Midterm Review Quiz

💡 This is a quiz I wrote for myself to review for the midterm

- What is a physical symbol system?

  A machine that contains a collection of symbol structures, and a program which can change them over time (ie. a computer has memory, which it can change over time)

- What are the definitions of AI?

  Minsky: "The engineering of intelligent artifacts"

  Schank: "Artificial Intelligence is what me and my friends work on, and that's not anything written in FORTRAN, or anything that Noam Chomsky does"

  Pollack: "Intelligence is the asymptotic limit to mechanisms which perceive their environment, compress history into memory, and take actions to change their environment."

- How do Lisp lists work?

  They're a right-branching nil-terminated tree. Specifically, they're composed of "cons cells", which is a data structure that has a left value (car) and right value (cdr). They can be used for lots of things. In a list, the car is a value and the cdr is the next cons cell, or nil if it's the last one.

- What's the recursive definition for append?

  - If `append_to` is empty, return it.
  - Return CONS of `append_to[0]` and `append(append_to[1:], to_append)`
  - Because Recursion™ , adds each element from `append_to` to the front of `to_append` one at a time, in the right order.

  ```
  (defun append (list1 list2) (
    (if (null list1) list2
  ```

```
    (cons (car list1) (append (cdr list1) list2))
  )
)
```

▼ How to use lisp loop macros?

Basically, you can use any combination of the following:

- Iteration & Enumeration
  - `for <var> in <list>`
  - `for <var> from <start> to <end> by <step>`
  - `for <var> from <start> downto <end>`
- Counting
  - `... sum <var> | (expression)`
  - `... count <var>`
  - `... maximize <var>` / `... minimize <var>`
- List Creation
  - `... append (expression that produces a list)`
  - `... collect <var> | (expression)`
- Finding
  - `... when (predicate expression)`: only does iterations where predicate is T
  - `... thereis (predicate)`: stops at first non-NIL
  - `... always (predicate)`: stops at first NIL
  - `... never (predicate)`: stops at first non-NIL
- Exit Conditions
  - `... until (predicate expression)`
  - `... while (expression)`
- Code Execution
  - `... do (expression)`

- `... with <var> = <value>`

▼ What are the different ways to check equality in lisp?

Numbers: `=`

Symbols & Integers: `EQ`

Strings: `EQL`

Lists (deep comparison): `EQUAL`

▼ What are some classic AI puzzles?

- Tower of Hanoi: 3 posts with n disks of increasing size on the first one. The goal is to get all the rings from one post to another without ever putting a bigger disk on top of a smaller one. Takes $2^n - 1$ steps, with n^3 possible states.
- Water Jugs: you have an 8 quart jug full of water, along with one each of an empty three quart and five quart jar. How can they end up with 4 quarts of water exactly in both the 8 and 5 quart jars? 20 possible states.
- Coin Puzzle: Which n add up to X cents? Number of states is polynomial with n.
- N Queens: place N queens on an $N \times N$ chessboard such that they can't attack each other. $n!$ possible states.
- Tic Tac Toe: 3^9 possible states.
- In general: $\text{Pieces}^{\text{Positions}}$

▼ What is constraint satisfaction?

- Common technique in perception
- Hard to implement, but effective if done well
- Makes inferences that constrains the possible solution set, then plays it out. Backtracks if needed.
- This was the wireframe blocks thing.

▼ What is the "manhattan distance"?

- A simple heuristic for many puzzles.

    - Ex (tile puzzle): average of the geometric distance of each tile to its target destination

- ▼ Why is pattern matching a bad solution to NLP?

  It gets you 90% of the way there, but the last 10% requires an exponential leap in AI.

- ▼ What are the linguistic parts of NLP?
  - Morphological: syllables, prefixes/suffixes (anti-, -ing, -ed, etc.), inflections, etc. ⇒ prefix/suffix segmentation, punctation segmentation
  - Lexical: word meanings and word categories ⇒ categorization, word meaning parsing
  - Syntax: structure of language ⇒ parse a linear sequence of words to a more complex structure
  - Semantics: meaning of language ⇒ convert the output of syntactic analysis to a stored knowledge representation
  - Discourse: links multiple utterances together ⇒ track context across multiple sentences
  - Pragmatics: understand what to do based on speech

- ▼ Syntactic Parsing
  - Heart of AI
  - Issues:
    - Efficiency/Performance
    - Is this plausibly how humans do it?
    - How to handle ambiguity?
    - Top down or bottom up?
    - Integrating with other form of knowledge?
  - Very interdisciplinary topic: Linguistics, Theoretical CompSci, Comp Linguistics, AI/NLP

- Autonomy of Syntax: grammatical correctness can be determined independently from logical correctness.

▼ Chomsky Hierarchy

Groundbreaking idea.

Generative Principle: if cognition is finite but language is infinite, then there must be a finite set of rules that can produce all valid sentences. Conversely, a sentence is valid if it can be generated by the grammar.

Noam Chomsky defined three types of languages, and proved that the first two were insufficient for parsing English. Turing-equivalent languages were later added as the fourth.

1. Regular Language/Finite State Machine:
    - Can parse `aaaaab`, `ababab`
    - Conceptually, a graph where each node is a state, and each edge is a token. The system advances from one state to another by consuming the token along the edge.
    - Can't do recursion, so grows infinitely and just doesn't really work
    - Can't parse `aaabbbccc`
    - *Doesn't work for English because you can always insert arbitrary text between things, and an FSM can't do recursion: Ex:* *`Sarah called [the person [she was meeting]] up`*.
2. Context-free languages/*Phrase Structure/Transformational Language/Push Down Automata*
    - Can parse `aaabbbbbbbbbccc` (paren balancing)
    - Insufficient because it can't handle things like: `Bill and Jane like chicken and salmon, respectively`
    - A grammar is defined by a set of possible intermediary states N and a set of transformations Σ.
    - *Always starts from a starting state S ($S \in N$), then applies transformations to get to the target sentence.*

- *Much more powerful*

3. Context-sensitive languages/*Linear Bounded Automata*
    - Can do something like `he shot himself` (coordinated pronouns) or `aaabbbccc`
4. Turing machines
    - Can do something like: only strings of prime-number length

▼ What can you do with a grammar?

- Generate some or all of the sequences in a language
- Determine if a sequence of symbols is valid in the language
- Determine one/some/all valid parse trees for a sequence of symbols
    - Depth-first search is often used here with aggressive pruning
    - Alternatively, a bottom-up technique, a Chart Parser, can be used
        - Runs in $O(n^3)$ time, but very effective in practice
        - Sometimes generates too many trees/noisy
        - Less efficient than a top-down parser with memoization
        - It's the upside-down right triangle thing

▼ What is ELIZA?

- Robotherapist which just used some simple pattern matching to get people to keep talking, along with some simple state to trigger prompts
- Shockingly effective
- Doesn't scale because of, among other things, rule conflict.

▼ Formal Logic

- Abduction: Approximate inferences. Not logically sound, but good huristics
- Induction: Generalize from specific cases to general cases (ex. robins fly and crows fly, therefore all birds fly)
- Deduction: Given facts, logical rules, and laws of logic, find new fact

- **Important:** Disjunctive Syllogism: `(A | B | C) & (D | ~B | E) = (A | C | D | E)`
- DeMorgan's Laws: `~(P & Q) = (~P | ~Q)`, `~(P | Q) = (~P & ~Q)`
- Conjunctive Normal Form: `(A | B | C) & (D | E)` (series of OR groups AND'd together).

▼ How to write a parse tree in lisp?

`(Type, ...Children)`

```
(S
  (NP (NOM MARY))
  (VP
    (V ATE)
    (NP (N SPAGHETTI))
  )
)
```