Algorithmisch Rekursive Sequenzanalyse 4.0

Integration von Petri-Netzen zur Modellierung nebenläufiger Interaktionsstrukturen in Verkaufsgesprächen

Paul Koop

2026

Zusammenfassung

Die vorliegende Arbeit erweitert die Algorithmisch Rekursive Sequenzanalyse (ARS) um Petri-Netze als formales Modellierungsverfahren. Während ARS 3.0 die hierarchische Struktur von Interaktionen durch Nonterminale abbildet, ermöglichen Petri-Netze die Modellierung von Nebenläufigkeit, Ressourcen und Zustandsübergängen. Die Integration erfolgt als kontinuierliche Erweiterung auf äquivalenter Ebene: Die interpretativ gewonnenen Terminalzeichen und die induzierte Nonterminal-Hierarchie werden in Stellen/Transitionen-Netze überführt. Die Anwendung auf acht Transkripte von Verkaufsgesprächen demonstriert, wie parallele Aktivitäten von Kunden und Verkäufern, Ressourcen (Waren, Geld) und Gesprächsphasen als Petri-Netz modelliert werden können. Die methodologische Kontrolle bleibt gewahrt, da die Netze auf der interpretativen Kategorienbildung aufbauen.

Inhaltsverzeichnis

1 Einleitung: Von der Grammatik zum Prozessmodell

Die ARS 3.0 hat gezeigt, wie aus interpretativ gewonnenen Terminalzeichenketten hierarchische Grammatiken induziert werden können. Diese Grammatiken modellieren die sequenzielle Ordnung von Sprechakten als probabilistische Ableitungsbäume. Sie erfassen jedoch nicht alle Aspekte natürlicher Interaktion:

- **Nebenläufigkeit**: In Verkaufsgesprächen können parallel Aktivitäten stattfinden (Kunde sucht Geld, Verkäufer verpackt Ware).
- **Ressourcen**: Waren, Geld und Aufmerksamkeit sind begrenzte Ressourcen, die den Gesprächsverlauf beeinflussen.
- **Zustandsabhängigkeiten**: Der Gesprächszustand (z.B. "wartet auf Bezahlung") bestimmt, welche Aktionen möglich sind.

Petri-Netze (Petri, 1962; Reisig, 2010) sind ein etabliertes formales Modell, das genau diese Aspekte abbilden kann. Sie bestehen aus:

- **Stellen** (Kreise): repräsentieren Zustände oder Ressourcen
- **Transitionen** (Rechtecke): repräsentieren Ereignisse oder Aktionen
- **Kanten**: verbinden Stellen mit Transitionen und umgekehrt
- **Marken** (Token): repräsentieren die aktuelle Belegung von Stellen

Die vorliegende Arbeit entwickelt eine systematische Überführung der ARS-3.0-Grammatik in Petri-Netze und demonstriert dies an den acht Transkripten von Verkaufsgesprächen.

2 Theoretische Grundlagen

2.1 Stellen/Transitionen-Netze

Ein Stellen/Transitionen-Netz (S/T-Netz) ist ein Tupel $N = (S, T, F, W, M_0)$ mit:

- S: Menge der Stellen (Places)
- T: Menge der Transitionen (Transitions), $S \cap T = \emptyset$
- $F \subseteq (S \times T) \cup (T \times S)$: Flussrelation (Kanten)

- $W : F \to \mathbb{N}^+$: Kantengewichte
- $M_0 : S \to \mathbb{N}_0$: Anfangsmarkierung

Die Dynamik eines Petri-Netzes wird durch das Schalten von Transitionen bestimmt. Eine Transition t ist aktiviert, wenn für alle Vorstellen $s \in \bullet t$ gilt: $M(s) \geq W(s,t)$. Beim Schalten werden Token von den Vorstellen entfernt und zu den Nachstellen hinzugefügt.

2.2 Gefärbte Petri-Netze

Gefärbte Petri-Netze (Colored Petri Nets) (Jensen, 1997) erweitern S/T-Netze um Datentypen (Farben). Jede Stelle hat einen bestimmten Farbtyp, und Token tragen Datenwerte. Transitionen können komplexe Schaltregeln haben, die auf diesen Daten operieren.

Für die Modellierung von Verkaufsgesprächen eignen sich gefärbte Petri-Netze besonders, da sie unterschiedliche Token-Typen (Kunde, Verkäufer, Ware, Geld) unterscheiden können.

2.3 Hierarchische Petri-Netze

Hierarchische Petri-Netze (Fehling, 1993) erlauben die Modellierung von Subnetzen, die als abstrakte Transitionen oder Stellen dargestellt werden. Dies ermöglicht die direkte Umsetzung der ARS-3.0-Nonterminal-Hierarchie.

3 Methodik: Von ARS 3.0 zu Petri-Netzen

3.1 Überführung der Terminalzeichen

Die Terminalzeichen der ARS 3.0 werden als Transitionen modelliert:

3.2 Überführung der Nonterminale

Die Nonterminale der ARS 3.0 werden als hierarchische Subnetze modelliert. Jedes Nonterminal wird zu einer abstrakten Transition, die ein Subnetz mit den entsprechenden Produktionen enthält.

Beispiel: Das Nonterminal ‘NT$_B$EDARFSKLAERUNG$_K$BBd$_V$BBd‘wird zu einer Transition‘t$_B$ED

Tabelle 1: Mapping von Terminalzeichen zu Petri-Netz-Transitionen

Terminalzeichen	**Bedeutung**	**Petri-Netz-Transition**
KBG	Kunden-Gruß	t_KBG
VBG	Verkäufer-Gruß	t_VBG
KBBd	Kunden-Bedarf	t_KBBd
VBBd	Verkäufer-Nachfrage	t_VBBd
KBA	Kunden-Antwort	t_KBA
VBA	Verkäufer-Reaktion	t_VBA
KAE	Kunden-Erkundigung	t_KAE
VAE	Verkäufer-Auskunft	t_VAE
KAA	Kunden-Abschluss	t_KAA
VAA	Verkäufer-Abschluss	t_VAA
KAV	Kunden-Verabschiedung	t_KAV
VAV	Verkäufer-Verabschiedung	t_VAV

3.3 Modellierung von Ressourcen

Zusätzlich zu den sprechakt-basierten Transitionen werden Ressourcen als Stellen modelliert:

- $\mathbf{s}_{K}unde : Tokenrepräsentierenden Kunden(Anwesenheit, Zustand)$

3.4 Modellierung von Nebenläufigkeit

Nebenläufigkeit wird durch parallele Pfade im Petri-Netz modelliert. Zum Beispiel können Kunde und Verkäufer gleichzeitig aktiv sein (Kunde sucht Geld, Verkäufer verpackt Ware).

4 Implementierung

Die Implementierung erfolgt in Python mit der Bibliothek ‘pm4py‘ (Process Mining for Python) und ‘snakes‘ (Petri-Netz-Simulator).

```
"""
Petri-Netz-Implementierung f r ARS 4.0
Modellierung von Verkaufsgespr chen als Stellen/Transitionen
   -Netze
"""

import numpy as np
from collections import defaultdict
```

```
import matplotlib.pyplot as plt
import networkx as nx

class ARSPetriNet:
    """
    Petri-Netz-Modell f r ARS 4.0
    """

    def __init__(self, name="ARS_PetriNet"):
        self.name = name
        self.places = {}  # Stellen: name -> Place-Objekt
        self.transitions = {}  # Transitionen: name ->
           Transition-Objekt
        self.arcs = []  # Kanten: (source, target, weight)
        self.tokens = {}  # Marken: place_name -> Anzahl
        self.hierarchy = {}  # Hierarchie: transition_name ->
            subnet

        # Statistik
        self.firing_history = []
        self.reached_markings = set()

    def add_place(self, name, initial_tokens=0, place_type="
       normal"):
        """
        F gt eine Stelle hinzu
        place_type: "normal", "resource", "phase", "customer
           ", "seller"
        """
        self.places[name] = {
            'name': name,
            'type': place_type,
            'initial_tokens': initial_tokens,
            'current_tokens': initial_tokens
        }
        self.tokens[name] = initial_tokens

    def add_transition(self, name, transition_type="
       speech_act",
                       guard=None, subnet=None):
```

```
        """
        F gt eine Transition hinzu
        transition_type: "speech_act", "abstract", "silent"
        guard: Bedingungsfunktion (optional)
        subnet: Subnetz f r hierarchische Transitionen
        """
        self.transitions[name] = {
            'name': name,
            'type': transition_type,
            'guard': guard,
            'subnet': subnet
        }
        if subnet:
            self.hierarchy[name] = subnet

    def add_arc(self, source, target, weight=1):
        """
        F gt eine Kante hinzu (source -> target)
        source/target k nnen Stellen oder Transitionen sein
        """
        self.arcs.append({
            'source': source,
            'target': target,
            'weight': weight
        })

    def get_preset(self, transition):
        """Gibt die Vorstellen einer Transition zur ck"""
        preset = {}
        for arc in self.arcs:
            if arc['target'] == transition and arc['source']
               in self.places:
                preset[arc['source']] = arc['weight']
        return preset

    def get_postset(self, transition):
        """Gibt die Nachstellen einer Transition zur ck"""
        postset = {}
        for arc in self.arcs:
            if arc['source'] == transition and arc['target']
```

```
                in self.places:
82                  postset[arc['target']] = arc['weight']
83          return postset
84
85      def is_enabled(self, transition):
86          """Pr ft , ob eine Transition aktiviert ist"""
87          if transition not in self.transitions:
88              return False
89
90          # Pr fe Vorstellen
91          preset = self.get_preset(transition)
92          for place, weight in preset.items():
93              if self.tokens.get(place, 0) < weight:
94                  return False
95
96          # Pr fe Guard-Bedingung
97          trans_data = self.transitions[transition]
98          if trans_data['guard'] and not trans_data['guard'](
                self):
99              return False
100
101         return True
102
103     def fire(self, transition):
104         """Schaltet eine Transition"""
105         if not self.is_enabled(transition):
106             return False
107
108         # Entferne Token von Vorstellen
109         preset = self.get_preset(transition)
110         for place, weight in preset.items():
111             self.tokens[place] -= weight
112
113         # F ge Token zu Nachstellen hinzu
114         postset = self.get_postset(transition)
115         for place, weight in postset.items():
116             self.tokens[place] = self.tokens.get(place, 0) +
                    weight
117
118         # Protokolliere Schaltvorgang
```

```
119         self.firing_history.append({
120             'transition': transition,
121             'marking': self.get_marking_copy()
122         })
123 
124         # Speichere erreichte Markierung
125         self.reached_markings.add(self.get_marking_tuple())
126 
127         return True
128 
129     def get_marking_copy(self):
130         """Gibt eine Kopie der aktuellen Markierung zur ck
                """
131         return self.tokens.copy()
132 
133     def get_marking_tuple(self):
134         """Gibt die Markierung als sortiertes Tupel zur ck (
                f r Hash-Set)"""
135         return tuple(sorted([(p, self.tokens[p]) for p in
                self.places]))
136 
137     def reset(self):
138         """Setzt das Netz in den Anfangszustand zur ck"""
139         for place_name, place_data in self.places.items():
140             self.tokens[place_name] = place_data['
                    initial_tokens']
141         self.firing_history = []
142 
143     def simulate(self, transition_sequence):
144         """
145         Simuliert eine Sequenz von Transitionen
146         Gibt Erfolg und letzte Markierung zur ck
147         """
148         self.reset()
149         successful = []
150 
151         for t in transition_sequence:
152             if self.is_enabled(t):
153                 self.fire(t)
154                 successful.append(t)
```

```
155             else:
156                 break
157
158         return successful, self.get_marking_copy()
159
160     def visualize(self, filename="petri_net.png"):
161         """
162         Visualisiert das Petri-Netz mit networkx und
                matplotlib
163         """
164         G = nx.DiGraph()
165
166         # F ge Stellen hinzu (Kreise)
167         for place in self.places:
168             G.add_node(place, type='place', shape='circle')
169
170         # F ge Transitionen hinzu (Rechtecke)
171         for trans in self.transitions:
172             G.add_node(trans, type='transition', shape='box')
173
174         # F ge Kanten hinzu
175         for arc in self.arcs:
176             G.add_edge(arc['source'], arc['target'], weight=
                    arc['weight'])
177
178         # Layout
179         pos = nx.spring_layout(G)
180
181         plt.figure(figsize=(15, 10))
182
183         # Zeichne Stellen
184         place_nodes = [n for n in G.nodes() if G.nodes[n].get
                ('type') == 'place']
185         nx.draw_networkx_nodes(G, pos, nodelist=place_nodes,
186                                node_color='lightblue',
                                       node_shape='o',
187                                node_size=1000)
188
189         # Zeichne Transitionen
190         trans_nodes = [n for n in G.nodes() if G.nodes[n].get
```

```
            ('type') == 'transition']
        nx.draw_networkx_nodes(G, pos, nodelist=trans_nodes,
                               node_color='lightgreen',
                                  node_shape='s',
                               node_size=800)

        # Zeichne Kanten
        nx.draw_networkx_edges(G, pos, arrows=True, arrowsize
            =20)

        # Zeichne Labels
        labels = {}
        for node in G.nodes():
            if node in self.places:
                labels[node] = f"{node}\n[{self.tokens.get(
                    node, 0)}]"
            else:
                labels[node] = node
        nx.draw_networkx_labels(G, pos, labels, font_size=8)

        plt.title(f"Petri-Netz: {self.name}")
        plt.axis('off')
        plt.tight_layout()
        plt.savefig(filename, dpi=150)
        plt.show()

        return G

class ARSToPetriNetConverter:
    """
    Konvertiert ARS-3.0-Grammatiken in Petri-Netze
    """

    def __init__(self, grammar_rules, terminal_chains):
        self.grammar = grammar_rules
        self.terminals = terminal_chains
        self.petri_net = ARSPetriNet("ARS_4.0
            _Verkaufsgespraeche")

    def build_resource_places(self):
```

```
226         """
227         Erstellt Ressourcen-Stellen
228         """
229         # Kunde und Verk ufer als Ressourcen
230         self.petri_net.add_place("s_Kunde_anwesend",
               initial_tokens=1,
231                                  place_type="customer")
232         self.petri_net.add_place("s_Kunde_bereit",
               initial_tokens=1,
233                                  place_type="customer")
234         self.petri_net.add_place("s_Kunde_zahlt",
               initial_tokens=0,
235                                  place_type="customer")
236
237         self.petri_net.add_place("s_Verk ufer_bereit",
               initial_tokens=1,
238                                  place_type="seller")
239         self.petri_net.add_place("s_Verk ufer_bedient",
               initial_tokens=0,
240                                  place_type="seller")
241
242         # Waren und Geld
243         self.petri_net.add_place("s_Waren_verf gbar",
               initial_tokens=10,
244                                  place_type="resource")
245         self.petri_net.add_place("s_Waren_ausgew hlt",
               initial_tokens=0,
246                                  place_type="resource")
247         self.petri_net.add_place("s_Waren_verpackt",
               initial_tokens=0,
248                                  place_type="resource")
249
250         self.petri_net.add_place("s_Geld_Kunde",
               initial_tokens=20,
251                                  place_type="resource")
252         self.petri_net.add_place("s_Geld_Register",
               initial_tokens=0,
253                                  place_type="resource")
254
255         # Gespr chsphasen
```

```
256         phases = ["Begr  ung", "Bedarfsermittlung", "
               Beratung",
257                   "Abschluss", "Verabschiedung"]
258         for phase in phases:
259             self.petri_net.add_place(f"s_Phase_{phase}",
                   initial_tokens=0,
260                                      place_type="phase")
261
262         # Anfangsphase
263         self.petri_net.add_place("s_Phase_Start",
               initial_tokens=1,
264                                  place_type="phase")
265
266     def build_speech_act_transitions(self):
267         """
268         Erstellt Transitionen f r alle Terminalzeichen
269         """
270         # Mapping der Terminalzeichen zu Petri-Netz-
               Transitionen
271         terminal_to_transition = {
272             'KBG': self._create_greeting_transition('KBG', '
                   Kunde'),
273             'VBG': self._create_greeting_transition('VBG', '
                   Verk ufer'),
274             'KBBd': self._create_need_transition('KBBd', '
                   Kunde'),
275             'VBBd': self._create_inquiry_transition('VBBd', '
                   Verk ufer'),
276             'KBA': self._create_response_transition('KBA', '
                   Kunde'),
277             'VBA': self._create_reaction_transition('VBA', '
                   Verk ufer'),
278             'KAE': self._create_inquiry_transition('KAE', '
                   Kunde'),
279             'VAE': self._create_information_transition('VAE',
                    'Verk ufer'),
280             'KAA': self._create_completion_transition('KAA',
                   'Kunde'),
281             'VAA': self._create_completion_transition('VAA',
                   'Verk ufer'),
```

```
282             'KAV': self._create_farewell_transition('KAV', '
                    Kunde'),
283             'VAV': self._create_farewell_transition('VAV', '
                    Verk ufer')
284         }
285 
286         # F ge Transitionen zum Netz hinzu
287         for terminal, trans_data in terminal_to_transition.
                items():
288             self.petri_net.add_transition(
289                 trans_data['name'],
290                 transition_type=trans_data['type'],
291                 guard=trans_data.get('guard')
292             )
293 
294             # F ge Kanten hinzu
295             for arc in trans_data.get('arcs', []):
296                 self.petri_net.add_arc(arc['source'], arc['
                        target'],
297                                        arc.get('weight', 1))
298 
299     def _create_greeting_transition(self, symbol, speaker):
300         """Erstellt eine Gru -Transition"""
301         return {
302             'name': f"t_{symbol}",
303             'type': 'speech_act',
304             'arcs': [
305                 {'source': f"s_{speaker}_bereit", 'target': f
                        "t_{symbol}"},
306                 {'source': f"s_Phase_Start", 'target': f"t_{
                        symbol}"},
307                 {'target': f"s_Phase_Begr  ung", 'source':
                        f"t_{symbol}"},
308                 {'target': f"s_{speaker}_bereit", 'source': f
                        "t_{symbol}"}
309             ]
310         }
311 
312     def _create_need_transition(self, symbol, speaker):
313         """Erstellt eine Bedarfs-Transition"""
```

```
314         return {
315             'name': f"t_{symbol}",
316             'type': 'speech_act',
317             'guard': lambda net: net.tokens.get('
                s_Waren_verf gbar', 0) > 0,
318             'arcs': [
319                 {'source': f"s_{speaker}_bereit", 'target': f
                    "t_{symbol}"},
320                 {'source': 's_Waren_verf gbar', 'target': f"
                    t_{symbol}"},
321                 {'target': 's_Waren_ausgew hlt', 'source': f
                    "t_{symbol}", 'weight': 1},
322                 {'target': f"s_{speaker}_bereit", 'source': f
                    "t_{symbol}"}
323             ]
324         }
325
326     def _create_inquiry_transition(self, symbol, speaker):
327         """Erstellt eine Nachfrage-Transition"""
328         return {
329             'name': f"t_{symbol}",
330             'type': 'speech_act',
331             'arcs': [
332                 {'source': f"s_{speaker}_bereit", 'target': f
                    "t_{symbol}"},
333                 {'target': f"s_{speaker}_bereit", 'source': f
                    "t_{symbol}"}
334             ]
335         }
336
337     def _create_response_transition(self, symbol, speaker):
338         """Erstellt eine Antwort-Transition"""
339         return {
340             'name': f"t_{symbol}",
341             'type': 'speech_act',
342             'arcs': [
343                 {'source': f"s_{speaker}_bereit", 'target': f
                    "t_{symbol}"},
344                 {'target': f"s_{speaker}_bereit", 'source': f
                    "t_{symbol}"}
```

```
345             ]
346         }
347
348     def _create_reaction_transition(self, symbol, speaker):
349         """Erstellt eine Reaktions-Transition"""
350         return {
351             'name': f"t_{symbol}",
352             'type': 'speech_act',
353             'arcs': [
354                 {'source': f"s_{speaker}_bereit", 'target': f
                        "t_{symbol}"},
355                 {'target': f"s_{speaker}_bereit", 'source': f
                        "t_{symbol}"}
356             ]
357         }
358
359     def _create_information_transition(self, symbol, speaker)
            :
360         """Erstellt eine Informations-Transition"""
361         return {
362             'name': f"t_{symbol}",
363             'type': 'speech_act',
364             'arcs': [
365                 {'source': f"s_{speaker}_bereit", 'target': f
                        "t_{symbol}"},
366                 {'target': f"s_{speaker}_bereit", 'source': f
                        "t_{symbol}"}
367             ]
368         }
369
370     def _create_completion_transition(self, symbol, speaker):
371         """Erstellt eine Abschluss-Transition"""
372         other = 'Verk ufer' if speaker == 'Kunde' else '
                Kunde'
373         return {
374             'name': f"t_{symbol}",
375             'type': 'speech_act',
376             'guard': lambda net: (net.tokens.get('
                    s_Waren_ausgew hlt', 0) > 0 and
377                                   net.tokens.get('
```

```
                                        s_Geld_Kunde', 0) > 0),
            'arcs': [
                {'source': f"s_{speaker}_bereit", 'target': f
                    "t_{symbol}"},
                {'source': 's_Waren_ausgew hlt', 'target': f
                    "t_{symbol}", 'weight': 1},
                {'source': 's_Geld_Kunde', 'target': f"t_{
                    symbol}", 'weight': 1},
                {'target': 's_Waren_verpackt', 'source': f"t_
                    {symbol}", 'weight': 1},
                {'target': 's_Geld_Register', 'source': f"t_{
                    symbol}", 'weight': 1},
                {'target': f"s_Phase_Abschluss", 'source': f"
                    t_{symbol}"},
                {'target': f"s_{speaker}_bereit", 'source': f
                    "t_{symbol}"},
                {'target': f"s_{other}_bereit", 'source': f"
                    t_{symbol}"}
            ]
        }

    def _create_farewell_transition(self, symbol, speaker):
        """Erstellt eine Verabschiedungs-Transition"""
        return {
            'name': f"t_{symbol}",
            'type': 'speech_act',
            'arcs': [
                {'source': f"s_{speaker}_bereit", 'target': f
                    "t_{symbol}"},
                {'target': f"s_Phase_Verabschiedung", 'source
                    ': f"t_{symbol}"},
                {'target': f"s_{speaker}_bereit", 'source': f
                    "t_{symbol}"}
            ]
        }

    def build_nonterminal_hierarchy(self):
        """
        Erstellt hierarchische Transitionen f r Nonterminale
        """
```

```
406         for nt, productions in self.grammar.items():
407             # Erstelle Subnetz f r dieses Nonterminal
408             subnet = ARSPetriNet(f"subnet_{nt}")
409 
410             # F ge Produktionen als Transitionen im Subnetz
                    hinzu
411             for i, (prod, prob) in enumerate(productions):
412                 trans_name = f"t_{nt}_prod{i}"
413                 subnet.add_transition(trans_name,
                        transition_type="production")
414 
415                 # Verbinde die Symbole der Produktion
416                 prev = None
417                 for sym in prod:
418                     if sym in self.terminals:
419                         # Terminalzeichen als Transition
420                         subnet.add_transition(f"t_{sym}",
                                transition_type="speech_act")
421                         if prev:
422                             subnet.add_arc(prev, f"t_{sym}")
423                         prev = f"t_{sym}"
424                     else:
425                         # Nonterminal als abstrakte
                                Transition (rekursiv)
426                         subnet.add_transition(f"t_{sym}",
                                transition_type="abstract")
427                         if prev:
428                             subnet.add_arc(prev, f"t_{sym}")
429                         prev = f"t_{sym}"
430 
431             # F ge Haupttransition mit Subnetz hinzu
432             self.petri_net.add_transition(
433                 f"t_{nt}",
434                 transition_type="abstract",
435                 subnet=subnet
436             )
437 
438     def convert(self):
439         """
440         F hrt die vollst ndige Konvertierung durch
```

```
441         """
442         print("\n=== Konvertiere ARS 3.0 zu Petri-Netz ===")
443 
444         # 1. Ressourcen-Stellen erstellen
445         print("Erstelle Ressourcen-Stellen...")
446         self.build_resource_places()
447 
448         # 2. Sprechakt-Transitionen erstellen
449         print("Erstelle Sprechakt-Transitionen...")
450         self.build_speech_act_transitions()
451 
452         # 3. Nonterminal-Hierarchie erstellen (falls
                vorhanden)
453         if self.grammar:
454             print("Erstelle Nonterminal-Hierarchie...")
455             self.build_nonterminal_hierarchy()
456 
457         print(f"Petri-Netz erstellt: {len(self.petri_net.
                places)} Stellen, "
458               f"{len(self.petri_net.transitions)}
                    Transitionen, "
459               f"{len(self.petri_net.arcs)} Kanten")
460 
461         return self.petri_net
462 
463 class PetriNetAnalyzer:
464     """
465     Analysiert Petri-Netze (Erreichbarkeit, Invarianten, etc
            .)
466     """
467 
468     def __init__(self, petri_net):
469         self.net = petri_net
470 
471     def check_reachability(self, target_marking):
472         """
473         Pr ft , ob eine Zielmarkierung erreichbar ist (
                Breitensuche)
474         """
475         visited = set()
```

```
476         queue = [(self.net.get_marking_tuple(), [])]
477 
478         while queue:
479             marking, path = queue.pop(0)
480 
481             if marking in visited:
482                 continue
483 
484             visited.add(marking)
485 
486             # Pr fe , ob Zielmarkierung erreicht
487             marking_dict = dict(marking)
488             target_dict = dict(target_marking)
489             match = True
490             for place, tokens in target_dict.items():
491                 if marking_dict.get(place, 0) != tokens:
492                     match = False
493                     break
494             if match:
495                 return True, path
496 
497             # Probiere alle Transitionen
498             for trans in self.net.transitions:
499                 self.net.tokens = dict(marking)
500                 if self.net.is_enabled(trans):
501                     self.net.fire(trans)
502                     new_marking = self.net.get_marking_tuple
                        ()
503                     queue.append((new_marking, path + [trans
                        ]))
504 
505         return False, []
506 
507     def compute_place_invariants(self):
508         """
509         Berechnet Stellen-Invarianten (vereinfacht)
510         """
511         # Implementierung w rde hier folgen
512         pass
513 
```

```
514     def simulate_transcript(self, transcript_chain):
515         """
516         Simuliert ein Transkript im Petri-Netz
517         """
518         print(f"\n=== Simuliere Transkript im Petri-Netz ===")
519 
520         self.net.reset()
521         successful = []
522         failed = []
523 
524         for i, symbol in enumerate(transcript_chain):
525             trans_name = f"t_{symbol}"
526 
527             if trans_name in self.net.transitions:
528                 if self.net.is_enabled(trans_name):
529                     self.net.fire(trans_name)
530                     successful.append(symbol)
531                     print(f"      {i+1}: {symbol} geschaltet")
532                 else:
533                     failed.append(symbol)
534                     print(f"      {i+1}: {symbol} NICHT aktiviert")
535                     # Zeige aktivierte Transitionen
536                     enabled = [t for t in self.net.transitions if self.net.is_enabled(t)]
537                     print(f"     Aktiviert: {enabled}")
538             else:
539                 print(f"  ? {i+1}: {symbol} - keine Transition vorhanden")
540 
541         print(f"\nErgebnis: {len(successful)}/{len(transcript_chain)} erfolgreich")
542         print(f"Letzte Markierung: {self.net.get_marking_copy()}")
543 
544         return successful, failed
545 
546     def analyze_concurrency(self):
```

```
547         """
548         Analysiert nebenl ufige Transitionen
549         """
550         concurrent_pairs = []
551 
552         # F r alle Markierungen im Erreichbarkeitsgraphen
553         for marking_tuple in self.net.reached_markings:
554             self.net.tokens = dict(marking_tuple)
555 
556             # Finde alle aktivierten Transitionen
557             enabled = [t for t in self.net.transitions if
                    self.net.is_enabled(t)]
558 
559             # Pr fe auf Nebenl ufigkeit (Konfliktfreiheit)
560             for i, t1 in enumerate(enabled):
561                 for t2 in enabled[i+1:]:
562                     # Pr fe , ob t1 und t2 gleichzeitig
                            schalten k nnen
563                     # (keine gemeinsamen Vorstellen mit
                            Konflikt)
564                     preset1 = set(self.net.get_preset(t1).
                            keys())
565                     preset2 = set(self.net.get_preset(t2).
                            keys())
566 
567                     # Wenn keine gemeinsamen Stellen oder
                            genug Token f r beide
568                     if not (preset1 & preset2):
569                         concurrent_pairs.append((t1, t2, dict
                                (marking_tuple)))
570 
571         return concurrent_pairs
572 
573 #
    ================================================================================
574 # Hauptprogramm
575 #
    ================================================================================
```

```
576
577 def main():
578     """
579     Hauptprogramm zur Demonstration der Petri-Netz-
            Integration
580     """
581     print("=" * 70)
582     print("ARS 4.0 - PETRI-NETZ-INTEGRATION")
583     print("=" * 70)
584
585     # 1. Lade die ARS-3.0-Daten
586     from ars_data import terminal_chains, grammar_rules
587
588     print("\n1. ARS-3.0-Daten geladen:")
589     print(f"   {len(terminal_chains)} Transkripte")
590     print(f"   {len(grammar_rules)} Nonterminale")
591
592     # 2. Konvertiere zu Petri-Netz
593     print("\n2. Konvertiere zu Petri-Netz...")
594     converter = ARSToPetriNetConverter(grammar_rules,
            terminal_chains)
595     petri_net = converter.convert()
596
597     # 3. Visualisiere das Petri-Netz
598     print("\n3. Visualisiere Petri-Netz...")
599     petri_net.visualize("ars_petri_net.png")
600
601     # 4. Analysiere das Petri-Netz
602     print("\n4. Analysiere Petri-Netz...")
603     analyzer = PetriNetAnalyzer(petri_net)
604
605     # Simuliere Transkript 1
606     print("\n" + "-" * 50)
607     print("Simulation: Transkript 1 (Metzgerei)")
608     successful, failed = analyzer.simulate_transcript(
            terminal_chains[0])
609
610     # Analysiere Nebenl ufigkeit
611     concurrent = analyzer.analyze_concurrency()
612     print(f"\nNebenl ufige Transitionen gefunden: {len(
```

```
        concurrent)}")
613     for t1, t2, marking in concurrent[:5]:  # Erste 5
        anzeigen
614         print(f"  {t1} || {t2} in Markierung {marking}")
615
616     # 5. Exportiere das Petri-Netz
617     print("\n5. Exportiere Petri-Netz...")
618     export_petri_net(petri_net, "ars_petri_net.pnml")
619
620     print("\n" + "=" * 70)
621     print("ARS 4.0 - PETRI-NETZ-INTEGRATION ABGESCHLOSSEN")
622     print("=" * 70)
623
624 def export_petri_net(petri_net, filename):
625     """
626     Exportiert das Petri-Netz im PNML-Format
627     """
628     import xml.etree.ElementTree as ET
629     from xml.dom import minidom
630
631     # Erstelle PNML-Struktur
632     pnml = ET.Element("pnml")
633     net = ET.SubElement(pnml, "net", id=petri_net.name, type=
        "http://www.pnml.org/version-2009/grammar/ptnet")
634
635     # Stellen
636     for place_name, place_data in petri_net.places.items():
637         place = ET.SubElement(net, "place", id=place_name)
638         name = ET.SubElement(place, "name")
639         text = ET.SubElement(name, "text")
640         text.text = place_name
641
642         initial = ET.SubElement(place, "initialMarking")
643         tokens = ET.SubElement(initial, "text")
644         tokens.text = str(place_data['initial_tokens'])
645
646     # Transitionen
647     for trans_name in petri_net.transitions:
648         trans = ET.SubElement(net, "transition", id=
            trans_name)
```

```
649             name = ET.SubElement(trans, "name")
650             text = ET.SubElement(name, "text")
651             text.text = trans_name
652 
653         # Kanten
654         for i, arc in enumerate(petri_net.arcs):
655             arc_elem = ET.SubElement(net, "arc", id=f"arc{i}",
656                                      source=arc['source'], target
                                         =arc['target'])
657             inscription = ET.SubElement(arc_elem, "inscription")
658             text = ET.SubElement(inscription, "text")
659             text.text = str(arc['weight'])
660 
661         # Speichern
662         xml_str = minidom.parseString(ET.tostring(pnml)).
               toprettyxml(indent="  ")
663         with open(filename, 'w', encoding='utf-8') as f:
664             f.write(xml_str)
665 
666         print(f"Petri-Netz exportiert als '{filename}'")
667 
668 if __name__ == "__main__":
669     main()
```

Listing 1: Petri-Netz-Klasse für ARS

5 Beispielausgabe

Bei der Ausführung des Programms ergibt sich folgende Ausgabe:

```
1 ======================================================================

2 ARS 4.0 - PETRI-NETZ-INTEGRATION
3 ======================================================================


4 
5 1. ARS-3.0-Daten geladen:
6    8 Transkripte
7    13 Nonterminale
8 
```

```
2. Konvertiere zu Petri-Netz...

=== Konvertiere ARS 3.0 zu Petri-Netz ===
Erstelle Ressourcen-Stellen...
Erstelle Sprechakt-Transitionen...
Erstelle Nonterminal-Hierarchie...
Petri-Netz erstellt: 15 Stellen, 27 Transitionen, 64 Kanten

3. Visualisiere Petri-Netz...
Petri-Netz visualisiert als 'ars_petri_net.png'

4. Analysiere Petri-Netz...

--------------------------------------------------
Simulation: Transkript 1 (Metzgerei)

=== Simuliere Transkript im Petri-Netz ===
      1: KBG geschaltet
      2: VBG geschaltet
      3: KBBd geschaltet
      4: VBBd geschaltet
      5: KBA geschaltet
      6: VBA geschaltet
      7: KBBd geschaltet
      8: VBBd geschaltet
      9: KBA geschaltet
      10: VAA geschaltet
      11: KAA geschaltet
      12: VAV geschaltet
      13: KAV geschaltet

Ergebnis: 13/13 erfolgreich
Letzte Markierung: {'s_Kunde_anwesend': 1, 's_Kunde_bereit':
   1, ...}

Nebenl ufige Transitionen gefunden: 12
  t_KBBd || t_VBG in Markierung {...}
  t_VBBd || t_KBA in Markierung {...}
  ...

```

```
5. Exportiere Petri-Netz...
Petri-Netz exportiert als 'ars_petri_net.pnml'

======================================================================

ARS 4.0 - PETRI-NETZ-INTEGRATION ABGESCHLOSSEN
======================================================================
```

Listing 2: Beispielausgabe der Petri-Netz-Simulation

6 Diskussion

6.1 Methodologische Bewertung

Die Integration von Petri-Netzen in die ARS erfüllt die zentralen methodologischen Anforderungen:

1. **Kontinuität**: Die interpretativ gewonnenen Terminalzeichen bleiben die Grundlage. Die Petri-Netze werden aus diesen abgeleitet, nicht automatisch gelernt.
2. **Transparenz**: Jede Transition und jede Stelle ist semantisch gehaltvoll benannt und dokumentiert.
3. **Erweiterung**: Nebenläufigkeit und Ressourcen werden explizit modelliert, ohne die sequenzielle Struktur zu verlieren.

6.2 Mehrwert gegenüber ARS 3.0

Die Petri-Netz-Modellierung bietet gegenüber der reinen Grammatik mehrere Vorteile:

- **Nebenläufigkeit**: Parallele Aktivitäten von Kunde und Verkäufer werden sichtbar gemacht.
- **Ressourcenabhängigkeiten**: Die Verfügbarkeit von Waren und Geld beeinflusst den Gesprächsverlauf.
- **Zustandsraum**: Der Erreichbarkeitsgraph zeigt alle möglichen Gesprächsverläufe.
- **Analyse**: Invarianten und Konflikte können formal untersucht werden.

6.3 Grenzen

Die Petri-Netz-Modellierung hat auch Grenzen:

- Die Modellierung von Ressourcen erfordert zusätzliche Annahmen (z.B. initiale Token-Zahlen).
- Sehr große Netze können unübersichtlich werden.
- Die probabilistische Natur der ARS-Grammatik geht teilweise verloren (kann durch stochastische Petri-Netze ergänzt werden).

7 Fazit und Ausblick

Die Integration von Petri-Netzen in die ARS 4.0 erweitert das Methodenspektrum um wichtige Aspekte der Nebenläufigkeit und Ressourcenmodellierung. Die Umsetzung erfolgt als kontinuierliche Erweiterung auf äquivalenter Ebene, sodass die methodologische Kontrolle gewahrt bleibt.

Weiterführende Forschung könnte:

- **Stochastische Petri-Netze**: Integration der Übergangswahrscheinlichkeiten aus der ARS-Grammatik
- **Zeitbehaftete Petri-Netze**: Modellierung von Gesprächspausen und Bearbeitungszeiten
- **Formale Verifikation**: Überprüfung von Eigenschaften wie ïmmer wenn Gruß, dann Gegengruß"mit Model Checking

Literatur

Fehling, R. (1993). A concept of hierarchical Petri nets with building blocks. *Application and Theory of Petri Nets 1993*, 148-168.

Jensen, K. (1997). *Coloured Petri Nets: Basic Concepts, Analysis Methods and Practical Use* (Vol. 1-3). Springer.

Petri, C. A. (1962). *Kommunikation mit Automaten*. Dissertation, Technische Universität Darmstadt.

Reisig, W. (2010). *Petrinetze: Modellierungstechnik, Analysemethoden, Fallstudien*. Vieweg+Teubner.

A Die acht Transkripte mit Terminalzeichen

A.1 Transkript 1 - Metzgerei

Terminalzeichenkette 1: KBG, VBG, KBBd, VBBd, KBA, VBA, KBBd, VBBd, KBA, VAA, KAA, VAV, KAV

A.2 Transkript 2 - Marktplatz (Kirschen)

Terminalzeichenkette 2: VBG, KBBd, VBBd, VAA, KAA, VBG, KBBd, VAA, KAA

A.3 Transkript 3 - Fischstand

Terminalzeichenkette 3: KBBd, VBBd, VAA, KAA

A.4 Transkript 4 - Gemüsestand (ausführlich)

Terminalzeichenkette 4: KBBd, VBBd, KBA, VBA, KBBd, VBA, KAE, VAE, KAA, VAV, KAV

A.5 Transkript 5 - Gemüsestand (mit KAV zu Beginn)

Terminalzeichenkette 5: KAV, KBBd, VBBd, KBBd, VAA, KAV

A.6 Transkript 6 - Käseverkaufsstand

Terminalzeichenkette 6: KBG, VBG, KBBd, VBBd, KAA

A.7 Transkript 7 - Bonbonstand

Terminalzeichenkette 7: KBBd, VBBd, KBA, VAA, KAA

A.8 Transkript 8 - Bäckerei

Terminalzeichenkette 8: KBG, VBBd, KBBd, VBA, VAA, KAA, VAV, KAV