CS 310 | Fall 2020 | Unit 5 | Survey of Logical Languages

Topics: *Logical programming, language translation, predicate logic*

Follow the instructions. Submit the indicated files (📎) via the assignment in the eCampus lecture section by the due date on the course calendar. For each source file, include a comment with your full name and a brief statement acknowledging that your work complies with the academic integrity policy. The grading rubric is in this document (percentages are scaled as in the syllabus).

Instructions

Choose **one** of the two predicates below and follow the steps for your chosen predicate (**100%**).

Deck Cutting Predicate

A. Write a 📎 script in **Prolog** with all the facts and rules necessary to implement the `cut/2` predicate (50%).
   1. The name is unrelated to the cut goal (`!`) in Prolog. Rather, it represents cutting a deck of cards.
   2. The semantics are that `cut(Before, After)` is true when `After` is `Before` with the first and second halves transposed.
   3. The halves are equal in size. If there is an odd number of elements, preserve the middle element between the two halves.
   4. Assume `Before` or `After` is instantiated when the predicate is queried.
   5. Assume `Before` and `After` are simple lists of atoms.
   6. These examples cover all four instantiation cases, three valid which must be supported (✓) and one undefined (?).
      i. ✓ Both variables instantiated: `cut([1,2,3,4,5,6,7], [5,6,7,4,1,2,3]).` yields a `true` or `yes` result.
      ii. ✓ Only `Before` instantiated: `cut([1,2,3,4,5,6], R).` unifies `R = [4,5,6,1,2,3]` as its one result.
      iii. ✓ Only `After` instantiated: `cut(L, [1,2]).` unifies `L = [2,1]` as its one result.
      iv. ? Neither variable instantiated: `cut(L, R).` is undefined (since results would be generated arbitrarily).

B. Write a 📎 class in **Java** equivalent in functionality to the Prolog script in step A (50%).
   1. All valid instantiation cases must be supported using one or more overloaded methods.
   2. The formal parameters of a method correspond to the instantiated variables of that case.
   3. The return value of a method corresponds to the result of that case.
      i. For cases which only yield `true` or `yes` results, define a method returning a boolean.
      ii. For cases which unify a variable, define a method which returns the unified result of that variable.

Iterated Logarithm Predicate

C. Write a 📎 script in **Prolog** with all the facts and rules necessary to implement the `lgstar/2` predicate (50%).
   1. The semantics are that `lgstar(N, Iterations)` is true when `Iterations` is the base-2 iterated logarithm $\lg^*(N)$.
   2. This is not the base-10 iterated logarithm $\log^*(N)$, so use a change of base accordingly.
   3. Assume `N` is instantiated when the predicate is queried.
   4. Assume `N` is a number and `Iterations` is an integer.
   5. These examples cover all four instantiation cases, two valid which must be supported (✓) and two undefined (?).
      i. ✓ Both variables instantiated: `lgstar(70000, 4).` yields a `true` or `yes` result.
         a. When intermediate non-integer values are floored to integers, 4 iterated applications of the base-2 logarithm are needed to reduce 70,000 to 1 or less.
         b. Otherwise, when intermediate non-integer values are not floored to integers, 5 applications are needed.
         c. Either of the above interpretation is acceptable.
      ii. ✓ Only `N` instantiated: `lgstar(70000, X).` unifies `X = 4` or `X = 5` as its one result (as interpreted above).
      iii. ? Only `Iterations` instantiated: `lgstar(Y, 4).` is undefined (since there are infinitely many `Y` instantiations).
      iv. ? Neither variable instantiated: `lgstar(A, B).` is undefined (since results would be generated arbitrarily).

D. Write a 📎 class in **Java** equivalent in functionality to the Prolog script in step C (50%), following the requirements of step B.

Bonus Opportunity

The following bonus opportunity is available.

E. Complete **both** predicates instead of just the required one.
   1. The bonus predicate is worth **25% bonus** added to your grade for the unit (maximum of 105%).
   2. Any excess bonus above the maximum grade for this unit is redistributed to another eligible unit.