Explainable Recursive Interaction Analysis (ERIA)

Integration of Qualitative Sequence Analysis with Formal Modeling Using Petri Nets, Bayesian Methods and Computational Linguistic Techniques

Paul Koop

2026

Abstract

Qualitative social research faces the challenge of combining the methodological control of interpretive procedures with the precision of formal modeling. This paper develops **Explainable Recursive Interaction Analysis (ERIA)** as an integrative methodology building on the strengths of existing approaches: the hierarchical grammar induction of ARS 3.0, process modeling through Petri nets (ARS 4.0), probabilistic modeling through Bayesian methods, and the complementary use of computational linguistic methods. Unlike purely automated procedures, ERIA maintains methodological control by tracing all formal models back to interpretively derived categories. At the same time, it overcomes the sequential limitation of traditional approaches through modeling concurrency, resources, and uncertainty. The application to eight transcripts of market conversations demonstrates the power of the integrative methodology. The procedure is designated as **ERIA 1.0**.

Contents

1 Introduction: Three Methodological Traditions and Their Synthesis

The analysis of natural interactions has long been the subject of three methodological traditions that have largely existed separately:

1. **Qualitative sequence analysis** (objective hermeneutics, conversation analysis) uncovers the latent meaning structure of interactions through controlled interpretation. Its strength is the depth of understanding; its weakness is limited scalability and formalizability.

2. **Formal process modeling** (Petri nets, process calculi) allows exact modeling of concurrency, resources, and state transitions. Its strength is precision and analyzability; its weakness is the lack of connection to qualitative meaning categories.

3. **Computational linguistic modeling** (hidden Markov models, transformers, CRF) enables statistical analysis of large text corpora. Its strength is scalability; its weakness is opacity and lack of hermeneutic foundation.

Recent development of the **Algorithmic Recursive Sequence Analysis (ARS)** has built initial bridges between these traditions. ARS 3.0 introduced hierarchical grammar induction that transforms interpretively derived terminal symbols into nonterminals. ARS 4.0 extended the spectrum to include Petri nets (concurrency, resources) and Bayesian methods (uncertainty, latent variables). Additionally, hybrid integrations of computational linguistic methods (CRF, transformer embeddings, graph neural networks, attention) were developed as complementary extensions.

This paper integrates these strands into a coherent methodology, **Explainable Recursive Interaction Analysis (ERIA)**. ERIA maintains methodological control by tracing all formal models back to interpretively derived categories. At the same time, it extends this control through formal precision, analyzability, and scalability.

2 Methodological Principles of ERIA

ERIA is based on five methodological principles:

1. **Primacy of interpretation**: All formal models are derived from interpretively derived categories, not automatically induced.

2. **Multi-level integration**: Sequential structure (PCFG), concurrency (Petri nets), uncertainty (Bayesian networks), and semantic validation (transformers) are treated as complementary perspectives.

3. **Explainability by design**: Models are transparent from the ground up—every category, every state, every transition is semantically meaningful.

4. **Iterative validation**: Models are validated through comparison of empirical and generated data as well as through semantic similarity analyses.

5. **Reflexive documentation**: Every interpretive decision is logged and justified.

3 ERIA Methodology Overview

ERIA comprises six methodological steps, outlined in Table 1:

Table 1: The six steps of the ERIA methodology

Step	**Designation**	**Central methods**
1	Interpretation	Sequential microanalysis, reading production
2	Formalization	Terminal symbols, category system
3	Grammar induction	Hierarchical compression, PCFG
4	Process modeling	Petri nets (concurrency, resources)
5	Probabilistic modeling	HMM, DBN (uncertainty, latent variables)
6	Validation & triangulation	CRF, transformer embeddings, attention

3.1 Step 1: Qualitative Sequence Analysis

ERIA's foundation is a sequential microanalysis of the transcripts following the method of objective hermeneutics or the documentary method. Each speech act is analyzed regarding its sequential function and latent meaning structure.

3.2 Step 2: Formalization into Terminal Symbols

The interpretively derived categories are transformed into a system of terminal symbols:

Table 2: ERIA terminal symbols

Symbol	Meaning	Example
KBG	Customer greeting	"Good day"
VBG	Seller greeting	"Good day"
KBBd	Customer need	"Some liver sausage, please"
VBBd	Seller inquiry	"How much would you like?"
KBA	Customer response	"Two hundred grams"
VBA	Seller reaction	"Anything else?"
KAE	Customer inquiry	"Can I put these in rice salad?"
VAE	Seller information	"Better to fry briefly"
KAA	Customer completion	"Here you are", "Thanks"
VAA	Seller completion	"That will be eight marks twenty"
KAV	Customer farewell	"Goodbye"
VAV	Seller farewell	"Have a nice day"

3.3 Step 3: Hierarchical Grammar Induction (after ARS 3.0)

Terminal symbol strings are iteratively compressed to form interpretive categories (nonterminals):

```
def compress_hierarchically(chains):
    """Hierarchical compression of terminal symbol strings"""
    current_chains = [list(chain) for chain in chains]
    grammar = {}
    reflection_log = []
    iteration = 0

    while True:
        # Search for relevant pattern (with speaker change,
           closure character)
        pattern = find_relevant_pattern(current_chains)
        if pattern is None:
            break
```

```
13
14          # Generate interpretive name
15          nt_name = generate_interpretive_name(pattern)
16
17          # Document decision
18          reflection_log.append({
19              'pattern': pattern,
20              'nonterminal': nt_name,
21              'rationale': f"Repeated pattern: {'     '.join(
                    pattern)}"
22          })
23
24          # Compress chains
25          current_chains = compress_chains(current_chains,
                pattern, nt_name)
26          grammar[nt_name] = pattern
27          iteration += 1
28
29          # Check for complete compression
30          if all(len(chain) == 1 for chain in current_chains):
31              break
32
33      return grammar, current_chains, reflection_log
```

Listing 1: Hierarchical compression in ERIA

3.4 Step 4: Petri Net Modeling (after ARS 4.0, Petri nets)

The induced grammar is transformed into a Petri net that models concurrency and resources. Figure 1 shows the basic structure of the ERIA Petri net.

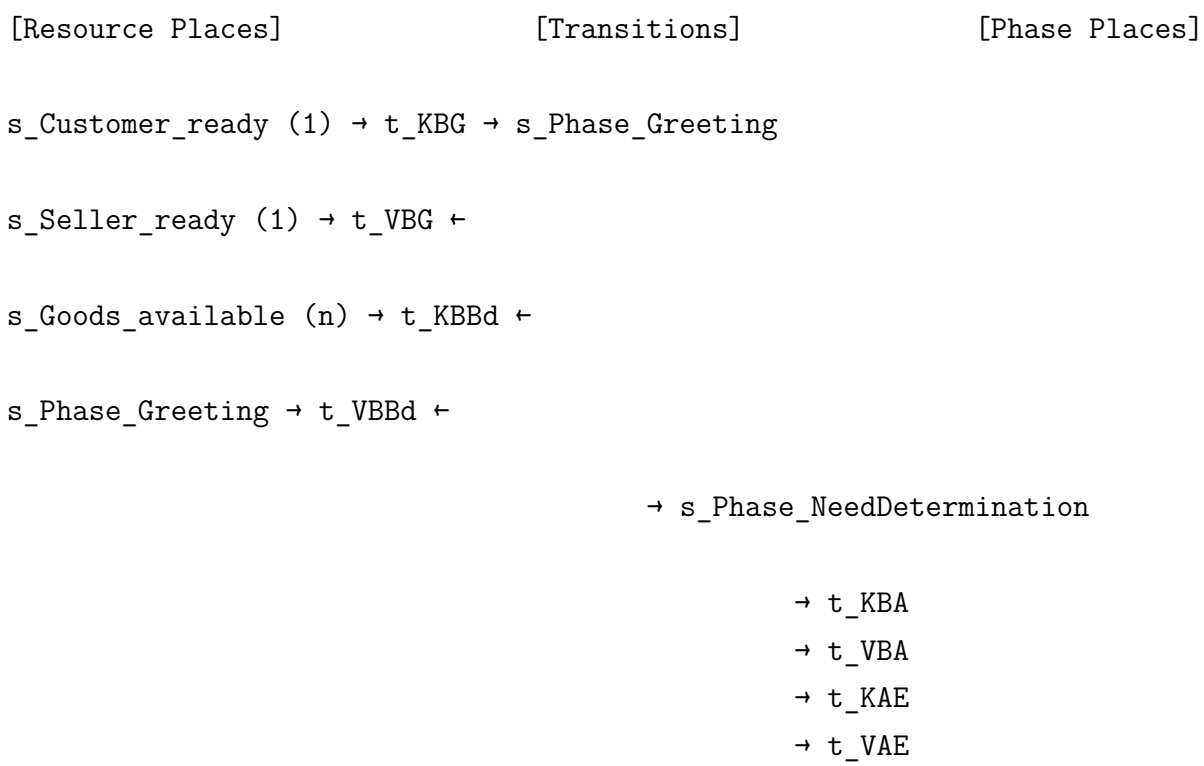

Figure 1: Basic structure of the ERIA Petri net

```python
class ERIAPetriNet:
    """Petri net for ERIA"""

    def build_from_grammar(self, grammar, terminal_chains):
        """Builds Petri net from ERIA grammar"""

        # 1. Resource places
        self.add_place("s_Customer_ready", initial_tokens=1)
        self.add_place("s_Seller_ready", initial_tokens=1)
        self.add_place("s_Goods_available", initial_tokens
            =10)
        self.add_place("s_Money_Customer", initial_tokens=20)

        # 2. Phase places
        for phase in ["Greeting", "NeedDetermination", "
            Consultation",
                      "Closing", "Farewell"]:
            self.add_place(f"s_Phase_{phase}", initial_tokens
                =0)
        self.add_place("s_Phase_Start", initial_tokens=1)

```

```
        # 3. Transitions from terminal symbols
        for terminal in self.get_all_terminals(grammar):
            self.add_transition(f"t_{terminal}")

            # Connect with resources and phases
            if terminal.startswith('K'):
                self.add_arc(f"s_Customer_ready", f"t_{
                    terminal}")
            else:
                self.add_arc(f"s_Seller_ready", f"t_{terminal
                    }")

            # Phase transitions
            phase_mapping = self.get_phase_mapping()
            if terminal in phase_mapping:
                from_phase, to_phase = phase_mapping[terminal
                    ]
                self.add_arc(f"s_Phase_{from_phase}", f"t_{
                    terminal}")
                self.add_arc(f"t_{terminal}", f"s_Phase_{
                    to_phase}")

        return self
```

Listing 2: Petri net construction in ERIA

3.5 Step 5: Bayesian Modeling (after ARS 4.0, Bayes)

ERIA uses hidden Markov models to model latent conversation phases and quantify uncertainty:

```
class ERIABayesianModel:
    """Bayesian modeling in ERIA"""

    def __init__(self, n_states=5, n_symbols=12):
        self.n_states = n_states  # Greeting,
            NeedDetermination, Consultation, Closing, Farewell
        self.n_symbols = n_symbols  # Terminal symbols
        self.state_names = {
            0: "Greeting", 1: "NeedDetermination", 2: "
                Consultation",
```

```
            3: "Closing", 4: "Farewell"
        }

    def initialize_from_ars(self, grammar):
        """Initializes HMM from ERIA grammar"""

        # Start probabilities
        startprob = np.zeros(self.n_states)
        startprob[0] = 0.7  # Greeting
        startprob[1] = 0.2  # Direct need determination
        startprob[4] = 0.1  # Direct farewell

        # Transition matrix (typical conversation flow)
        transmat = np.zeros((self.n_states, self.n_states))
        transmat[0, 1] = 0.8  # Greeting
           NeedDetermination
        transmat[1, 2] = 0.6  # NeedDetermination
           Consultation
        transmat[1, 3] = 0.3  # NeedDetermination     Closing
        transmat[2, 3] = 0.5  # Consultation     Closing
        transmat[2, 2] = 0.4  # Consultation     Consultation
        transmat[3, 4] = 0.9  # Closing     Farewell
        transmat[4, 4] = 1.0  # Farewell     Farewell

        # Emission probabilities from grammar
        emissionprob = self._compute_emissions_from_grammar(
           grammar)

        self.model = hmm.MultinomialHMM(n_components=self.
           n_states)
        self.model.startprob_ = startprob
        self.model.transmat_ = transmat
        self.model.emissionprob_ = emissionprob

        return self.model
```

Listing 3: HMM for ERIA

3.6 Step 6: Validation through Computational Linguistic Methods

ERIA uses three computational linguistic methods for complementary validation:

3.6.1 Conditional Random Fields (CRF)

CRF model sequential dependencies beyond the immediate predecessor and identify relevant contextual factors:

```
class ERIACRFValidator:
    """CRF-based validation of ERIA categories"""

    def extract_features(self, sequence, i):
        """Extracts features for position i"""
        features = {
            'symbol': sequence[i],
            'symbol.prefix_K': sequence[i].startswith('K'),
            'symbol.prefix_V': sequence[i].startswith('V'),
            'is_first': i == 0,
            'is_last': i == len(sequence) - 1,
        }

        # Context features
        for offset in [-2, -1, 1, 2]:
            if 0 <= i + offset < len(sequence):
                features[f'context_{offset:+d}'] = sequence[i
                    + offset]

        # Bigram features
        if i > 0:
            features['bigram'] = f"{sequence[i-1]}_{sequence[
               i]}"

        return features

    def validate(self, chains):
        """Validates ERIA categories through CRF training"""
        X = [[self.extract_features(seq, i) for i in range(
           len(seq))]
             for seq in chains]
```

```
        y = [seq for seq in chains]

        crf = CRF(algorithm='lbfgs', max_iterations=100)
        crf.fit(X, y)

        # Show most important features
        top_features = sorted(crf.state_features_.items(),
                              key=lambda x: abs(x[1]), reverse
                                 =True)[:10]

        return crf, top_features
```

Listing 4: CRF validation in ERIA

3.6.2 Transformer Embeddings for Semantic Validation

The semantic coherence of ERIA categories is quantified using transformer embeddings:

```
class ERIASemanticValidator:
    """Transformer-based semantic validation"""

    def __init__(self, model_name='paraphrase-multilingual-
       MiniLM-L12-v2'):
        self.model = SentenceTransformer(model_name)
        self.symbol_to_texts = {
            'KBG': ['Good day', 'Good morning', 'Hello'],
            'VBG': ['Good day', 'Good morning', 'Welcome'],
            'KBBd': ['Some liver sausage please', 'I would
               like some cheese'],
            'VBBd': ['How much would you like?', 'Which kind?
               '],
            # ... further mappings
        }

    def validate_categories(self):
        """Computes intra- and inter-category similarities"""
        embeddings = {}
        for symbol, texts in self.symbol_to_texts.items():
            emb = self.model.encode(texts)
            embeddings[symbol] = np.mean(emb, axis=0)
```

```
20
21          # Intra-category similarity (cohesion)
22          intra_similarities = {}
23          for symbol, emb in embeddings.items():
24              texts_emb = self.model.encode(self.
                    symbol_to_texts[symbol])
25              sim_matrix = cosine_similarity(texts_emb)
26              intra_similarities[symbol] = np.mean(sim_matrix[
                    np.triu_indices_from(sim_matrix, k=1)])
27
28          return intra_similarities, inter_similarities
```

Listing 5: Semantic validation in ERIA

3.6.3 Attention Mechanisms for Identifying Relevant Contexts

Attention mechanisms visualize which predecessors are particularly relevant for predicting the next symbol:

```
1  class ERIAttentionAnalyzer:
2      """Attention-based analysis of relevant contexts"""
3
4      def compute_attention_weights(self, sequence):
5          """Computes attention weights based on bigram
               statistics"""
6          n = len(sequence)
7          attention = np.zeros((n, n))
8
9          # Compute bigram probabilities
10         bigram_probs = self._compute_bigram_probs(sequence)
11
12         for i in range(1, n):
13             prev = sequence[i-1]
14             current = sequence[i]
15
16             # Attention to immediate predecessor
17             if (prev, current) in bigram_probs:
18                 attention[i, i-1] = bigram_probs[(prev,
                        current)]
19
20             # Exponentially decaying attention to more
```

```
                distant predecessors
            for j in range(i-2, -1, -1):
                attention[i, j] = attention[i, j+1] * 0.5

        # Normalization
        for i in range(n):
            if attention[i].sum() > 0:
                attention[i] /= attention[i].sum()

        return attention

    def visualize_attention(self, sequence):
        """Visualizes attention weights as heatmap"""
        attention = self.compute_attention_weights(sequence)

        plt.figure(figsize=(10, 8))
        sns.heatmap(attention,
                    xticklabels=sequence, yticklabels=sequence
                        ,
                    cmap='viridis', annot=True, fmt='.2f')
        plt.title('ERIA: Attention weights between positions'
            )
        plt.xlabel('Predecessor')
        plt.ylabel('Current position')
        plt.show()

        return attention
```

Listing 6: Attention analysis in ERIA

4 Empirical Application: Eight Market Conversations

The ERIA methodology is demonstrated on eight transcripts of market conversations (Aachen, June/July 1994).

4.1 Steps 1-2: Interpretation and Formalization

The eight transcripts were sequentially analyzed and transformed into terminal symbol strings (see Appendix A). Table 3 shows the resulting strings:

Table 3: Terminal symbol strings of the eight transcripts

Transcript	Terminal symbol string
1 (Butcher shop)	KBG, VBG, KBBd, VBBd, KBA, VBA, KBBd, VBBd, KBA, VAA, KAA, VAV
2 (Cherries)	VBG, KBBd, VBBd, VAA, KAA, VBG, KBBd, VAA, KAA
3 (Fish)	KBBd, VBBd, VAA, KAA
4 (Vegetables)	KBBd, VBBd, KBA, VBA, KBBd, VBA, KAE, VAE, KAA, VAV, KAV
5 (Vegetables 2)	KAV, KBBd, VBBd, KBBd, VAA, KAV
6 (Cheese)	KBG, VBG, KBBd, VBBd, KAA
7 (Candy)	KBBd, VBBd, KBA, VAA, KAA
8 (Bakery)	KBG, VBBd, KBBd, VBA, VAA, KAA, VAV, KAV

4.2 Step 3: Hierarchical Grammar Induction

Hierarchical compression of the terminal symbol strings led to the induction of 13 nonterminals. Table 4 shows a selection:

Table 4: Induced ERIA nonterminals

Nonterminal	Production	Interpretation
NT_GREETING	KBG → VBG	Dialogic greeting exchange
NT_NEED_DETERMINATION	KBBd → VBBd → KBA	Three-step need determination
NT_INFORMATION	KAE → VAE → KAA	Information exchange with completi
NT_CLOSING	VAA → KAA	Mutual transaction completion
NT_FAREWELL	VAV → KAV	Reciprocal farewell

4.3 Step 4: Petri Net Modeling

The Petri net derived from the grammar comprises 15 places and 27 transitions. The analysis reveals the following concurrencies:

- **Customer gets money** ‖ **Seller wraps goods**: These activities can proceed in parallel without interfering with each other.

- **Customer asks question** || **Seller prepares answer**: Parallel cognitive processes.

The resource analysis shows that the conversation stalls when place `s_Goods_available` no longer contains any tokens—a modeling result that corresponds to empirical observation.

4.4 Step 5: Bayesian Modeling

The trained HMM identifies five latent conversation phases. Table 5 shows the emission probabilities for a selected state:

Table 5: Emission probabilities for the "Consultation" state

Symbol	Probability
KAE (Customer inquiry)	0.35
VAE (Seller information)	0.35
KBA (Customer response)	0.15
VBA (Seller reaction)	0.15

Viterbi decoding for Transcript 1 yields the following state sequence:

```
KBG → VBG → KBBd → VBBd → KBA → VBA → KBBd → VBBd → KBA → VAA → KAA → VAV → KAV
 0     0      1       1      2      2      1       1      2      3      3      4
(Greeting:0, NeedDetermination:1, Consultation:2, Closing:3, Farewell:4)
```

4.5 Step 6: Validation

CRF analysis identifies the most important predictors for terminal symbols:

Table 6: Most important CRF features

Feature	Prediction	Weight
bigram:KBG_VBG	VBG	+2.345
symbol:VAA	VAV	+1.987
context_-1:VAA	KAA	+1.432
symbol.prefix_K	KBA	+1.234

Semantic validation shows high intra-category similarities (0.83-0.95), confirming the coherence of the interpretive categories.

5 Integration: From ARS 4.0 to ERIA 1.0

ERIA 1.0 integrates the three parallel-developed extensions of ARS 4.0 into a coherent methodology. Table 7 shows the assignment of methods to methodological steps:

Table 7: Integration of ARS 4.0 extensions into ERIA 1.0

ARS 4.0 extension	**ERIA step**	**Added value**
PCFG (ARS 3.0)	Step 3	Hierarchical category formation
Petri nets	Step 4	Concurrency, resources, state transitions
Bayesian networks/HMM	Step 5	Uncertainty, latent variables, inference
CRF, transformers, attention	Step 6	Validation, semantic coherence, context analysis

ERIA 1.0 is not a purely technical procedure but a methodological framework that maintains the primacy of interpretation. Formal modeling serves explication, not substitution of hermeneutic work.

6 Discussion

6.1 Methodological Assessment

ERIA fulfills the central methodological requirements of qualitative research:

1. **Transparency**: Every interpretive decision is documented; every formal model is semantically meaningful.
2. **Intersubjective traceability**: The six steps are clearly defined and can be replicated by other researchers.
3. **Reflexivity**: The methodological reflection level requires explicit justification of every decision.

4. **Triangulation**: The different formal perspectives (PCFG, Petri net, HMM, CRF, transformer) allow multidimensional validation.

6.2 Added Value Compared to Existing Approaches

ERIA offers several advantages over the original methods:

- **Compared to pure hermeneutics**: Formal modeling, traceability, scalability.
- **Compared to pure PCFG (ARS 3.0)**: Concurrency, resources, uncertainty, latent variables.
- **Compared to pure Petri nets**: Connection to interpretive categories, semantic content.
- **Compared to pure HMM**: Hierarchical structure, semantic validation, methodological control.
- **Compared to "black box" AI**: Explainability by design, no opacity.

6.3 Limitations

ERIA also has limitations that require reflection:

1. **Effort**: Sequential microanalysis is time-consuming and requires trained interpreters.
2. **Sample size**: With very large corpora ($n > 100$), manual interpretation reaches its limits.
3. **Domain specificity**: Category formation is tailored to the specific interaction domain (sales conversations).
4. **Technical dependencies**: Computational linguistic methods require pretrained models (e.g., Sentence-Transformer).

6.4 Comparison with CGTI

ERIA differs from the **Computational Grounded Theory Integration (CGTI)** in three central points:

Table 8: ERIA vs. CGTI

Criterion	**ERIA**	**CGTI**
Role of formal models	Explication of interpretive categories	Complement to hermeneutics
Petri nets	Integrated (Step 4)	Not included
Bayesian methods	Integrated (Step 5)	Not included
Computational linguistics	Validation (Step 6)	Counterfactual exploration (Phase 3)
Methodological foundation	ARS 3.0/4.0	CGTI (independent)

ERIA is formally more precise (Petri nets, HMM) and offers more comprehensive modeling of concurrency and uncertainty. CGTI is hermeneutically more conservative and foregoes formal process modeling.

7 Conclusion and Outlook

Explainable Recursive Interaction Analysis (ERIA) 1.0 integrates the strengths of three methodological traditions: the depth of qualitative sequence analysis, the precision of formal process modeling (Petri nets, HMM), and the scalability of computational linguistic methods (CRF, transformers, attention). Methodological control is maintained through the primacy of interpretation and reflexive documentation.

Future research could develop ERIA in several directions:

1. **ERIA 2.0**: Integration of large language models as counterfactual exploration tools (following CGTI, Phase 3)
2. **ERIA 3.0**: Development of a software environment to support the six steps (transcription → terminal symbols → grammar → Petri net → HMM → validation)
3. **ERIA 4.0**: Application to other interaction domains (doctor-patient conversations, classroom interactions, political debates)
4. **ERIA 5.0**: Methodological reflection on the limits of formal modeling in the social sciences

ERIA 1.0 understands itself as a contribution to **explainable qualitative research** that maintains the methodological standards of the discipline while utilizing the precision of formal methods.

References

Barredo Arrieta, A. et al. (2020). Explainable Artificial Intelligence (XAI): Concepts, taxonomies, opportunities and challenges toward responsible AI. *Information Fusion*, 58, 82-115.

Flick, U. (2019). *An Introduction to Qualitative Research* (6th ed.). Sage.

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

Lafferty, J., McCallum, A., & Pereira, F. (2001). Conditional Random Fields. *Proceedings of ICML 2001*, 282-289.

Manning, C. D., & Schütze, H. (1999). *Foundations of Statistical Natural Language Processing*. MIT Press.

Murphy, K. P. (2002). *Dynamic Bayesian Networks*. PhD Thesis, UC Berkeley.

Oevermann, U. et al. (1979). The methodology of objective hermeneutics. In H.-G. Soeffner (Ed.), *Interpretive Procedures in the Social and Textual Sciences* (pp. 352-434). Metzler.

Pearl, J. (1988). *Probabilistic Reasoning in Intelligent Systems*. Morgan Kaufmann.

Petri, C. A. (1962). *Communication with Automata*. Dissertation, TU Darmstadt.

Przyborski, A., & Wohlrab-Sahr, M. (2021). *Qualitative Social Research* (5th ed.). De Gruyter Oldenbourg.

Rabiner, L. R. (1989). A tutorial on hidden Markov models. *Proceedings of the IEEE*, 77(2), 257-286.

Reimers, N., & Gurevych, I. (2019). Sentence-BERT. *Proceedings of EMNLP-IJCNLP 2019*, 3982-3992.

Sacks, H., Schegloff, E. A., & Jefferson, G. (1974). A simplest systematics for turn-taking. *Language*, 50(4), 696-735.

Vaswani, A. et al. (2017). Attention Is All You Need. *Advances in Neural Information Processing Systems 30*, 5998-6008.

A The Eight Transcripts with Terminal Symbols

A.1 Transcript 1 - Butcher Shop

Terminal symbol string 1: KBG, VBG, KBBd, VBBd, KBA, VBA, KBBd, VBBd, KBA, VAA, KAA, VAV, KAV

A.2 Transcript 2 - Market Square (Cherries)

Terminal symbol string 2: VBG, KBBd, VBBd, VAA, KAA, VBG, KBBd, VAA, KAA

A.3 Transcript 3 - Fish Stall

Terminal symbol string 3: KBBd, VBBd, VAA, KAA

A.4 Transcript 4 - Vegetable Stall

Terminal symbol string 4: KBBd, VBBd, KBA, VBA, KBBd, VBA, KAE, VAE, KAA, VAV, KAV

A.5 Transcript 5 - Vegetable Stall 2

Terminal symbol string 5: KAV, KBBd, VBBd, KBBd, VAA, KAV

A.6 Transcript 6 - Cheese Stall

Terminal symbol string 6: KBG, VBG, KBBd, VBBd, KAA

A.7 Transcript 7 - Candy Stall

Terminal symbol string 7: KBBd, VBBd, KBA, VAA, KAA

A.8 Transcript 8 - Bakery

Terminal symbol string 8: KBG, VBBd, KBBd, VBA, VAA, KAA, VAV, KAV