *= 25
```

```
#         temp_dist = output.softmax()
#         temp_dist /= temp_dist.sum()

#         m = (temp_dist > np.random.rand()).argmax()
        m = output.data.argmax()
        c = vocab[m]
        input = Tensor(np.array([m]))
        s += c
    return s

batch_size = 16
bptt = 25
n_batches = int((indices.shape[0] / (batch_size)))

trimmed_indices = indices[:n_batches*batch_size]
batched_indices = trimmed_indices.reshape(batch_size,
n_batches).transpose()

input_batched_indices = batched_indices[0:-1]
target_batched_indices = batched_indices[1:]

n_bptt = int(((n_batches-1) / bptt))
input_batches =
input_batched_indices[:n_bptt*bptt].reshape(n_bptt,bptt,batch_size)
target_batches = target_batched_indices[:n_bptt*bptt].reshape(n_bptt,
bptt, batch_size)
min_loss = 1000

# Training des neuronalen Netztes
def train(iterations=400):
    for iter in range(iterations):
        total_loss = 0
        n_loss = 0

        hidden = model.init_hidden(batch_size=batch_size)
        batches_to_train = len(input_batches)
    #     batches_to_train = 32
        for batch_i in range(batches_to_train):

            hidden = (Tensor(hidden[0].data, autograd=True),
Tensor(hidden[1].data, autograd=True))

            losses = list()
            for t in range(bptt):
                input = Tensor(input_batches[batch_i][t],
autograd=True)
                rnn_input = embed.forward(input=input)
                output, hidden = model.forward(input=rnn_input,
hidden=hidden)

                target = Tensor(target_batches[batch_i][t],
```

```
autograd=True)
                batch_loss = criterion.forward(output, target)

                if(t == 0):
                    losses.append(batch_loss)
                else:
                    losses.append(batch_loss + losses[-1])

            loss = losses[-1]

            loss.backward()
            optim.step()
            total_loss += loss.data / bptt

            epoch_loss = np.exp(total_loss / (batch_i+1))
            min_loss =1000
            if(epoch_loss < min_loss):
                min_loss = epoch_loss
                print()

            log = "\r Iter:" + str(iter)
            log += " - Alpha:" + str(optim.alpha)[0:5]
            log += " - Batch
"+str(batch_i+1)+"/"+str(len(input_batches))
            log += " - Min Loss:" + str(min_loss)[0:5]
            log += " - Loss:" + str(epoch_loss)
            if(batch_i == 0):
                log += " - " + generate_sample(n=70,
init_char='T').replace("\n"," ")
            if(batch_i % 1 == 0):
                sys.stdout.write(log)

        optim.alpha *= 0.99



train(100)

def generate_sample(n=30, init_char=' '):
    s = ""
    hidden = model.init_hidden(batch_size=1)
    input = Tensor(np.array([word2index[init_char]]))
    for i in range(n):
        rnn_input = embed.forward(input)
        output, hidden = model.forward(input=rnn_input, hidden=hidden)
        output.data *= 15
        temp_dist = output.softmax()
        temp_dist /= temp_dist.sum()
```

```
#         m = (temp_dist > np.random.rand()).argmax() # sample from
predictions
        m = output.data.argmax() # take the max prediction
        c = vocab[m]
        input = Tensor(np.array([m]))
        s += c
    return s
print(generate_sample(n=500, init_char='\n'))


print(generate_sample(n=500, init_char='\n'))
```

Ausgabe eines generierten Beispiels:

```
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
```

```
print(generate_sample(n=500, init_char=' '))
```

```
Im Gegensatz zu kognitivistischen Modellen
(ARS, Koop,P. Grammar Induction, Parser, Grammar Transduction)
erklärt ein solches Großes Sprachmodell nichts unddeshalb werden
Große Sprachmodell von Postmodernismus, Posthumanismus und
Transhumanismus
mit parasitärer Intention gefeiert.

Wenn man ein Lehrbuch über die Regeln von Verkaufsgesprächen schreiben
will,
aber einen Softwareagenten erhält, der gerne Verkaufsgespräche führt,
hat man auf sehr hohem Niveau schlechte Arbeit gemacht.
```

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 Verkaufsgespräches 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 Verkaufsgespraeche                               ;;
;;                                                              ;;
;; Die Simulation wurde ursprunglich 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                                                  ;;
;; Verabscheidung        := 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 wertizeichen 0)
     (do ((i 0 (+ i 1)))
      ( (equal? (vector-ref lexikon i) zeichen))
      (set! wertizeichen (+ 1 i))
     )
     ;;index zurueckgeben
     wertizeichen
   )

;; 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 einfügene
   (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 ursprunglich entwickelt,                               ;;
;; um die Verwendbarkeit von kontextfreien Grammatiken                         ;;
;; fuer die Algorithmisch Rekursive Sequanzanalyse                             ;;
;; zu ueberpruefen                                        ;;
;; Modellcharakter hat allein der Quelltext.                                   ;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;                                                                            ;;
;;                          VKG                                               ;;
;;    ______________________|_____________________                            ;;
;;    |                     |                     |                           ;;
;;    BG------------------->VT------------------->AV                          ;;
;;    |               _____|_____                |                           ;;
;;    |              |           |               |                           ;;
;;    |              B---------->A               |                           ;;
;;    |         _____|____       ___|_____       |                           ;;
;;    |        |         |       |       |       |                           ;;
;;    |       BBd----->BA       AE----->AA       |                           ;;
;;  __|__   ___|__    __|__   __|__   __|__   __|__                         ;;
;;  |   |   |    |    |   |   |   |   |   |   |   |                         ;;
;; KBG->VBGKBBd->VBBdKBA->VBAKAE->VAEKAA->VAAKAV-> VAV                        ;;
;;                                                                            ;;
;; 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                                     ;;
;; Verkaufstaetigkeit    := VT                                      ;;
;; Bedarfsteil           := B                                       ;;
;; Abschlussteil         := A                                       ;;
;; Begruessung           := BG                                      ;;
;; Bedarf                := Bd                                      ;;
;; Bedarfsargumentation  := BA                                      ;;
;; Abschlusseinwaende    := AE                                      ;;
;; Verkaufsabschluss     := AA                                      ;;
;; Verabscheidung        := AV                                      ;;
;; Kunde                 := vorangestelltes K                       ;;
;; Verkaeufer            := vorangestelltes V                       ;;
;;                                                                  ;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;                                                                  ;;
;; - Die Fallstruktur wird rein physikalisch protokolliert          ;;
;;   mechanisch, magnetisch, optisch oder digital D/A-Wandler       ;;
;;   (interpretationsfreies physikalisches Protokoll)               ;;
;;   z.B. Mikrophonierung, Kinematographie,                         ;;
;;   Optik, Akustik, mechanische, analoge, digitale Technik         ;;
;; - Das Protokoll wird transkribiert                               ;;
;;   (Vertextung, diskrete Ereigniskette,                           ;;
;;    Plausibilitaet, Augenscheinvalidität)                         ;;
;;   Searle, Austin: Sprechakte, Paraphrase, moegl.                 ;;
;;   Intentionen, konstitutive, konventionelle Regeln               ;;
;; - Durch Lesartenproduktion und Lesartenfalsifikation             ;;

;;   wird Sequenzstelle fuer Sequenzstelle informell                ;;
;;   das Regelsystem erzeugt                                        ;;
;;   Searle, Austin: Sprechakte, Paraphrase, moegl.                 ;;
;;   Intentionen, konstitutive, konventionelle Regeln               ;;
;;   (bei jeder Sequenzstelle werden extensiv Lesarten erzeugt,     ;;
;;    die Lesarten jeder nachfolgenden Sequenzstelle                ;;
;;    falsifizieren die Lesarten der vorausgehenden Sequenzstelle,  ;;
;;    Oevermann: Sequenzanalyse                                     ;;
;;    das Regelsystem bildet ein kontextfreies Chomskysystem,       ;;
;;    die Ersetzungsregeln sind nach Auftrittswahrscheinlichkeit    ;;
;;    gewichtet, die Interkodierreliabilitaet wird bestimmt,        ;;
;;    z.B. Mayring R, Signifikanz z.B. Chi-Quadrat)                 ;;
;; - Die Regeln werden in ein K-System uebersetzt                   ;;
;;   dabei werden die Auftrittshaeufigkeiten kumuliert              ;;
;;   um den Rechenaufwand zur Laufzeit zu minimieren                ;;
;;   Chomsky: formale Sprachen                                      ;;

;; - Auf einem Computer wird unter LISP eine Simulation gefahren    ;;
;;   McCarthy, Papert, Solomon, Bobrow, Feuerzeig
;; - Das Resultat der Simulation, eine terminale Zeichenkette,      ;;
```

```
;;   wird in ein Protokoll uebersetzt                                ;;
;; - Das künstlich erzeugte Protokoll wird auf seine Korrelation    ;;

;;   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 Verkaufsgespraeche
 ((s vkg) 100 ende)
 )
)

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;                                                                   ;;
;; Kunde teilt Bedarf mit, Verkaeufer spiegelt Bedarf Kunde          ;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;


(setq bbd
'(
 (kbbd 100 vbbd)
 )
)

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


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

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;                                                                   ;;
```

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

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

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;                                                                      ;;
;; Verkaufsabschluss                                                   ;;
;; des Abschlussteils nach den Abschlusseinwaenden                      ;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

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


;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;                                                                      ;;
;; Bedarfsteils                                                        ;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

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


;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;                                                                      ;;
;; Abschlussteil                                                       ;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;


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

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
```

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

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

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;                                                                    ;;
;; Begruessung                                                        ;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(setq bg
'(
 (kbg 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 zurück, wenn das Sequenzende ereicht 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 Verkaufsgespräches 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, KBBD, VBBD, 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;



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




(*-------------------------------------------------------------------------
---*)
  (* 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
      ELSE
       BEGIN
        PAgenda^.next := Paar;
        Paar^.next    := NIL;
        Paar^.back    := Pagenda;
        Pagenda       := Pagenda^.next;
```

```
    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;
        Wurzel.zeigt  := Wurzel.zeigt^.nach
       END;
       wurzel.spalte  := wurzel.spalte^.vor;
       wurzel.zeigt   := wurzel.spalte;
      END
     END
```

```
      END;


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

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

  PROCEDURE NimmAgendaEintrag(VAR PEintrag:PTAgenda);
   BEGIN
      IF PAgenda = Agenda
      THEN
       BEGIN
        PEintrag := Agenda;
        PAgenda  := NIL;
        Agenda   := NIL;
       END
      ELSE
       BEGIN
        PAGENDA       := PAGENDA^.back;
        PEintrag      := PAgenda^.next;
        PAGENDA^.next := NIL;
       END;
   END;




(*-------------------------------------------------------------------------
---*)
  (* Regel2EineNeueKanteAnlegen
*)

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

  PROCEDURE Regel2EineNeueKanteAnlegen( Kante     :PTKante;
                                        Kategorie :TKategorien;
                                        Gram      :TGrammatik );
   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;



(*-----------------------------------------------------------------------
---*)
  (* Regel1EineKanteErweiternen
*)

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

  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);
      Regel1EineKanteErweitern(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
```

```python
import re

# Lesen des Korpus aus einer Datei
#with open("VKGKORPUS.TXT", "r") as f:
#    korpus = f.read()
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"
# 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
BBD -> KBBD | VBBD | KBBD | VBBD | KBBD | VBBD | KBBD | VBBD
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.

```python
import re
from collections import defaultdict

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"
```

```
# Erstellen eines Wörterbuchs, um die Anzahl der Vorkommen von
Terminalsymbolden 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 -> KBBD
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 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]
    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
  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
```

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

```python
import random

# Die gegebene probabilistische Grammatik
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, 'KAV': 0.25, 'KBBD': 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, "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}
}

# 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: KBBD Agent K: KBBD 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 einfaches erweiterte Version des Programms:

```
import random

# Die gegebene probabilistische Grammatik
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, 'KAV': 0.25, 'KBBD': 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
```