ECE 479/579

Principles of Artificial Intelligence

Dr. Marefat

ASSIGNMENT # 1

"Developing a Rule-based System using *Jess*"

1 Problem description

In this project, you are asked to build a small production system to answer specific questions about a family tree. The production system should be implemented in Jess (Java Expert System Shell), an expert (or production) system shell in Java. Jess allows you to type in rules and facts (predicates), then perform both forward and backward reasoning with those facts. Jess itself is a very easy to use tool and to help you get started we have provided a simple example of a small production system.

1.1 Notes on getting Jess started

Starting Jess is extremely easy. Simply download and install Jess to your ece account or your Laptop/computer (make sure to follow the rules for correctly installing Jess (Please refer to documentation manual, FAQ and demo). Once you start Jess, the command-line interface will show up in which you can type in the knowledge of the domain and the queries or ask Jess to do inferences. Also note that instead of typing in rules and facts one at a time in the command-line interface, you can write them in a file (with an extension '.clp') and give it to Jess in batch mode. For example your file can be named 'myrules.clp'.

2 An example small production system

Let us say that we want to try to build the following description as a production system:

> All mammals are animals. All animals inhale oxygen (this is not biologically correct, but let's use it anyway). Sperm whales and elephants are mammals. Mammals are warm blooded. Elephants are grey, wrinkled and like peanuts. Sperm whales and sharks live in the ocean. Sharks are animals. All sharks and sperm whales are friends. Clyde is an elephant, John is a shark, and Henry is a sperm whale.

The encoding of the above quotation is fairly straight forward, however remember a few things:

- When we want to introduce a rule, it has the following form:

rule:
(defrule (and (pred1 term1 term2 .....) (pred2 term1 term2 .....) .....) => (predn term1 term2 .....))

- A fact is simply:

fact:
assert (predicate term1 term2 .....)

And comment lines start with the % character.

Type the following into a file:

%%% Printout of example
% this is a comment

%rule:
% Printout of example: mammals are animals
(defrule rule-mammal-animal (mammal ?X) => (assert (animal ?X)))

%rule:
% animals inhale oxygen
(defrule rule-animal-inhales (animal ?X) => (assert (inhales ?X oxygen)))

%rule:
% sperm whales & elephants are mammals
(defrule rule-sperm-whale-animal (and (whale ?X) (type ?X sperm)) => (assert (mammal ?X)))
(defrule rule-elephant-animal (elephant ?X) => (assert (mammal ?X)))

%rule:
% mammals are warm blooded
(defrule rule-mammal-warm-blood (mammal ?X) => (assert (blood_temp ?X warm)))

%rule:
% elephants are grey, wrinkled and like peanuts
(defrule rule-elephant-grey (elephant ?X) => (assert (color ?X grey)))
(defrule rule-elephant-wrinkled (elephant ?X) => (assert (skin ?X wrinkled)))
(defrule rule-elephant-like-peanuts (elephant ?X) => (assert (favorite_food ?X peanuts)))

%rule:
% sperm whales and sharks live in the ocean
(defrule rule-sperm-whale-ocean (and (whale ?X) (type ?X sperm)) => (assert (lives ?X ocean)))
(defrule rule-shark-ocean (shark ?X) => (assert (lives ?X ocean)))

%rule:
% sharks are animals
(defrule rule-shark-animal (shark ?X) => (assert (animal ?X)))

%rule:
% all sharks and sperm whales are friends
(defrule rule-shark-sperm-whale-friends (and (shark ?X) (whale ?Y) (type ?Y sperm)) => (assert (friends ?X ?Y)))

%fact:
(assert (elephant clyde))

%fact:
(assert (shark john))

%fact:

(assert (whale henry))

%fact:
(assert (type henry sperm))

The above predicates should be fairly obvious. Now save that file under some name, say *mammals.clp*. Hence we have 6 predicates in our problem. This is how your file should look like (comments which start with a % have been removed below for clarity):

```
(reset)

% facts
(assert (elephant clyde))
(assert (shark john))
(assert (whale henry))
(assert (type henry sperm))

% rules
(defrule rule-mammal-animal (mammal ?X) => (assert (animal ?X)))
(defrule rule-animal-inhales (animal ?X) => (assert (inhales ?X oxygen)))
(defrule rule-sperm-whale-animal (and (whale ?X) (type ?X sperm)) => (assert (animal
?X)))
(defrule rule-elephant-animal (elephant ?X) => (assert (mammal ?X)))
(defrule rule-mammal-warm-blood (mammal ?X) => (assert (blood_temp ?X warm)))
(defrule rule-elephant-grey (elephant ?X) => (assert (color ?X grey)))
(defrule rule-elephant-wrinkled (elephant ?X) => (assert (skin ?X wrinkled)))
(defrule rule-elephant-like-peanuts (elephant ?X) => (assert (favorite_food ?X
peanuts)))
(defrule rule-sperm-whale-ocean (and (whale ?X) (type ?X sperm)) => (assert (lives ?X
ocean)))
(defrule rule-shark-ocean (shark ?X) => (assert (lives ?X ocean)))
(defrule rule-shark-animal (shark ?X) => (assert (animal ?X)))
(defrule rule-shark-sperm-whale-friends (and (shark ?X) (whale ?Y) (type ?Y sperm))
=> (assert (friends ?X ?Y)))


%(facts)
(watch facts)
(run)

% define the query for finding whoever inhales oxygen
(defquery find-inhales "find inhales" (declare (variables ?Y)) (inhales ?X ?Y))

% run the above query
(bind ?it (run-query find-inhales oxygen))

% print out the results by iterating
(while(?it hasNext)
        (bind ?token (call ?it next))
        (bind ?fact (call ?token fact 1))
        (bind ?slot (fact-slot-value ?fact __data))
        (bind ?datum (nth$ 1 ?slot))
        (printout t ?datum crlf))
        (printout t ?datum crlf))
```

As can be seen we have used a `defquery` statement in the above file and some associated commands. These show you how to execute a query given certain facts have been derived. This particular query simply prints out names of mammals which inhale oxygen. The following should be the output of the running " mammals.clp" in Jess:

```
Jess, the Java Expert System Shell
Copyright (C) 1998 E.J. Friedman Hill and the Sandia Corporation
Jess Version 7.0 12/7/2007

f-0    (MAIN::initial-fact)
f-1    (MAIN::elephant clyde)
f-2    (MAIN::shark john)
f-3    (MAIN::whale henry)
f-4    (MAIN::type henry sperm)
For a total of 5 facts.
 ==> f-5 (MAIN::friends john henry)
 ==> f-6 (MAIN::animal henry)
 ==> f-7 (MAIN::lives henry ocean)
 ==> f-8 (MAIN::inhales henry oxygen)
 ==> f-9 (MAIN::animal john)
 ==> f-10 (MAIN::inhales john oxygen)
 ==> f-11 (MAIN::lives john ocean)
 ==> f-12 (MAIN::favorite_food clyde peanuts)
 ==> f-13 (MAIN::mammal clyde)
 ==> f-14 (MAIN::animal clyde)
 ==> f-15 (MAIN::inhales clyde oxygen)
 ==> f-16 (MAIN::blood_temp clyde warm)
 ==> f-17 (MAIN::color clyde grey)
 ==> f-18 (MAIN::skin clyde wrinkled)
 ==> f-19 (MAIN::__query-trigger-find-inhales oxygen)
 <== f-19 (MAIN::__query-trigger-find-inhales oxygen)
henry
john
clyde
```

The following is now the actual assignment description.

3 The family

Note: if a person's name looks *emphasized*, then that person is female.

Jenny had three children: *Grace*, *Harriet*, and Paul. Neither *Harriet* nor Paul married. However, *Grace* married and had two children: Leo and Gerald.

Leo had one child: *Wendy* who never married

Gerald had twins: *Marsha* and *Michelle*. Although *Michelle* never married, *Marsha* married and had two children: *Megan* and Matthew.

4 Predicates

The predicates you have available are:

- father(X,Y) - X is the father of Y.
- mother(X,Y) - X is the mother of Y.
- gender(X,Y) - If Y=male then X is male. If Y=female then X is female.
- brother(X,Y) - X is the brother of Y.
- sister(X,Y) - X is the sister of Y.
- son(X,Y) - X is the son of Y.
- daughter(X,Y) - X is the daughter of Y.
- grandson(X,Y) - X is the grandson of Y. Your grandson is any son of any of your children.
- greatgrandson(X,Y) - X is the great-grandson of Y. Your great-grandson is any son of any of your grand-children. (A grand-child is a grandson or granddaughter).
- granddaughter(X,Y) - X is the granddaughter of Y. Your granddaughter is any daughter of any of your children.
- greatgranddaughter(X,Y) - X is the great-granddaughter of Y. Your great-granddaughter is any daughter of any of your grand-children.
- cousin(X,Y) - X is the cousin of Y. Your cousins are the children of your parent's siblings.
- niece(X,Y) - X is the niece of Y. Your niece is a daughter of any of your siblings.
- nephew(X,Y) - X is the nephew of Y. Your nephew is a son of any of your siblings.
- aunt(X,Y) - X is the aunt of Y. Your aunt is any sister of any of your parents.
- greataunt(X,Y) - X is the great-aunt of Y. Your great-aunt is any aunt of any of your parents.
- uncle(X,Y) - X is the uncle of Y. Your uncle is a brother of any of your parents.
- greatuncle(X,Y) - X is the great-uncle of Y. Your great-uncle is an uncle of any of your parents.
- grandfather(X,Y) - X is the grandfather of Y. Your grandfather is a father of any of your parents.
- greatgrandfather(X,Y) - X is the great-grandfather of Y. Your great-grandfather is a grandfather of any of your parents.
- grandmother(X,Y) - X is the grandmother of Y. Your grandmother is a mother of any of your parents.
- greatgrandmother(X,Y) - X is the great-grandmother of Y. Your great-grandmother is a grandmother of any of your parents.

5 Implementation and restrictions

You are required to encode the above family tree with the above primitives. Recall that we can either setup a bunch of facts or a small set of facts along with a set of rules that can derive more facts. Therefore you are *only* allowed to use the primitives father(X,Y), mother(X,Y) and gender(X,Y) to encode **the facts** of the family tree. Also, you should not include unnecessary facts. For example, if you type in the fact Jenny is a mother, you do not need the fact that Jenny is female. That can be derived. You must derive the other facts with rules about sons, daughters, cousins, etc.... The idea is to make the facts as few as possible and reason with them to generate the rest of the knowledge.

As far as how you write the rules, that is up to you. Because these relationships often have ``inverses'' there is no one correct solution to the problem. For example, you could either encode each rule based on

the idea of parents and siblings, or you could also write the rule for aunt once you know something about niece and nephew. **Please use lower case for all predicates, correct case for names/constants (Leo, Gerald, male/female etc.) and correct spellings for names and predicates (its niece and not neice)!** Some frequently asked questions (FAQ's) have been included in Appendix B.

6 Questions to answer

For the purposes of your assignment, you should submit one text file that contains your facts and rules of the above description. This file should be loadable by Jess and produce the intuitively correct results for any of the above predicates. However, in order to help you test your program you may try to answer the following:

- Who are cousins?
- Who is the great uncle of Kimberly?
- Who are aunts, uncles, grandfathers, grandmothers?
- Who are great-aunts, great-uncles, great-grandparents?

Any two of the above should be made into a query, along with an appropriate iteration and print block as shown in the mammals example. Any 2 of the above mentioned 4 queries should be included in the family.clp file after all the facts and rules have been encoded. You may also test your program by using the test cases described in the Appendix A.

7 Turning in the assignment

You will submit an electronic copy of your work (details to follow). Please put all of your facts and rules in one text file loadable by Jess, and call this file `family.clp`.

Also print out a copy of your `family.clp` file and a sample run (with 2 query results) and bring them to class to turn in on the due date of the assignment.

Copying any part of another student's assignment, or working with others on this assignment other than discussing the approach will be considered plagiarism and breaking the code of academic integrity.

Appendix A
Testing your code

Please conduct these tests before you turnin your program, to make sure that you have coded correctly. These 2 tests can be conducted once you have run the family.clp file from the Jess shell prompt using the "(batch ./family.clp)" command. Then do "(batch ./test1.clp)" and "(batch ./test2.clp)" to conduct these tests (in proper order - test1 followed by test2).

Test 1 - `test1.clp`

This test can be conducted by adding an additional relevant fact (Kimberley is the daughter of Leo):

```
(assert (gender Kimberley female))
(assert (father Leo Kimberley))
(watch facts)
(run)
```

and testing new inferences. 15 new facts should be inferred (if you did not use any of your own predicates!). They are:

(MAIN::sister Kimberley Wendy)
(MAIN::sister Wendy Kimberley)
(MAIN::daughter Kimberley Leo)
(MAIN::niece Kimberley Gerald)
(MAIN::grandmother Grace Kimberley)
(MAIN::uncle Gerald Kimberley)
(MAIN::granddaughter Kimberley Grace)
(MAIN::greatuncle Paul Kimberley)
(MAIN::cousin Michelle Kimberley)
(MAIN::cousin Kimberley Michelle)
(MAIN::greataunt Harriet Kimberley)
(MAIN::cousin Marsha Kimberley)
(MAIN::cousin Kimberley Marsha)
(MAIN::greatgrandmother Jenny Kimberley)
(MAIN::greatgranddaughter Kimberley Jenny)

Test2 - `test2.clp`

This test involved testing with an additional query (who are the nieces?):

```
% define the query for finding whoever are whoever's niece's
(defquery find-nieces "find nieces" (niece ?X ?Y))

% run the query find-nieces
(bind ?it (run-query find-nieces))

% print out the results by iterating
(printout t "Who are nieces?" crlf)
(while(?it hasNext) (bind ?token (call ?it next))
(bind ?fact (call ?token fact 1))
(bind ?slot (fact-slot-value ?fact __data))
(bind ?datum1 (nth$ 1 ?slot))
(bind ?datum2 (nth$ 2 ?slot))
(printout t ?datum1 " is the niece of " ?datum2 ". " crlf))
(printout t " " crlf)
```

It should produce 5 facts:

Kimberley is the niece of Gerald.
Wendy is the niece of Gerald.
Marsha is the niece of Leo.
Michelle is the niece of Leo.
Megan is the niece of Michelle.

Appendix B
Frequently Asked Questions (FAQ)

Q. Can I run .clp files from the Jess shell prompt ?

A. Assuming that your .clp file is in the same directory from which the jess shell was invoked, you can run your .clp file by using the batch command.

Jess, the Java Expert System Shell
Copyright (C) 1998 E.J. Friedman Hill and the Sandia Corporation
Jess Version 6.0 12/7/2001

Jess> **(batch ./myFile.clp)**

Q. predicates of my own ? This will make some of my rules simpler to encode!

A. No. Please do not add predicates of your own (such as parent etc.). Stick to the predicates mentioned in the assignment. It makes grading difficult if you use your own predicates in rules (cause then extra facts may be inferred using that extra predicate which you introduced). Also please use the proper spellings/case for predicates and names/constants (refer to assignment [PDF]).

Q. family.clp file seems to be giving an error when I try to run it using Jess ?

A. It has been noticed that some comments lead to errors sometimes. So keep your comments short and simple (stick alphabets and no fancy equations probably). Specifically avoid curly braces "{" "}" in your comments.

Q. How do I use eq and neq for testing whether two variables are equal or not ?

A. If you want to use new (or eq) in any of your rules then nest it inside a test statement. Hence something like (and (gender ?X male) ..... (test (neq ?X ?Y)).....) => .....

Q. How do I define multiple queries in a single .clp file ?

A. Writing multiple queries is very much possible. Simply define seperate queries and iterate over the results generated by each and printing out the relevant answers, as shown in the assignment example.

Q. Kimberley ?

A. One of the queries wants you to find the great uncle of Kimberly. This is correct since it should not return anything! Remember that we can always add a fact about Kimberly (just to test whether you programmed correctly or not) and then this query should work (and hopefully produce something)!

Q. Can I use as many facts as I want ?

A. Please code only facts which are necessarily needed. As a hint if you encode the fact that x is y's mother (assuming x and y are constants), do you need to encode the gender of x ?? If you can come up with a technique for avoiding this type of assertions, then we don’t need to add a million mothers to the Knowledge Base, we are potentially saving time by not hard coding a million facts.