Prolog Tutorial Exercise 1 (Intro to Prolog)

Topics: Logic programming and Prolog terms
Prolog rules, facts, and queries

Goals: Upon successful completion of this tutorial you should be able to:

1. Write a simple Prolog rule
2. Create a base of facts for a prolog session
3. Write a simple Prolog query

Related text sections:

Prolog Tutorial Exercise 1 Instructions

For this tutorial you will be starting your journey into the world of Prolog! This tutorial is divided into 3 parts. The first part of this tutorial consists of a set of questions regarding the language Prolog. You may use web and/or printed resources to answer these questions. The second part of this tutorial is a directed Prolog exercise using gprolog on the linux platform. In the third part of this tutorial you will be creating a set of Prolog facts and queries.

Tutorial 1 Part 1.

1. To what programming paradigm family does the language Prolog belong?

2. For what types of applications is Prolog an appropriate programming language?

3. Describe each of the following terms with respect to the Prolog programming language:

    a. world
    b. fact
    c. rule
    d. base
    e. predicates
    f. arguments
    g. arity
    h. list
    i. instantiation
    j. unification

4. Describe the Prolog syntax rules regarding each of the following:

    a. selecting variable names
    b. defining constant (literal) values
    c. use of periods (.)
    d. use of semicolons (;)
    e. use of commas (,)
    f. formulation of facts
    g. formulation of queries
    h. formulation of rules
    i. assignment statements
    j. conditional expressions

5. How do you create a Prolog source file? How do you document a Prolog source file? How do you load the file when using the Prolog interpreter?

Tutorial 1 Part 2.

1. Copy the files forecast6.pl and photoshoot6.pl from ~tiawatts/cs460pickup to your account.

2. Start the Gnu Prolog interpreter by entering **gprolog** at the linux prompt.

3. Load the file forecast6.pl by entering **['forecast.pl'].** at the Prolog interpreter prompt.

4. List the contents of the file by entering **listing.** at the Prolog interpreter prompt.

5. Query the (data)base to determine if Friday will be sunny by entering **weather(friday,sunny).** at the Prolog interpreter prompt.

6. Query the (data)base to determine if Saturday will be rainy.

7. Query the (data)base for Monday's weather forecast by entering **weather(monday,What).** at the Prolog interpreter prompt.

8. Create and enter a query for today's weather forecast.

9. Query the (data)base for a rainy day by entering **weather(Day,rainy).** at the Prolog interpreter prompt.

10. Create and enter a query for a snowy day.

11. Create and enter a query for fair days; use the semicolon operator to continue the query.

12. Exit from the Prolog interpreter by entering **halt.**

13. Restart the Gnu Prolog interpreter and load and compile the file photshoot6.pl.

14. The file 'photoshoot.pl' contains the line: **sky(blue, A) :- weather(A, fair).** What does this statement mean?

15. Execute the query: **sky (blue, friday).** What is the result of this query?

16. Write a query to determine the color Friday's sky.

17. Write a query to determine the color of today's sky.

18. The file 'photoshoot.pl' contains the line: **goodpictures(A) :- sky(blue, A), weekend(A).** What does this statement mean?

19. Execute the query: **goodpictures(monday).** What is the result of this query? How do you think Prolog arrives at this conclusion?

20. Turn on the Prolog trace option (by entering trace.) and reenter the above query. Press the enter key to move from one Prolog operation to the next.

21. Enter and trace the query: **goodpictures(D).** What is the result of this query?

22. Exit from Prolog.

Tutorial 1 Part 3.

1. For a topic of your choice, create a set of 10 facts and enter them into a prolog data base file.

2. Create 2 simple rules and and one compound (conjunction or disjunction) rule for your data base.

3. Create a list of 10 queries and their results for your program.
4. Use gprolog to compile your data base and test your queries.

Prolog Tutorial Exercise 2 (Prolog “Functions” and Lists)

Topics: Prolog compound rules
Prolog List structures

Goals: Upon successful completion of this tutorial you should be able to:

1. Write a compound prolog rule
2. Manipulate the items in a prolog list

Related text sections:

Prolog Tutorial Exercise 2 Instructions

Tutorial 2 Part 1.

1. Copy the files factorial.pl and list.pl from ~tiawatts/cs460pickup to your account.

2. Start the Gnu Prolog interpreter and load the file factorial.pl.

```
factorial(0,1).

factorial(A,B) :-
        A > 0,
        C is A-1,
        factorial(C,D),
        B is A*D.
```

1. Enter the query **factorial(5,What).** Explain this result:

2. Enter the query **factorial(0,Value).** Explain this result:

3. Analyze the factorial function; you should consider the following questions:

    a) What is the role of the statement `factorial (0,1).` in the definition of factorial?
    b) What is the role of the statement `A > 0` in the definition of factorial?
    c) What is the role of the statement `factorial (C,D)` in the definition of factorial?
    d) Explain the pattern of variable uses in the last four lines.

4. Exit from the Prolog interpreter.

Tutorial 2 Part 2.

1. What is a Prolog "list"?

2. What is the syntax used to create a Prolog list?

3. How are the terms "head" and "tail" defined with respect to a Prolog list?

4. How is membership in a list determined?

5. How is an empty list represented?

Tutorial 2 Part 3.

1. Start the Gnu Prolog interpreter and load the file list.pl. Use the following activities to analyze the statements in the file list.pl.

2. Use the [user] command to create 2 list facts, list1 and list2.

```
[user].
list1([a,b,c,d,e]).
list2([x,y,z]).
Ctrl-D
```

3. Enter the following query to determine if a is the first element in list1:

```
list1(L),is_first(a,L).
```

4. Enter the following query to determine the identity of the first element in list2:

```
list2(L),is_first(X,L).
```

5. Create similar queries to test is_last, is_in, how_long, and place.

6. Add comments to the file list.pl to explain how each of these rules works. (You may wish to use the trace command to view these rules in action).

7. Enter the following command to push the item m onto the stack represented by list1.

list1(List),insert_first(m,List,Newlist).

8. Enter the following command to pop the first item from the stack represented by list2.

list2(List),remove_first(List,Newlist).

9. Enter the following command to concatenate 2 lists.

concat([a,b,c],[d,e,f],L).

10. Create a query that uses concat to determine if one list is the prefix of another. Create a query that uses concat to determine if one list is the suffix of another.

1. Add comments to the file list.pl to explain how each of these rules works. (You may wish to use the trace command to view these rules in action).

Tutorial 2 Part 4.

1. Add a rule called insert_last to this set of rules. insert_last should have 3 arguments: an element, a list, and a resulting list. The element should be appended to the end of the original list to create the resulting list.

2. Add a rule called remove_last to this set of rules. remove_last should have 2 arguments: an original list and a modified list. The last element of the original list should be removed to create the modified list.

3. Add a rule called switch. switch should have 2 arguments: an original list and a modified list. The elements in the original list should be reversed to create the modified list.

Prolog Tutorial Exercise 3 (Prolog Sorting)

Topics: Sorting in Prolog

Goals: Upon successful completion of this tutorial you should be able to:

1. Use prolog rules of unification to create a sorted list

Related text sections:

Prolog Tutorial Exercise 3 Instructions

1. The following rules provide the framework for a Prolog implementation of quicksort. Using the list rules you created for tutorial2, write rules for quick_split/4. Test your quick_split and quicksort rules. Place these rules in a file called sort.pl.

```
quicksort([],[]).
quicksort([Element],[Element]).
quicksort([Head|Tail],SortedList) :-
                                    quick_split(Head,Tail,L,H),
                                    quicksort(L,SL),
                                    quicksort(H,SH),

concat(SL,[Head|SH],SortedList).
```

OR/AND

The following rules provide the framework for a Prolog implementation of mergesort. Using the list rules you created for tutorial2, write rules for merge_split/3 and merge/3. Test your merge_split, merge and mergesort rules. Place these rules in a file called sort.pl.

```
mergesort([],[]).
mergesort([Element],[Element]).
mergesort(List,SortedList) :-
                                    merge_split(List,L1,L2),
                                    mergesort(L1,SL1),
                                    mergesort(L2,SL2),
                                    merge(SL1,SL2,SortedList).
```

2. Test your sort(s) using the rules in cs460lpickup/tutorial3.pl and the following gprolog script:

```
$ cat tutorial3.pl sort.pl > temp.pl
$ gprolog
GNU Prolog 1.2.8
By Daniel Diaz
Copyright (C) 1999-2001 Daniel Diaz
| ?- [temp].
compiling temp.pl for byte code...
temp.pl compiled, 96 lines read - 2978 bytes written, 39 ms
(10 ms) yes
| ?- setup.
yes
| ?- run1(list0,list1),ilist(list1,L).
L = [-3,-2,-1,1,1,2,2,3,3] ?
yes
| ?- run2(list0,list2),ilist(list2,L).
L = [3,3,2,2,1,1,-1,-2,-3] ?
yes
| ?- halt.
$
```

3. Place your well documented rules for tutorials 2 and 3 in a file called *your-last-name*.pl and drop the file into the cs460drop folder.