

Assignment 4: Prolog

Assigned: Friday, 11 February 2011

Due: 11:00 p.m., Thursday, 17 February 2011

Summary: In this assignment, you will further explore some simple applications of Prolog.

Purposes: To give you more experience with Prolog.

Expected Time: Three to four hours.

Collaboration: I would prefer that you work on this assignment in groups of two or three. You may also work on this assignment alone or in groups of up to size four. You may consult with anyone you wish on this assignment (including other classmates), provided you cite your consultants.

Submitting: Upload a tarball to Pioneerweb. See further details below.

Warning: So that this assignment is a learning experience for everyone, I may spend class time publicly critiquing your work.

Part I: Sorting and Stuff

1. Selection Sort

Implement a `selection_sort(L,S)` predicate that holds if `S` is the sorted version of `L` and that makes the determination using selection sort.

You may find it helpful to write a `smallest(L,X)` predicate that holds when `X` is the smallest element in `L`. You may also find it helpful to write a `remove(L,X,M)` predicate that holds when `M` is the result of removing one copy of `X` from `L`.

2. Insertion Sort

Implement an `insertion_sort(L,S)` predicate that holds if `S` is the sorted version of `L` and that makes the determination using insertion sort.

You may find it helpful to write an `insert(L,X,M)` that holds if `M` is the result of inserting `X` into the proper place in `L` (assuming that `L` is already sorted).

3. Merge Sort

Implement a `merge_sort(L,S)` predicate that holds if `S` is the sorted version of `L` and that makes the determination using merge sort.

You may find it helpful to write a `split(L, L1, L2)` procedure that holds if `L1` and `L2` are the result of splitting `L` in half. (And you may find it easiest to split a list of you alternately put elements in to one list or the others.)

You may also find it helpful to write a `merge(S1, S2, S)` that holds when `S` is the result of merging sorted lists `S1` and `S2`.

4. Quicksort

Implement a `quicksort(L, S)` predicate that holds if `S` is the sorted version of `L` and that makes the determination using Quicksort.

You may find it useful to write a `partition(Pivot, L, Smaller, Equal, Larger)` predicate.

5. Quickselect

Implement a `quickselect(L, K, X)` predicate that holds if `X` would be the `K`th element of the sorted version of `L`. (Of course, you should not sort to solve this problem.)

The Quickselect algorithm is much like Quicksort, except that you only need to recurse on one of the partitions.

Part 2: Representing Integers

As you may have noted, we can represent a non-negative integer, n , as a list of n values (say, n 1's). Let's play with that representation.

6. Addition

Implement a `add(X, Y, Sum)` procedure that holds if `Sum` is the sum of `X` and `Y`.

For example,

```
sum([1,1], [1,1,1], [1,1,1,1,1])
```

You may not use the built in arithmetic operations.

7. Converting Representations

Implement a `int2list(N, L)` predicate that holds when `N` can be represented as `L` using the list representation. Implement a `list2int(L, N)` predicate that holds when `L` represents `N`. (It's helpful to have two separate predicates because you may have to use `is`, which does not do well with variables on the right.)

```
?- int2list(5, [1,1,1,1,1])
yes
?- int2list(5, X).
X = [1,1,1,1,1] ;
yes
?- list2int([1,1,1,1,1], 5).
yes
?- list2int([1,1,1,1,1], X).
X = 5 ;
yes
```

Note that this pair of procedures can be helpful for testing your `sum` procedure.

```
int2list(3,X), int2list(4,Y), sum(X,Y,Z), list2int(Z,N).
```

8. Implementing Comparison

Implement a `lessthan(X,Y)` predicate that holds if `X` is less than `Y`. (`X` and `Y` should use the list representation.)

9. Implementing Multiplication

Implement a `multiply(X,Y,Product)` procedure that holds if `Product` is the product of `X` and `Y`.

Important Evaluation Criteria

I will look at the standard “C” criteria: Is your code **correct**? Is your code **concise**? (This second attribute is particularly appropriate in Ruby.) Is your code appropriately **commented**? Have you used any particularly **clever** techniques. How **comely** is your code?

Submitting Your Homework

Create a directory for this assignment of the form `hw03.names`. Within that directory, create subdirectories for each part of the assignment (e.g., `part1`, `part2`). Each subdirectory should contain:

- The code that you have written for that part of the assignment.
- A typescript or typescripts in which you demonstrate the utility of the code you have written.
- (Optional) Unit tests.
- (Optional) A README file with any additional info you want to provide.

Make a gzipped tarball of the assignment directory. (If you don’t know how to do that, let me know.) Upload it via the Assignment 4 link on Pioneerweb.