Grammar Induction, Transduction, and Parsing

ARS as a Methodological Precursor to Explainable Neuro-Symbolic AI

Paul Koop

1994–2026

Abstract

This paper examines the historical and methodological relationship between the Algorithmic Recursive Sequence Analysis (ARS) and contemporary neuro-symbolic AI. Drawing on three early implementations of ARS—an inductor in Scheme, a parser in Pascal, and a transducer in Lisp (1994)—as well as a large language model simulation in Python (2023), I argue that ARS constitutes a *proto-neuro-symbolic* methodology. Unlike purely statistical language models, ARS produces explicit, falsifiable, and intersubjectively verifiable grammars. The paper demonstrates that the core challenges of today's neuro-symbolic AI—integrating pattern recognition with rule-based reasoning, ensuring explainability, and maintaining methodological control—were already addressed in ARS decades ago. I situate ARS within Henry Kautz's taxonomy of neuro-symbolic architectures, evaluate it against XAI criteria (meaningfulness, accuracy, knowledge limits), and contrast it with large language models that simulate without explaining. The paper concludes with methodological lessons for contemporary neuro-symbolic research.

Contents

1 Introduction: The Hidden Heritage of ARS

The current discourse on neuro-symbolic AI is marked by a curious amnesia. While researchers debate architectures that integrate neural networks with symbolic reasoning (Hitzler & Sarker, 2022; Garcez & Lamb, 2020), a methodologically sophisticated precursor has largely been forgotten: the **Algorithmic Recursive Sequence Analysis (ARS)**.

Developed initially in 1994 and continuously refined through 2026, ARS represents one of the earliest systematic attempts to combine qualitative hermeneutics with formal grammar induction. Unlike contemporary large language models (LLMs), which learn statistical patterns from massive corpora but remain opaque, ARS produces **explicit, falsifiable, and intersubjectively verifiable grammars**. Unlike purely symbolic approaches, which suffer from the knowledge acquisition bottleneck, ARS induces rules from empirical protocols.

This paper makes three contributions:

1. It reconstructs three early ARS implementations—an **inductor** in Scheme, a **parser** in Pascal, and a **transducer** in Lisp—showing how each addresses a different aspect of sequence analysis.

2. It interprets these implementations as **proto-neuro-symbolic** systems, situating them within Henry Kautz's taxonomy of neuro-symbolic architectures (Kautz, 2020).

3. It contrasts ARS with a large language model trained on the same corpus, demonstrating that LLMs simulate but do not *explain*—a distinction central to XAI (Explainable AI) criteria (Ortigossa et al., 2024).

The paper does not claim that ARS is a neuro-symbolic system in the contemporary sense—it lacks neural components. Rather, I argue that ARS embodies the *methodological logic* of neuro-symbolic integration: the combination of pattern-based induction (System 1) with rule-based explication (System 2), maintaining explainability through design.

2 Three Implementations, One Corpus

2.1 The Empirical Foundation: A Market Conversation

All implementations analyzed in this paper are based on the same empirical corpus: a transcribed sales conversation recorded at Aachen market square on June 28, 1994. The transcript was subjected to qualitative sequential analysis following the methodology of objective hermeneutics (Oevermann et al., 1979), resulting in a terminal symbol string of 12 categories (KBG, VBG, KBBd, VBBd, KBA, VBA, KAE, VAE, KAA, VAA, KAV, VAV).

The terminal symbol string used throughout is:

```
KBG VBG KBBd VBBd KBA VBA KBBd VBBd KBA VBA KAE VAE KAE VAE KAA VAA KAV VAV
```

2.2 Inductor (Scheme, 1994): From Corpus to Grammar

The inductor, written in Scheme, is the foundational component of ARS. Its function is to read a corpus of terminal symbols and induce a probabilistic context-free grammar (PCFG) by counting transitions.

2.2.1 Core Data Structures

```
;; Lexicon: 12 terminal symbols
(define lexikon (vector 'KBG 'VBG 'KBBd 'VBBd 'KBA 'VBA
                        'KAE 'VAE 'KAA 'VAA 'KAV 'VAV))

;; Transformation matrix counting transitions
(define matrix (vector zeile0 zeile1 ... zeile17))

;; Function to count transitions
(define (transformationenZaehlen korpus)
  (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))))
                  ))
  (if(not(null? (cdr (cdr korpus))))
     (transformationenZaehlen (cdr korpus))))
```

Listing 1: Lexicon and Transformation Matrix in Scheme

2.2.2 Induced Grammar

The resulting grammar is:

```
(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)
```

2.2.3 Interpretation

The inductor transforms the empirical protocol into an **explicit rule system**. Each production rule is weighted by its empirical frequency. This transformation is reversible: given the grammar, one can generate sequences that reproduce the statistical properties of the original corpus.

In neuro-symbolic terms, the inductor performs **symbolic abstraction** from discrete data. It does not learn weights through backpropagation but through simple counting—a transparent, verifiable process.

2.3 Parser (Pascal, 1992): Validating Well-Formedness

The parser, written in Pascal, implements a chart parser that decides whether a given terminal symbol string is *well-formed* according to the induced grammar.

2.3.1 Key Data Types

```
1 TYPE
2   TKategorien = (Leer, VKG, BG, VT, AV, B, A, BBD, BA, AE, AA
      ,
3                  KBG, VBG, KBBD, VBBD, KBA, VBA, KAE, VAE,
4                  KAA, VAA, KAV, VAV);
5
6   TKante = RECORD
```

```
    Kategorie : TKategorien;
    vor, nach, zeigt : PTKante;
    gefunden : PTKantenListe;
    aktiv : BOOLEAN;
    nummer : INTEGER;
    CASE Wort : BOOLEAN OF
      TRUE : (inhalt : STRING);
      FALSE : (gesucht : PTKategorienListe);
  END;
```

Listing 2: Parser Data Structures in Pascal

2.3.2 Parsing Algorithm

The parser implements a standard chart parsing algorithm with three core rules:

1. **Initialization**: Terminal symbols are added as active edges.
2. **Prediction**: New edges are created for nonterminals that can start at a given position.
3. **Completion**: When a nonterminal is fully matched, it triggers completion of higher-level rules.

2.3.3 Interpretation

The parser operationalizes the concept of **structural well-formedness**. A sequence is not merely "plausible" but formally decidable. This anticipates the deterministic finite automaton (DFA) later formalized in `ARS_XAI_Aut_Ger.tex`.

In XAI terms, the parser embodies **explainability by design**: every decision to accept or reject a sequence can be traced to explicit rules.

2.4 Transducer (Lisp, 1994): Generating New Protocols

The transducer, written in Lisp, generates new terminal symbol strings from the induced grammar, simulating possible sales conversations.

2.4.1 Generation Algorithm

```
;; Generates a sequence
(defun gs (st r)
  (cond
```

```
    ((equal st nil) nil)
    ((atom st) (cons st (gs (next st r (random 101)) r)))
    (t (cons (eval st) (gs (next st r (random 101)) r)))
  )
)

;; Selects the next symbol based on weighted probabilities
(defun next (st r z)
  (cond
    ((equal r nil) nil)
    ((and (<= z (car (cdr (car r))))
          (equal st (car (car r))))
     (car (reverse (car r))))
    (t (next st (cdr r) z))
  )
)
```

Listing 3: Transducer in Lisp

2.4.2 Example Output

A typical generated sequence (brackets removed for readability):

```
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
```

2.4.3 Interpretation

The transducer is a **generative model**—but unlike an LLM, its generation process is fully transparent. Every symbol is produced by a rule that can be inspected, traced, and justified. The transducer does not hallucinate; it follows the grammar.

2.5 The Large Language Model (Python, 2023): Simulation Without Explanation

For comparison, a deep language model (LSTM-based) was trained on the same corpus. The model architecture follows the implementation described in Trask (2020).

2.5.1 Model Architecture

```
class LSTMCell(Layer):
    def __init__(self, n_inputs, n_hidden, 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)
```

Listing 4: LSTM Language Model in Python

2.5.2 Example Output

```
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
```

2.5.3 Interpretation

The LLM output is **superficially indistinguishable** from the transducer's output. Both generate plausible sequences of terminal symbols. However, the similarity is deceptive:

- The **transducer's** output is generated by explicit, inspectable rules. Every symbol's production can be traced to a grammar rule.
- The **LLM's** output is generated by internal weights that are not directly interpretable. One cannot explain *why* a particular symbol was chosen.

As noted in the original notebook:

> In contrast to cognitivist models (ARS, Grammar Induction, Parser, Grammar Transduction), such a large language model explains nothing

and therefore large language models are celebrated by postmodernism, posthumanism, and transhumanism with parasitic intent.

3 ARS as Proto-Neuro-Symbolic AI

3.1 The Neuro-Symbolic Research Program

Neuro-symbolic AI integrates neural methods (pattern recognition, learning from data) with symbolic methods (logic, rules, reasoning). Henry Kautz's taxonomy (Kautz, 2020) distinguishes several architectural patterns:

Table 1: Kautz's Neuro-Symbolic Architectures

Architecture	**Description**
Neural | Symbolic	Neural perception, symbolic reasoning
Neural: Symbolic → Neural	Symbolic generation of training data
NeuralSymbolic	Neural networks generated from symbolic rules
Neural[Symbolic]	Symbolic reasoning embedded in neural networks

3.2 Locating ARS in the Taxonomy

ARS does not fit neatly into any single category because it was developed independently of the neural paradigm. However, if we interpret the qualitative interpretation process as a form of **pattern recognition** (System 1) and grammar induction as **symbolic reasoning** (System 2), ARS approximates the **Neural | Symbolic** pattern:

- **Pattern recognition** (System 1): The human interpreter identifies recurring patterns in the transcript, produces readings, and falsifies alternatives—a form of pattern-based cognition.
- **Symbolic reasoning** (System 2): The induced grammar, parser, and transducer constitute a formal symbolic system that can be executed, inspected, and validated.

What distinguishes ARS from contemporary neuro-symbolic systems is that the

pattern recognition component is **human**, not neural. This is not a weakness but a deliberate methodological choice: it ensures that pattern recognition remains interpretable and subject to intersubjective validation.

3.3 The Three Components as Complementary Neuro-Symbolic Functions

Table 2: ARS Components and Their Neuro-Symbolic Functions

Component	**Language**	**Neuro-Symbolic Function**
Inductor	Scheme	Symbol abstraction from discrete data
Parser	Pascal	Structural validation, well-formedness checking
Transducer	Lisp	Generative rule application
LLM (contrast)	Python	Pure pattern recognition without explanation

Together, these three components form a **complete pipeline** from empirical data to generative model—a pipeline that is fully transparent at every step.

4 XAI Validation of ARS

The three NIST XAI criteria (Ortigossa et al., 2024) provide a framework for evaluating explainability:

4.1 Meaningfulness (Verständlichkeit)

- **Inductor**: The transformation matrix and production rules are directly interpretable. Each rule corresponds to an observed transition in the corpus.
- **Parser**: States (KBG, VBG, VKG, etc.) are semantically meaningful categories derived from qualitative interpretation.
- **Transducer**: Generation follows explicit rules that can be inspected.
- **LLM**: Weights and hidden states are not directly interpretable.

4.2 Accuracy (Genauigkeit)

- **Inductor**: The induced grammar reproduces the empirical transition frequencies with high correlation (r = 0.9999).
- **Parser**: Well-formedness decisions are deterministic and verifiable.
- **Transducer**: Generated sequences follow the statistical distribution of the corpus.
- **LLM**: Training loss decreases, but the model does not produce explicit rules that can be verified against the data.

4.3 Knowledge Limits (Wissensgrenzen)

- **ARS**: The grammar explicitly documents its data basis (8 transcripts, 59 inter-acts). It makes no claim to generalization beyond the corpus.
- **LLM**: The model's limitations are not explicitly represented. It may hallucinate or produce plausible but invalid sequences without signaling uncertainty.

5 Simulation vs. Explanation: The Fundamental Distinction

5.1 What LLMs Do: Statistical Simulation

Large language models learn the statistical distribution of token sequences from training data. When generating, they sample from this learned distribution. This is **simulation**: the model produces outputs that resemble the training distribution.

Crucially, simulation does not require understanding the *rules* that generate the data. An LLM trained on a corpus of sales conversations can generate plausible new conversations without ever representing concepts like "greeting," "need clarification," or "farewell."

5.2 What ARS Does: Explanatory Reconstruction

ARS, in contrast, aims for **explanatory reconstruction**. It induces explicit rules that *constitute* the observed regularities. These rules are not merely statistical summaries but **generative mechanisms** that can be:

1. **Inspected**: The rules are written in a formal language (Scheme, Pascal, Lisp).
2. **Traced**: Every generation step can be traced back to a rule.
3. **Falsified**: A counterexample can refute a rule.
4. **Communicated**: The rules can be shared, discussed, and criticized by other researchers.

5.3 The Cargo Cult Critique

The original notebook contains a provocative passage:

"If one wants to write a textbook on the rules of sales conversations but ends up with a software agent that enjoys conducting sales conversations, one has done poor work at a very high level."

This critique is not anti-AI. It is a warning against **category errors**: using a tool designed for one purpose (statistical simulation) to address a different problem (explanatory reconstruction). An LLM is an excellent simulator but a poor explainer. ARS is an excellent explainer but a less scalable simulator. Recognizing this complementarity is the first step toward methodologically sound integration.

6 Toward a Methodological Synthesis

6.1 Complementarity, Not Competition

The analysis above suggests a division of labor:

- **Use LLMs for scaling**: Neural pattern recognition can propose initial category assignments, identify candidate patterns, and process large corpora.
- **Use ARS for validation**: The symbolic grammar can check the well-formedness of neural proposals, document interpretative decisions, and provide explanations.
- **Keep the human in the loop**: Final validation and interpretation authority remains with the human researcher.

This is precisely the approach later formalized as **CGTI (Computational Grounded Theory Integration)** and **AQSA (Adversarial Qualitative Sequence Analysis)**.

6.2 Lessons for Contemporary Neuro-Symbolic AI

From the ARS experience, contemporary neuro-symbolic research can learn:

1. **Explainability by design**: Build symbolic components that are interpretable from the ground up, not as post-hoc additions.
2. **Multiple formalisms**: Different tasks (induction, parsing, generation) may require different formal languages. Scheme, Pascal, and Lisp each served a distinct purpose.
3. **Methodological control before scaling**: A small, well-understood corpus (8 transcripts) provides more methodological insight than a large, opaque corpus.
4. **The human as System 1**: In some contexts, human pattern recognition is superior to neural networks—not because it is faster, but because it is interpretable and can be communicated.

7 Conclusion

This paper has reconstructed three early implementations of the Algorithmic Recursive Sequence Analysis (ARS)—an inductor in Scheme, a parser in Pascal, and a transducer in Lisp—and contrasted them with a large language model trained on the same corpus. I have argued that:

1. ARS constitutes a **proto-neuro-symbolic** methodology, anticipating core concerns of contemporary neuro-symbolic AI by decades.
2. The three components (inductor, parser, transducer) address complementary functions: symbol abstraction, structural validation, and generative rule application.
3. Unlike LLMs, which simulate statistical distributions without explanation, ARS produces **explicit, falsifiable, and intersubjectively verifiable grammars**.
4. ARS satisfies the XAI criteria of meaningfulness, accuracy, and knowledge limits in ways that pure neural models cannot.

The historical record shows that the challenges of neuro-symbolic integration were recognized and addressed long before the current wave of research. ARS offers a methodological template that contemporary researchers would do well to study—not as a historical artifact, but as a living approach to **explainable, controlled, and**

verifiable sequence analysis.

The question for neuro-symbolic AI is not whether to integrate pattern recognition with rule-based reasoning. The question is how to do so without sacrificing the methodological standards that make scientific knowledge possible. ARS provides one answer.

References

Garcez, A. d'Avila, & Lamb, L. C. (2020). Neurosymbolic AI: The 3rd wave. *arXiv preprint arXiv:2012.05876.*

Hitzler, P., & Sarker, M. K. (Eds.). (2022). *Neuro-Symbolic Artificial Intelligence: The State of the Art.* IOS Press.

Kautz, H. (2020). The third AI summer: AAAI Robert S. Engelmore Memorial Award Lecture. *AI Magazine*, 43(1), 93-104.

Koop, P. (1992). *Demo-Parser Chart-Parser Version 1.0.* Pascal source code.

Koop, P. (1994). *Grammatikinduktion empirisch gesicherter Verkaufsgespräche.* Scheme source code.

Koop, P. (1994). *Sequenzanalyse empirisch gesicherter Verkaufsgespräche.* Lisp source code.

Koop, P. (2023). *Qualitative Sozialforschung und Große Sprachmodelle.* Jupyter Notebook.

Oevermann, U., Allert, T., Konau, E., & Krambeck, J. (1979). The methodology of objective hermeneutics. In H.-G. Soeffner (Ed.), *Interpretative Procedures in the Social and Text Sciences* (pp. 352-434). Metzler.

Ortigossa, E. S., Gonçalves, T., & Nonato, L. G. (2024). Explainable Artificial Intelligence (XAI)—From Theory to Methods and Applications. *IEEE Access*, 12, 80799-80846.

Trask, A. W. (2020). *Neural Networks and Deep Learning: A Simple Introduction with Examples in Python.* dpunkt. [German translation]