2009 November 30

# Chat
# Example Test Questions

Describe and explain how the chat program works. For readline describe the basic steps and data structures involved – Prolog is not necessary. For parse and respondTo describe and explain the basic syntax, semantics and data structures used to handle facts and queries.

```
chat :- repeat
        , readLine(Sentence)
        , parse(Clause , Sentence , _ )
        , respondTo(Clause)
        , Clause = stop .
```

**1.**

For the `parse` and `respondTo` predicates in `chat` describe the syntax and explain the basic semantics and data structures used to handle facts and queries.

**2.**

Describe and explain how the readline predicate in the chat program works. Describe the steps and data structures involved – Prolog is not necessary

**3.**

For the `parse` and `respondTo` predicates in `chat` describe the syntax and explain the basic semantics and data structures used to handle facts and queries.

**4.**

The chat program to query the ring cycle database is to be extended with the following query, where Fafner could be replaced with any name and the word "die" could be replaced with another word; i.e. the query is to be programmed in a more general way than strictly necessary for querying the ring cycle database. No other variables are to be inferred from the sample query.

Does Fafner die for more than one reason?

Define a parse rule for the above query using the functor ">1reason" for the resulting clause.

The database contains the following predicate – `dies(X)` – asserts that X dies
You need to use following predicates. The atoms in the database are all in lower case.

`plural(Singular, Plural)` – asserts that the atom Plural is the plural of the atom – Singular; plural(love, loves) is true.
`uppercap(Lower, Upper)` – Asserts that the atoms Lower and Upper are equal except – the case of the first letter is lower case for Lower and – upper case for Upper. Lower cannot be a variable.
`lowercap(Upper, Lower)` – Like uppercap except that Upper cannot be a variable.

The following predicates may be of use.

```
bagof(Variable, Predicate, Bag)
setof(Variable, Predicate, Set)
findall(Variable, Predicate, List)
```

**5.**

Define a respondTo rule for the query in Question 4. The functor for the clause is ">1reason". The response is one of the following, depending upon the facts in the database.

Yes, for N reasons. – where N > 1 is the number of reasons.
No, Fafner does not die.
No, Fafner dies for only 1 reason.

If you need predicates other than those given in Question 4, then you must give their definition.

**6.**

**A** You want to extend the chat program to query the ring cycle database. The database contains the following predicates

`brother(X,Y)` – Y is a brother of X
– a set of instances is in the database, for example brother(Fasolt, Fafner) could be in the database.

`dies(X)` – a predicate that is true if X dies.

Give a parse rule for the following type of question.

Does the ____ of ____ ___?

An example instance of the type of question is the following. Note the predicate is dies and the verb in the question is die. The character "s" is ASCII 115.

Does the brother of Fasolt die?

**B** Give a respondTo rule, corresponding to the parse rule in Part A that gives an appropriate response, depending upon which facts are in the database, as in the following.

"Yes, Fafner does."
or "No, Fafner does not die."
or "Fasolt has no brother."

**7.**

**A** Give a parse rule for the following type of query to add to the database. Make the resulting clause unique with the prefix "isa".

Is ___ the ___ of ___? Example: Is Alice the aunt of Mary?

**B** Give a response rule for the query in Part A. An example response would be "No, Alice is not the mother of Mary." or "Yes, Alice is the mother of Mary.", depending upon whether or not the database contained the appropriate facts at the time the query was made.

**8.**

**A** Give a parse rule for the following query to be added to the exercise in Report 4. Make the resulting clause unique with the functor "attr".

What are the attributes of the grid abc?

`Parse(Clause) -->` You supply the rest in the context of Part B; i.e. the resulting clause is to contain the complete query needed for Part B.

**B** Give a response rule for the query in Part A, where the following are the possible responses. You must use the query as constructed in Part A.

- There is no grid abc. – if the grid does not exist.
- The grid has rrr rows and ccc columns, and has nnn timesTen cells.

**9.**

The parse rules in the chat program were entered in a user-friendly form, for example, the following.

```
parse(Clause) --> [ Who , is , the ] , type(T) , [ of ] ,
                  thing(Name) , [ '?' ] ,
                { Goal =.. [T,X,Name] , Clause = '?w'(Goal) , !
}.
```

Such rules are translated into standard Prolog syntax. Give a translation of the above rule and explain how the translation parses a sentence and what semantic actions take place.

**10.**

**C** Give a parse rule for the following type of rule to add to the database

Alice is the aunt of Mary if Alice is the sister of Eliza and Eliza is the parent of Mary.

**11.**

Consider statements of the following form.

The _1_ of _2_ are _3_ . **(1)**

where _1_ can be any predicate of the form _1_(_2_, arg1, arg2) and _3_ can be any predicate of the form _3_(arg)

For example, the following.

| | | |
|---|---|---|
| The plants of Alice are green. | plants('Alice' , rose , petunia). | green(rose). |
| The parents of Tom are tall. | parents('Tom' , 'Astra' , 'Ali'). | tall('Astra'). |

The first statement, if true, corresponds to the following query.
plants('Alice', Plant1, Plant2) , green(Plant1) , green(Plant2).

While the second statement, if true, corresponds to the following query.
parents('Tom', Parent1, Parent2) , tall(Parent1) , tall(Parent2).

**A.** Give a parse rule to read in statements like form **(1)** and produce the output clause as a three argument compound term with functor '??' – ??(Predicate_1, Predicate_2, Predicate_3).

**B.** Write a respondTo rule that corresponds to the parse rule in Part A. You are to handle only the cases shown in the following example.

Assume the following facts are the only ones in the database.

| | |
|---|---|
| plants('Alice' , rose , petunia). | parents('Tom' , 'Astra' , 'Ali'). |
| plants('Tom' , _ , lilac). | parents('Jing' , _ , 'Leila'). |
| plants('Ali' , _ , rose). | parents('Xun', _ , 'Yevgeniy'). |
| green(lilac). | tall('Leila'). |

Entering "The plants of Leila are green." gives a response of "Leila has no plants."
Entering "The parents of Yevgeniy are tall." gives a response of "Yevgeniy has no parents.

Entering "The plants of Tom are green." gives a response of "Only Tom's second of plants is green.
Entering "The parents of Jing are tall" gives a response of "Only Jing's second of parents is tall.

Entering "The plants of Ali are green." gives a response of "Ali's second of plants is not green.
Entering "The parents of Xun are tall" gives a response of "Xun's second of parents is not tall.

Entering any other case gives a response of "Case is not handled."

Assume the existence of the predicate addPossessive(Name, Possessive) that asserts Possessive = Name's. For example, addPossessive('Alice', P) makes P = Alice's.