German University in Cairo
Department of Computer Science

Frederic-Gerald Morcos – 4-1805
Ahmad Hisham – 4-3258
Mohammed El Mehdar – 4-4608

Eliza-like Agents

- Jabberwacky
  - Created by Rollo Carpenter, is a chatter-bot created to simulate human chat in a humorous and entertaining way. It is different from other artificial intelligence software in that it learns from interacting with users and storing the conversations to be used later to find the most appropriate response (in another conversation). The purpose of the project is to create an artificial intelligence software that passes the Turing Test. George and Joan, male and female "counterparts" of Jabberwacky won the Loebner Bronze prize in 2005 and 2006 respectively.

- Eliza Test
  - This project is an exact Java implementation of the original Eliza by Joseph Weizenbaum and runs in a Java Applet [7].

- Virtual Woman
  - A software game including a chat-bot, a virtual human and artificial intelligence. When the game starts, the user can choose several properties of the 3D woman (personality, ethnicity, etc...) and then can start conversing with her. The conversations are very similar to those of Eliza. Some versions support voice input and conversations information can be downloaded from the Internet for better interactivity. Newer versions of Virtual Woman are being criticized for leading users to social isolation.

- PARRY
  - One of the early chatter-bots. Written by Kenneth Colby in 1972 simulating a paranoid schizophrenic. It supported conversational strategies and thus was much more advanced than Eliza. Parry and Eliza met several times where the most famous encounter was over ARPANET.

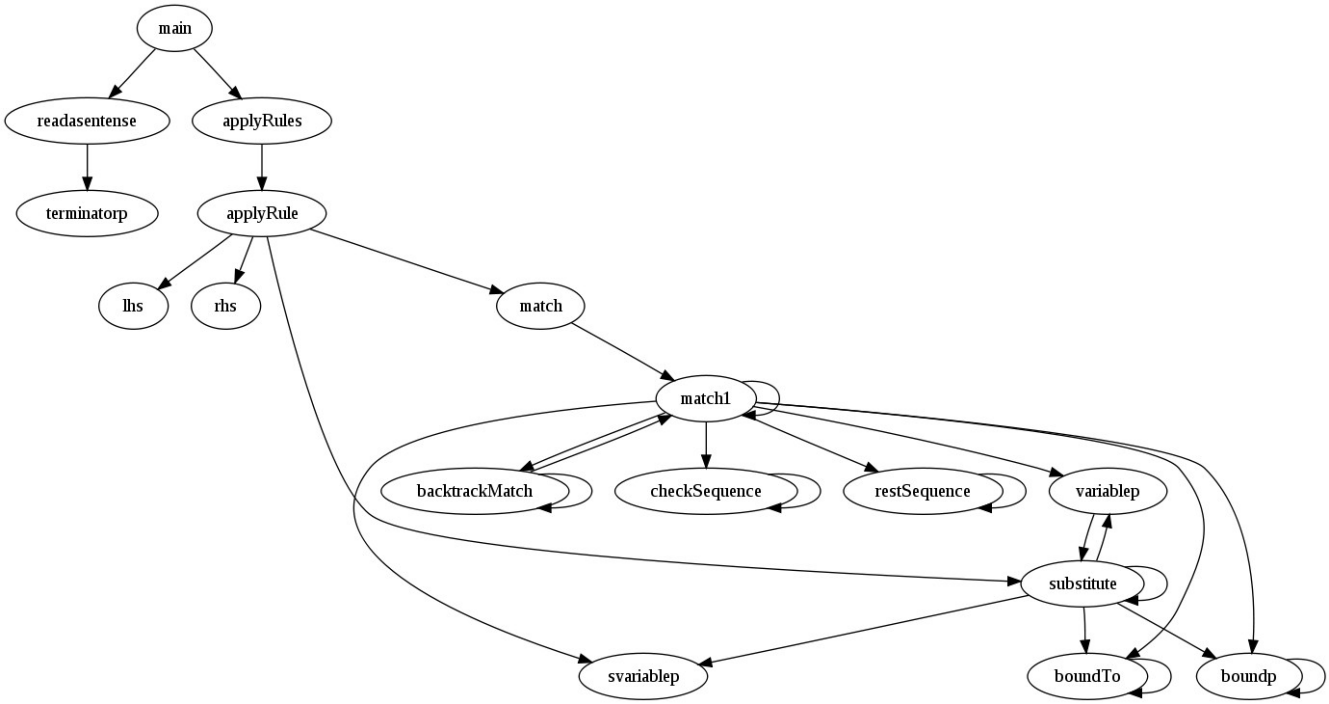
- MegaHAL
  - Is a free software (GPL) chatter-bot created by Jason Hutchens with the goal of simulating human natural language. Sometimes it outputs good coherent sentences and sometimes it outputs gibberish. Like Jabberwacky, it learns as the conversation goes and has the ability to replace older words and sentences with newer ones. It is popular for its humorous nature and known to output funny twisted and non-sense statements. MegaHAL won the Loebner Prize Contest in 1998.

- Hex
  - Also created by Jason Hutchens, is based on a database of keywords related to response sentences. The program detects trick questions. It won the Loebner Prize in 1996.

- Racter
  - A short for raconteur, is a program written by William Chamberlain and Thomas Etter. It generates random English language prose using a template system for text generation. There were claims that Racter wrote the book *The Policeman's Beard is Half Constructed.* The program was never released to the public but Mindscape, Inc. released an interactive remake which could have never written the mentioned book.

- Dr. Sbaitso
  - Is an artificial intelligence software program that is distributed with several Creative sound cards. Its name is an acronym for Sound Blaster Acting Intelligent Text to Speech Operator. It was simple with no complicated interactions while conversing with the user. Dr. Sbaitso speaks text written after the command “SAY”. Swearing from the user would “break down” the program.

General Structure of Mini-Eliza



Pattern-Matching Rules

```
(($x alike) (In what way?))
(($x something or other) (Can youu think of a specific example?))
(($x are like $y) (what resemblance do youu see?))
(($x is like $y) (what resemblance do youu see?))

(($x my $y) ($x your $y))
(($x me $y) ($x youm $y))
(($z your $x made youm $y) (your $x made youu $y))
(($x you are $y) ($x youu aree $y))
(($x you $y) ($x youi $y))
((youu aree $x but $y) (what makes you think ii amm $x))
```

```
((($x I $y) ($x you $y))
(($x am $y) ($x are $y))
((you need $x that $y) (what would it mean to you if you got $x))
((you need $x) (what would it mean to you if you got $x))

(($x you $y learn $z your mother) (tell me more about your family))
(($x you $y learn $z your father) (tell me more about your family))
(($x you $y learn $z your brother) (tell me more about your family))
(($x you $y learn $z your sister) (tell me more about your family))

((your mother $x of youm) (who else in your family $x of you?))
((your brother $x of youm) (who else in your family $x of you?))
((your father $x of youm) (who else in your family $x of you?))
((your sister $x of youm) (who else in your family $x of you?))

((he says you are $x much of the time) (i am sorry to hear you are $x))
((he says you are $x most of the time) (i am sorry to hear you are $x))
(($x you are unhappy $y) (do you think coming here will help you not to be
unhappy?))

((youi $x with youm $y) (why do youu think ii $x with you))
((youu aree $x youm) (does it please you to believe ii amm $x you))
((your $x is $y) (what else comes to mind when you think of your $x))
((bullies) (does that have anything to do with the fact that you are here))

(($x but $y) ($x))
(($x and $y) ($x))
(($x deadlines  $y) (do deadlines make you worry))
(($x deadline $y) (is it hard to cope up with all the deadlines))
(($x you are tired of $y but $z) (why are youu tired of $y))
(($x you are tired of $y) (why are youu tired of $y))
(($x lisp is $y)(how is lisp $y))
(($x takes all your energy $y) (do you think you would do better off without $x))
(($x you worry about $y) (is $y important to you))
(($x your $y make youm feel like you have to $z) (do you care what your $y make
you feel))
(($x sometimes $y) (when was the last time?))
(($x youi think $z guc $y) (ii amm a guc counselor so i am not allowed to comment
on that))
((what do $z think $y) (ii amm not sure what $z think))
((what does $z think $y) (ii amm not sure what $z thinks))
(($x yes you $y do $z) (then its probably true))
(($x you have to go now $y) (good luck then))
```

The syntax for our pattern-matching rules are as fellows:

```
(($x some-input to be recognized $y) ($x some-output $y))
```

The first parameter in the parenthesis is the input, it indicates that if the 'some-input to be recognized' is found between two sequence variables $x $y (either could be NIL i.e. not exist) somewhere in the input-line from the user, this rule will output '$x some-output $y' .. apply-rules will then take this output and match it with the other rules as well, which leads to a problem:

```
(1)       (($x I $y) ($x you $y))
(2)       (($x am $y) ($x are $y))
```

Since our apply-rules works on a rule-list recursively starting in the order they were written, suppose we need a rule to recognize when the user says something like 'I need some help' ?

```
((I need $x) (what would it mean to you if you got $x))
```

However, if this rule occurs after the first rule , the 'I' would have already been replaced by 'you', so we modify this to:

```
(3)       ((you need $x) (what would it mean to you if you got $x))
```

This leads to another problem though, what if the user says something like 'you need help' ? The rule will treat it the same as if the user said 'i need help'. The way we solve this is by defining a second (final) set of rules to be applied after the initial ones, and then we define the first rules in our initial ruleset as fellows:

```
(4)       (($x you are $y) ($x youu aree $y))
(5)       (($x you $y) ($x youi $y))
(6)       (($x my $y) ($x your $y))
(7)       (($x me $y) ($x youm $y))
```

What this does is insure that if a 'you' that resulted from applying rule (1) on an 'I' is different then a 'you' which was entered by the user, if the user input contains 'I' it'll be transformed to you and all the subsequent rules which are meant to recognize this 'I' from the user will work by recognizing 'you' from the output of the first few rules. If a rule is meant to recognize a 'you' from the user it will set out to recognize 'youu' , like this:

```
(8)       ((youu aree $x but $y) (what makes you think ii amm $x))
```

Bearing in mind this rule must come after rule no. (4), this prevents this rule from being matched to something like "I am okay but sad " from the user, it will be matched only to sentences "you are 'something but 'something". notice that the output is also modified (instead of 'i am' ii amm), this is because otherwise the output 'i am' will go through rules (1),(2) again (due to recursiveness of the function call) and transform to 'you are'.

In the end, we define all our conversation rules in the first rule-set according to this methodology, the only thing left then is to move back from our modified symbols 'youu, ii' to the output we require, this will be done in the second rule-set with rules like:

```
(defparameter *secondary-rules*
     '(
     (($x youu $y) ($x you $y))
     (($x youm $y) ($x you $y))
     (($x youi $y) ($x i $y))
     (($x aree $y) ($x are $y))
     (($x ii $y) ($x i $y))
     (($x amm $y) ($x am $y))
     )
)
```

Recall our function call was :

```
(format t "~{~A ~}" (apply-rules (apply-rules input *rules*) *secondary-rules*))
```

So the secondary rules will be applied to the output of the initial rule set.

Mechanisms for Regeneration of an Eliza Dialogue

Comments in the source code files “match.lisp” and “eliza.lisp” along with the explanations of the pattern matching rules in the previous section of the report are sufficient enough to explain how the mechanisms used are enough to regenerate the Eliza dialogue in Weizenbaum’s paper. The next section shows a sample dialogue with Mini-Eliza similar to the one in Weizenbaum’s paper.

Sample Session with Mini-Eliza

```
men are all alike.
IN WHAT WAY?
they are always bugging us about something or other.
CAN YOU THINK OF A SPECIFIC EXAMPLE?
well my boyfriend made me come here.
YOUR BOYFRIEND MADE YOU COME HERE
he says i am depressed much of the time.
I AM SORRY TO HEAR YOU ARE DEPRESSED
it is true i am unhappy.
DO YOU THINK COMING HERE WILL HELP YOU NOT TO BE UNHAPPY?
i need some help that much seems certain.
WHAT WOULD IT MEAN TO YOU IF YOU GOT SOME HELP
perhaps i could learn to get along with my mother.
TELL ME MORE ABOUT YOUR FAMILY
my mother takes care of me.
WHO ELSE IN YOUR FAMILY TAKES CARE OF YOU?
my father.
YOUR FATHER
you are like my father in some ways.
WHAT RESEMBLANCE DO YOU SEE?
you are not very aggressive but i think you dont want me to notice that.
WHAT MAKES YOU THINK I AM NOT VERY AGGRESSIVE
you dont argue with me.
WHY DO YOU THINK I DONT ARGUE WITH YOU
you are afraid of me.
DOES IT PLEASE YOU TO BELIEVE I AM AFRAID OF YOU
my father is afraid of everybody.
WHAT ELSE COMES TO MIND WHEN YOU THINK OF YOUR FATHER
bullies.
DOES THAT HAVE ANYTHING TO DO WITH THE FACT THAT YOU ARE HERE
bye.
GOOD-BYE
```

```
i am tired of all the lisp but i feel satisfied when it works.
WHY ARE YOU TIRED OF ALL THE LISP
well lisp is awful.
HOW IS LISP AWFUL
it takes all your energy.
```

```
DO YOU THINK YOU WOULD DO BETTER OFF WITHOUT IT
it would be easier to keep up with all the deadlines.
DO DEADLINES MAKE YOU WORRY
i worry about getting good grades but somtimes i cant.
IS GETTING GOOD GRADES IMPORTANT TO YOU
my parents make me feel like i have to.
DO YOU CARE WHAT YOUR PARENTS MAKE YOU FEEL
sometimes i do.
WHEN WAS THE LAST TIME?
last thursday.
LAST THURSDAY
do you think coming to the guc was a good idea?
I AM A GUC COUNSELOR SO I AM NOT ALLOWED TO COMMENT ON THAT
what do you think about me studying harder?
I AM NOT SURE WHAT I THINK
i have to go now.
GOOD LUCK THEN
bye.
GOOD-BYE
```

Directions for Running and Using Mini-Eliza

Mini-Eliza should be loaded from the clisp interpreter and ran from there:

$ clisp
[]> (load “match.lisp”)
[]> (load “eliza.lisp”)
[]> (in-package eliza)
[]> (main)
…
…
bye.

Note All entered sentences should be single-lined and ending with either ., ! or ?. The use of commas or quotes in sentences isn't supported.

No external libraries were used, only the standard Common Lisp environment.

References

1. Common Lisp, An Interactive Approach. Stuart C. Shapiro.
2. http://en.wikipedia.org/wiki/Loebner_Prize
3. http://en.wikipedia.org/wiki/Rollo_Carpenter
4. http://en.wikipedia.org/wiki/Turing_Test
5. http://www.jabberwacky.com/
6. http://en.wikipedia.org/wiki/Jabberwacky
7. http://www.chayden.net/eliza/Eliza.html
8. http://en.wikipedia.org/wiki/Virtual_Woman
9. http://virtualwoman.net/
10. http://en.wikipedia.org/wiki/PARRY
11. http://www-2.cs.cmu.edu/afs/cs/project/ai-repository/ai/areas/classics/parry/
12. http://en.wikipedia.org/wiki/MegaHAL
13. http://sourceforge.net/projects/megahal/
14. http://megahal.alioth.debian.org/
15. http://www.simonlaven.com/hex.htm
16. http://en.wikipedia.org/wiki/Racter
17. http://en.wikipedia.org/wiki/Dr._Sbaitso
18. http://www.cs.cmu.edu/Groups/AI/html/cltl/clm/