Lecture 4: Rules-based chatbots

PSYC 51.07: Models of language and communication

Jeremy R. Manning
Dartmouth College
Winter 2026

Assignment 1: Q & A

Questions welcome

Let's take some time to address any questions about Assignment 1:

- Implementation challenges
- Pattern matching strategies
- Configuration file format
- Testing approaches

Remember

There are no bad questions. If you're confused about something, others probably are too!

Common issues and tips

Technical tips

- You can use raw strings for regular expressions: `r"pattern"`
- Test patterns on regex101.com or https://colab.research.google.com/
- Handle edge cases (empty input, special chars)
- Print intermediate results for debugging

Common pitfalls

- Testing on only a few inputs and missing edge cases
- Missing "special cases" like memory or goto statements
- Handling whitespace and/or punctuation inconsistently
- Matching keywords in the wrong order (instead of in descending order of rank)
- Greedy vs. non-greedy pattern matching

Beyond ELIZA...

A foundation for more

ELIZA (1966) was just the beginning! Weizenbaum's ideas inspired other researchers to test the limits of what rules-based systems could do.

Today we'll explore...

- **PARRY** (1972): a different kind of simulation
- **A.L.I.C.E.** (1995): pattern matching at scale
- **Formal grammars**: theoretical foundations of pattern matching
- **Fundamental limits** of rules-based approaches

PARRY (1972)

Further reading

Colby, Weber, & Hilf (1971, *Artificial Intelligence*): Artificial Paranoia

- Created by psychiatrist **Kenneth Colby** at Stanford
- Simulated a patient with **paranoid schizophrenia**
- Different goal than ELIZA: model a specific mental illness
- Had internal state: beliefs, emotional level, goals

Key insight

ELIZA *reflects*; PARRY *models*. ELIZA avoids commitment; PARRY has a coherent (if paranoid) worldview.

ELIZA *versus* PARRY

ELIZA (1966)

- Non-directive therapy
- Reflects user input back
- No internal state or beliefs
- Avoids making claims
- Goal: keep user talking

PARRY (1972)

- Intended to simulate "beliefs" about the world
- Tracks emotional "state" across three dimensions: anger, fear, and mistrust
- Uses internal state to select responses
- Makes paranoid claims
- Goal: behave like a paranoid schizophrenic patient

> **Historical note**
> In 1973, Vint Cerf (one of the "fathers of the Internet") used ARPANET (the precursor to the Internet) to connect ELIZA and PARRY in a text-based conversation! You can read the transcript here.

The PARRY algorithm

User input → Pattern match → Update emotions → Select response → Output

PARRY's processing pipeline with emotional state

Key difference from ELIZA

PARRY maintains a **persistent emotional state** across turns. This state influences which responses are selected, creating the illusion of coherent paranoid behavior over time.

Pattern matching: detect trigger keywords in user input

User input → **Pattern match** → Update emotions → Select response → Output

How does it work?

PARRY first scans the input for **trigger keywords** organized by topic. Each topic has associated emotional effects and response pools. This is much simpler than ELIZA's decomposition/reassembly mechanisms!

Example trigger categories

Category	Keywords	Emotional effect
Mafia/mob	"mafia", "mob", "gangster"	Fear +4, Mistrust +5
Police	"police", "cop", "arrest"	Mistrust +4, Anger +3
Trust	"trust", "believe", "honest"	Mistrust +3
Racetrack	"horses", "racing", "track"	Anger -1 (calming)

Emotional update: modify internal state

User input → Pattern match → **Update emotions** → Select response → Output

How does it work?

When a trigger pattern matches, PARRY adjusts its emotional variables. These values persist across the conversation.

```python
class Parry:
    def __init__(self):
        self.anger = 5          # 0-20 scale
        self.fear = 8           # 0-20 scale
        self.mistrust = 10      # 0-15 scale

    def process_trigger(self, topic):
        if topic == "mafia":
            self.fear += 4
            self.mistrust += 5
            self.anger += 2
```

Response selection: choose based on emotional state

User input → Pattern match → Update emotions → **Select response** → Output

How does it work?

PARRY selects responses from different pools based on current emotional thresholds. Higher emotions trigger more paranoid responses.

Response pools by emotional state

Emotional level	Response style	Example
Low (calm)	Cooperative	"I used to gamble on horses."
Medium	Guarded	"I don't want to talk about that."
High (paranoid)	Hostile	"Are you one of THEM?"

Try it out!

Use the Chatbot Evolution Demo to interact with PARRY. The "Rule Breakdown" tab illustrates the internal state changes and how they affect responses.

The Turing Test, revisited

Further reading

Colby, Hilf, Weber, & Kraemer (1972, *Artificial Intelligence*): Turing-like indistinguishability tests for the validation of a computer simulation of paranoid processes

PARRY's big test

In 1972, PARRY was tested via teletype against real patients and psychiatrists. Judges could not reliably distinguish PARRY from actual patients with paranoid schizophrenia.

- This was one of the first informal "Turing tests"
- Success? Or a comment on how we judge understanding?
- Psychiatrists were looking for *symptoms*, not *understanding*

Think about it

Does passing a specialized test mean the system understands anything? What does it mean that experts could be fooled?

A.L.I.C.E. (1995)

Further reading

A.L.I.C.E. (Artificial Linguistic Internet Computer Entity) was created by Richard Wallace.

- **AIML** (Artificial Intelligence Markup Language)
- Over **40,000 patterns** (vs ELIZA's ~200)
- Won the Loebner Prize three times (2000, 2001, 2004)
- Open source, widely studied and extended
- Very similar to ELIZA, but scaled up to a much larger rule set
- Explores the limits of pattern matching at scale

The A.L.I.C.E. algorithm

User input → Normalize → Pattern search → Extract wildcards → Process template → Response

A.L.I.C.E.'s AIML processing pipeline

Key innovation

AIML supports **recursive processing** via `<srai>` (Symbolic Reduction AI), allowing patterns to trigger other patterns. This enables handling many input variations with fewer rules.

Normalization: prepare input for matching

User input → Normalize → Pattern search → Extract wildcards → Process template → Response

How does it work?

First, A.L.I.C.E. normalizes input to uppercase and removes punctuation before pattern matching. This reduces the number of patterns needed.

Example normalization

Input	Normalized
"Don't you think so?"	"DO NOT YOU THINK SO"
"What's your name?"	"WHAT IS YOUR NAME"
"I can't believe it!"	"I CAN NOT BELIEVE IT"

Pattern search and wildcard extraction:

User input → Normalize → Pattern search → Extract wildcards → Process template → Response

How does it work?

A.L.I.C.E. matches inputs to 40,000+ ranked patterns. Patterns use wildcards (`*`) to capture variable text.

```
<category>
  <pattern>MY NAME IS *</pattern>
  <template>Nice to meet you, <star/>.</template>
</category>

<category>
  <pattern>I AM FEELING *</pattern>
  <template>Why are you feeling <star/>?</template>
</category>
...
```

Sound familiar?

This is uses essentially the same decomposition/reassembly approach as ELIZA, just in XML format with more extensive coverage.

SRAI: recursive pattern matching

User input → Normalize → Pattern search → Extract wildcards → **Process template** → Response

How does it work?

The <srai> tag redirects processing to another pattern. This allows many input variations to map to a single response. It works like goto statements in ELIZA.

```
<!-- These all redirect to the same base pattern -->
<category>
  <pattern>HI THERE</pattern>
  <template><srai>HELLO</srai></template>
</category>

<category>
  <pattern>HOWDY</pattern>
  <template><srai>HELLO</srai></template>
</category>

<category>
  <pattern>HELLO</pattern>
  <template>Hello! How can I help you today?</template>
</category>
...
```

Live demo

Try it out!

Use the Chatbot Evolution Demo to interact with A.L.I.C.E.

Explore the rules (and some additional nuances)

- Use the AIML Breakdown tab to see some additional details, like tracking the current topic, remembering the most recent response, and remembering the user's name.
- How do PARRY and A.L.I.C.E. differ from ELIZA? How are they similar?
- What do they do well?
- Where do they break down?

Formal grammars: theory behind pattern matching

Further reading

Chomsky (1956, *IRE Transactions on Information Theory*): Three Models for the Description of Language

In 1956, Noam Chomsky introduced a **hierarchy of formal grammars** that classify languages by the complexity of rules needed to generate them.

Why does this matter?

The Chomsky hierarchy tells us what kinds of patterns different computational systems can recognize—and what they *cannot*.

The Chomsky hierarchy

Type	Grammar	Recognizer	Example
Type 3	Regular	Finite automaton	`a*b+` (any number of a's followed by one or more b's)
Type 2	Context-free	Pushdown automaton	Palindromes: e.g., `racecar`
Type 1	Context-sensitive	Linear-bounded automaton	$a^n b^n c^n$
Type 0	Unrestricted	Turing machine	Any computable language: e.g., is this a valid Python program?

Key insight

Regular expressions (which ELIZA, PARRY, and A.L.I.C.E. use) are **Type 3**—the simplest class!

Remain calm...

This isn't a theory of computation course; you don't need to follow all of the details here. The key takeaways are that (a) there are *different levels of complexity* in the kinds of patterns languages can have, and that (b) regular expressions are at the *simplest* level.

Type 3 grammars

Formal definition

A **Type 3 grammar** (regular grammar) has production rules of the form:

- $S \to aA \mid bB$
- $A \to aB$
- $B \to bA$
- $A \to a$
- $A \to \varepsilon$

where A and B are non-terminal symbols, a and b are terminal symbols, and ε is the empty string (also a terminal symbol). A *terminal symbol* appears in the final output string, whereas a *non-terminal symbol* is a placeholder that can be replaced by other symbols according to the production rules. S is a special non-terminal symbol called the *start symbol*; it represents the entire string generated by the grammar.

Example Type 3 grammar: generate strings with even number of a's and b's

- $S \to aA \mid bB$
- $A \to aS \mid a$
- $B \to bS \mid b$

Challenge!

Can you write down a Type 3 grammar that generates your first name, repeated 0 or more times?

Regular expressions *are* Type 3 grammars

Mathematical equivalence

Regular expressions and Type 3 (regular) grammars are **provably equivalent**—they recognize exactly the same class of languages. In other words, for any regular expression, there exists a Type 3 grammar that generates the same language, and vice versa.

Example equivalence

Regular Expression	Type 3 Grammar	Description

`(ab)*c+`	`S → A | C`
	`A → abA | abC`
	`C → c | cC`
		Matches: "c", "cc", "abc", "ababcc"

`a(b|c)*`	`S → aA` `A → bA | cA | ε`	Matches: "a", "ab", "ac", "abbc", "acbc"

Limitation

Regular languages **cannot** match nested structures like palindromes or recursive syntax. This (among other reasons) is why rule-based chatbots struggle with complex language.

Stuff rule-based systems can't handle

- Novel situations not covered by rules
- Context that spans multiple turns
- Contextually dependent patterns
- Nested or recursive structure
- Conceptual (semantic) similarity (beyond defining equivalent keywords)
- Complex patterns (e.g., detecting whether something is a valid Python program)
- ...and more!

If rules can't fully capture human language, where do we go from here?

The harsh reality

Even with **40,000+ hand-crafted patterns**, A.L.I.C.E. cannot:

- Handle novel combinations of known concepts
- Maintain context across a conversation
- Understand implicit meaning or subtext
- Generalize beyond its training examples

The key insight

What if, instead of *writing* rules by hand, we could *learn* patterns automatically from massive amounts of text data?

Up next...

Week 2: Computational linguistics

We're leaving hand-crafted rules behind! Next week we explore how to **learn from data**:

- **Lecture 5:** Data cleaning and preprocessing
- **Lecture 6:** Tokenization—breaking text into meaningful units
- **Lecture 7:** Text classification
- **Lecture 8:** POS tagging and sentiment analysis

Hand-crafted rules → **Learning from data**

The paradigm shift that enables modern NLP

The key idea

Instead of writing rules, we'll learn to **extract patterns** from large text corpora automatically.

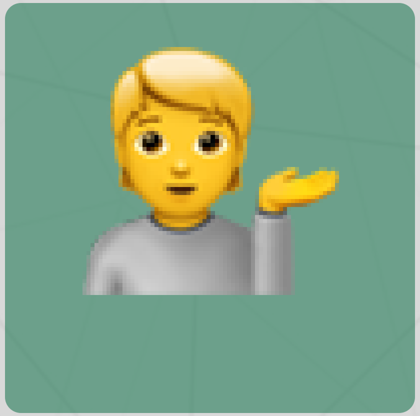
Key takeaways

1. **PARRY adds emotional state:** Pattern matching + persistent variables = coherent personality
2. **A.L.I.C.E. scales patterns:** 40,000 rules with AIML and recursive SRAI processing
3. **Chomsky hierarchy:** Regular expressions (Type 3) are the *weakest* class of grammars
4. **Fundamental limits:** Rules capture syntax, not meaning—no amount of patterns can bridge this gap

Questions? Want to chat more?

Email me

Join our Discord



Come to office hours

> **Tip**
>
> Start working on Assignment 1 now if you haven't already. Reach out early if you get stuck.