CS 5600/6600: F20: Intelligent Systems
Assignment 6

Vladimir Kulyukin
Department of Computer Science
Utah State University

October 09, 2021

Learning Objectives

1. Symbolic Pattern Matching
2. Knowledge Engineering for ELIZA

Problem 1 (2 points)

Write a one-page analysis of "Computing Machinery and Intelligence" by Dr. A.M. Turing published in *Mind* in 1950. Unfortunately, I misplaced my paper copy of the original article and had to request an OCRed version from the library. It has a few OCR-related errors, but, in my opinion, is quite legible and enjoyable. There're a few things that I'd like to point out about this article for you before you read it.

Those of you who've taken or are taking my CS5000: Theory of Computability will undoubtedly and immediately recognize that Sections 4 (Digital Computers) and 5 (Universality of Digital Computers) are informally describing a finite-state automaton that came to be known in computability theory as the Turing Machine (TM). If you've never heard about TMs before, don't be afraid or discouraged – one doesn't need to know the formal definition of the TM or fundamental TM theorems to appreciate these sections of Dr. Turing's article.

Let me give you a quick and informal description of the TM. A TM is a device with a control unit defined by a finite number of rules. The control unit is operating on a tape, infinite in both directions. The control unit's reading head is reading one cell of the tape at a time. The control unit has a "program" (i.e., a finite sequence of rules). Each rule has the following form: if the TM is in state q and its head is reading symbol s on the tape, then take action a and switch to state q', where action a can be: 1) move head one cell left; 2) move head one cell right; 3) write symbol s' on the cell the TM's head is reading; or 4) erase the cell the TM's head is reading (i.e., make it blank). At the start, the TM's reading head is positioned exactly one cell left of the first cell of the input. When the TM halts (if it ever does – and it's a big if), the output of the computation is the contents of the tape. On the abstract level, the TM is designed to simulate the work of a human computer with an infinite supply of paper sheets. The word "computer" in the previous sentence is used in the sense of "accountant" or "calculator."

Dr. Turing painstakingly goes through a number of counter-arguments to his imitation game he proposes in Section 1 and to his response to the question "Can machines think?". Do your best to have an open mind when reading these sections, especially the statements in Section 6.1 (The Theological Objection). Recall that in our first lecture on the Dialogue with a Machine (see `CS5600_6600_F21_DialogueWithMachine_Part01.pdf` in Canvas) we discussed two views on Mind – dualism and materialism. Dr. Turing's views on mind and intelligence definitely belong in the

materialistic camp. Hence, some of the statements he makes in this section. In general, people tend to make all manner of statements about other cultures and religions that, upon careful investigation and deliberation, have zero basis in reality (e.g., see "Herman Melville" or "Technics and Civilization" by Lewis Mumford). But, what else is new under the sun?

I think that Sections 6.3 (The Mathematical Objection), 6.4 (The Argument from Consciousness), and 6.9 (The Argument from Extrasensory Perception) are intellectually engaging. I hope you enjoy them as much I do whenever I come back to this article. In my humble opinion, Dr. Turing fails to refute these counter-arguments. But, you're free to disagree with me, of course, and I look forward to reading what you think about his refutations.

Dr. Turing wrote this article 70 years ago. It is, in many respects, a series of scientific prophecies. For example, in Section 7 (Learning Machines), Dr. Turing predicts advances in Machine Learning (ML), outlines the theoretical structure of Reinforcement Learning (RL), and predicts research on Genetic Programming (GP).

Problem 2 (3 points)

I originally thought of assigning you the article "ELIZA–A Computer Program For the Study of Natural Language Communication Between Man and Machine" by Dr. Weizenbaum. However, we've covered all the main points of this article in the two lectures this week. I've nonetheless included it for your reference in the zip. You can read it, if you want, for your general background and better understanding or leave it for later if you're short of time.

This is a seminal symbolic AI article, because it was a direct response to Turing's Imitation Game challenge. Dr. Weizenbaum outlines a system that can hold natural language (NL) conversations with humans. The system can also be taught new things through knowledge engineering (see Section 7 in Dr. Turing's article) in that programmers/knowledge engineers can write new rules thereby teaching the system how to handle new conversational themes. This technology remains at the core of modern chatbots.

And that's exactly what we'll do in this problem – some knowledge engineering for ELIZA by writing new rules for it. You'll need to install Lisp for this problem. If you're on Linux, you can use CLISP. CLISP should be available on most Linux distributions. If it's not available on yours, do these two commands in your Linux terminal.

```
sudo apt-get update
sudo apt-get install clisp
```

And that should do it. If you're on Windows, you may want to try these in the WSL. If you're on Windows, Allegro Common Lisp (CL) from Franz, Inc. is an option. Allegro CL is a commercial distribution of Lisp, but free downscaled distributions of Allegro CL for various platforms (including Windows) are available at franz.com/downloads/clp/download. If you're new to Lisp and want to learn more about it, a pretty accessible book is at www.gigamonkeys.com/book/. A comprehensive compilation of various Lisp resources is at franz.com/newtolisp/index.lhtml. I will use Lisp and Common Lisp (CL) interchangeably in this document. CL is the modern dialect of Lisp used by researchers and developers who do symbolic AI. Guy Steele's "Common Lisp the Language, 2nd Edition" remains one of the most comprehensive references.

I assume henceforth that you've successfully installed Lisp on your machine. Let's start by loading ELIZA. The `hw06` zip contains a file `load_eliza.lisp`. The code part of the loader file looks as follows.

```
(defparameter *eliza-path*
  "/home/vladimir/teaching/AI/F21/hw/hw06/")
(defparameter *eliza-files*
  '("auxfuns.lisp" "eliza.lisp"))
(defun load-eliza (path files)
  (mapc #'(lambda (file)
            (load (concatenate 'string path file)))
        files))
```

Semicolons in Lisp introduce one line comments and multi-line comments are given between `#|` and `|#`. The first parameter, `*eliza-path*`, defines the directory where the two ELIZA files (`auxfuns.lisp` and `eliza.lisp`) are saved.

Parameters in Lisp are global variables that can be changed either by the program or by the programmer, and are always given inside the pair of maching `*`'s. This is a Lisp coding convention.

The ELIZA system consists of 2 files – `auxfns.lisp` and `eliza.lisp`. The former contains a bunch of auxiliary functions; the latter is an implementation of the system described in Weizenbaum's article.

So, change `*eliza-path*` accordingly and let's start Lisp and load ELIZA into it. Here's my output from CLISP.

```
vladimir@VladimirLaptop:~/teaching/AI/F21/hw/hw06/hw06$ clisp
  i i i i i i i       ooooo    o        ooooooo   ooooo   ooooo
  I I I I I I I      8     8   8           8     8     o  8    8
  I  \ `+' /  I      8         8           8     8        8    8
   \  `-+-'  /       8         8           8      ooooo   8oooo
    `-__|__-'        8         8           8           8  8
        |            8     o   8           8     o     8  8
  ------+------       ooooo    8oooooo  ooo8ooo   ooooo   8

Welcome to GNU CLISP 2.49.60+ (2017-06-25) <http://clisp.org/>

Copyright (c) Bruno Haible, Michael Stoll 1992, 1993
Copyright (c) Bruno Haible, Marcus Daniels 1994-1997
Copyright (c) Bruno Haible, Pierpaolo Bernardi, Sam Steingold 1998
Copyright (c) Bruno Haible, Sam Steingold 1999-2000
Copyright (c) Sam Steingold, Bruno Haible 2001-2010

Type :h and hit Enter for context help.

[1]> (load "load_eliza.lisp")
;; Loading file load_eliza.lisp ...
;; Loaded file load_eliza.lisp
#P"/home/vladimir/teaching/AI/F21/hw/hw06/hw06/load_eliza.lisp"
[2]> (load-eliza)
;; Loading file /home/vladimir/teaching/AI/F21/hw/hw06/hw06/auxfuns.lisp ...
;; Loaded file /home/vladimir/teaching/AI/F21/hw/hw06/hw06/auxfuns.lisp
;; Loading file /home/vladimir/teaching/AI/F21/hw/hw06/hw06/eliza.lisp ...
;; Loaded file /home/vladimir/teaching/AI/F21/hw/hw06/hw06/eliza.lisp
("auxfuns.lisp" "eliza.lisp")
```

I first load `load_eliza.lisp` with `(load "load_eliza.lisp")` and then call `(load-eliza)` to load

ELIZA.

The ELIZA rules are saved in `*eliza-rules*` in `eliza.lisp`. Let's evaluate this parameter in Lisp and see its value.

```
[3]> *eliza-rules*
((((?* ?X) HELLO (?* ?Y))
  (HOW DO YOU DO. PLEASE STATE YOUR PROBLEM.))
 (((?* ?X) COMPUTER (?* ?Y))
  (DO COMPUTERS WORRY YOU?)
  (WHAT DO YOU THINK ABOUT MACHINES?) (WHY DO YOU MENTION COMPUTERS?)
  (WHAT DO YOU THINK MACHINES HAVE TO DO WITH YOUR PROBLEM?))
 (((?* ?X) NAME (?* ?Y)) (I AM NOT INTERESTED IN NAMES))
 (((?* ?X) SORRY (?* ?Y)) (PLEASE DON 'T APOLOGIZE) (APOLOGIES ARE NOT NECESSARY)
  (WHAT FEELINGS DO YOU HAVE WHEN YOU APOLOGIZE))
 (((?* ?X) I REMEMBER (?* ?Y)) (DO YOU OFTEN THINK OF ?Y)
  (DOES THINKING OF ?Y BRING ANYTHING ELSE TO MIND?)
  (WHAT ELSE DO YOU REMEMBER) (WHY DO YOU RECALL ?Y RIGHT NOW?)
  (WHAT IN THE PRESENT SITUATION REMINDS YOU OF ?Y)
  (WHAT IS THE CONNECTION BETWEEN ME AND ?Y)))
```

So, the value of the variable `*eliza-rules*` is a list of rules, each of which is, in turn, a list. Let's get rule 1 from `*eliza-rules*` and save it in the variable `rule-1`. Sequence counting in Lisp starts at 0. No surprises here. So, we're really retrieving the second rule from the list of rules saved in `*eliza-rules*`.

```
[4]> (setf rule-1 (elt *eliza-rules* 1))
(((?* ?X) COMPUTER (?* ?Y))
  (DO COMPUTERS WORRY YOU?)
  (WHAT DO YOU THINK ABOUT MACHINES?)
  (WHY DO YOU MENTION COMPUTERS?)
  (WHAT DO YOU THINK MACHINES HAVE TO DO WITH YOUR PROBLEM?))
```

Rule 1 looks as follows.

```
[5]> rule-1
(((?* ?X) COMPUTER (?* ?Y))
  (DO COMPUTERS WORRY YOU?)
  (WHAT DO YOU THINK ABOUT MACHINES?)
  (WHY DO YOU MENTION COMPUTERS?)
  (WHAT DO YOU THINK MACHINES HAVE TO DO WITH YOUR PROBLEM?))
```

This rule contains the symbolic pattern `((?* ?X) COMPUTER (?* ?Y))` with two segment variables (i.e., `(?* ?X)` and `(?* ?Y)`). They are used in matching against the user input and the following four responses one of which, if the pattern matches the input, is chosen randomly (recall Dr. Turing's discussion of random number generators as a way to simulate free will in Section 4 of his article):

1. `(DO COMPUTERS WORRY YOU?)`;
2. `(WHAT DO YOU THINK ABOUT MACHINES?)`;
3. `(WHY DO YOU MENTION COMPUTERS?)`;
4. `(WHAT DO YOU THINK MACHINES HAVE TO DO WITH YOUR PROBLEM?)`.

You can access the pattern and the responses of this rule or any other rule with the functions `rule-pattern` and `rule-responses` defined in `eliza.lisp`. Here's how.

```
> (rule-pattern rule-1)
((?* ?X) COMPUTER (?* ?Y))
> (rule-responses rule-1)
((DO COMPUTERS WORRY YOU?) (WHAT DO YOU THINK ABOUT MACHINES?)
 (WHY DO YOU MENTION COMPUTERS?)
 (WHAT DO YOU THINK MACHINES HAVE TO DO WITH YOUR PROBLEM?))
```

I've defined six rules in `*eliza-rules*`. They suffice for having some conversations (not really intersting ones) with ELIZA. Simply call the function `eliza` to get into the conversation loop. Here's how.

```
[6]> (eliza)
ELIZA> hello eliza
HOW DO YOU DO. PLEASE STATE YOUR PROBLEM.
ELIZA> are you a computer
WHAT DO YOU THINK MACHINES HAVE TO DO WITH YOUR PROBLEM?
ELIZA> i don't remember having said anything about my problems
WHY DO YOU RECALL HAVING SAID ANYTHING ABOUT YOUR PROBLEMS RIGHT NOW?
ELIZA> good bye
GOOD BYE
```

As you can see above, to get out of the loop, you have to type "good bye." Also, remember that all punctuation marks are ignored in this version.

Let me mention an important aspect of Lisp debugging. If you want to fix a bug or modify your code, you don't have to re-start Lisp. You can modify your code and then load it into the same process running an instance of the Lisp machine (LM). The LM will update its symbol tables automatically. If you're new to Lisp, this may feel weird at first, because it's so much unlike what we're used to doing in imperative languages such as C/C++, Python, JAVA, etc., but you should take advantage of it. This is what makes it possible for Lispers (or should I say "Lithpers"?) to write systems that generate new code at run time, file it away, and then load it into the same running LM.

Here's an example. Let's say you want to add a rule to enable ELIZA to recognize the phrase "so long" as one of the possible ways of saying good bye. You can open the file `eliza.lisp` and type this rule right ater rule 6.

```
;;; rule 7
(((?* x) so long (?* y))
 (good bye)
 (bye)
 (see you)
 (see you later))
```

Then save `eliza.lisp` and load it into the same running LM and use the Lisp function `last` to retrieve the new rule. Note that the LM has updated the value of `*eliza-rules*`.

```
[7]> (load "eliza.lisp")
;; Loading file eliza.lisp ...
;; Loaded file eliza.lisp
```

```
#P"/home/vladimir/teaching/AI/F21/hw/hw06/hw06/eliza.lisp"
[8]> (last *eliza-rules*)
((((?* X) SO LONG (?* Y)) (GOOD BYE) (BYE) (SEE YOU) (SEE YOU LATER)))
```

You can also re-start ELIZA in the same LM process and test that this rule has been added.

```
[9]> (eliza)
ELIZA> hello eliza
HOW DO YOU DO. PLEASE STATE YOUR PROBLEM.
ELIZA> so long
SEE YOU
```

So we've just taught ELIZA a new way of saying good bye without killing the Linux process running an LM instance.

Here's your concrete assignment. Write 20 new rules for ELIZA and add them to `*eliza-rules*`. Mark the beginning and end of your rules with the Lisp comments `;` and add them at the end of `*eliza-rules*`. You can use the six rules in `eliza.lisp` as examples.

```
(defparameter *eliza-rules*
  '(
    ;;; rule 1
    (((?* ?x) hello (?* ?y))
    (How do you do.  Please state your problem.))

    ;;; rule 2
    (((?* ?x) computer (?* ?y))
     (Do computers worry you?)
     (What do you think about machines?)
     (Why do you mention computers?)
     (What do you think machines have to do with your problem?))

    ;;; rule 3
    (((?* ?x) name (?* ?y))
     (I am not interested in names))

    ;;; rule 4
    (((?* ?x) sorry (?* ?y))
     (Please don't apologize)
     (Apologies are not necessary)
     (What feelings do you have when you apologize))

    ;;; rule 5
    (((?* ?x) I remember (?* ?y))
     (Do you often think of ?y)
     (Does thinking of ?y bring anything else to mind?)
     (What else do you remember)
     (Why do you recall ?y right now?)
     (What in the present situation reminds you of ?y)
     (What is the connection between me and ?y))

    ;;; rule 6
```

```
  (((?* x) good bye (?* y))
   (good bye))

  ;;; ========== your rules begin
  ;;; add your rules here
  ;;; ========== your rules end

 ))
```

Put your best (e.g., funniest, longest, etc) conversation you had with ELIZA after enhancing its knowledge with your rules in a multi-line comment in `eliza.lisp`. As you write the rules, you may want to spend some time pondering the question of whether ELIZA can learn to construct its own rules.

If you want some inspiration, below are two online versions of ELIZA that you may interact with to figure out what kinds of rules these versions have. Great for reverse knowledge engineering, but don't get addicted!

1. www.med-ai.com/models/eliza.html;
2. www.eclecticenergies.com/ego/eliza.

What to Submit

1. `cs5600_6600_F21_hw06_paper.pdf` with your analysis of Dr. Turing's paper;
2. The `eliza.lisp` files with your rules added to `*eliza-rules*` and your best conversation you had with ELIZA in a multi-line comment.

Happy Reading, Writing, and LITHPIN'!