Quiz 13a Rubric

1. (2 points) We’re trying to use logic programming to solve the following puzzle:

Both Andy and Charlie are taller than Bob, and Dan is taller than both Andy and Charlie. Who is the tallest? (Assume that there is no ambiguity – we can deduce who the tallest person is.)

We add the following facts:

(assert! (taller andy bob))

(assert! (taller charlie bob))

(assert! (taller dan andy))

(assert! (taller dan charlie))

Now let’s actually solve the puzzle. The first thing that comes to mind is to say that nobody is taller than the tallest person: (not (taller ?other ?tallest)). Unfortunately, this doesn’t work. Fix the query. You can assume that member has been written, and that we only consider Andy, Bob, Charlie, and Dan (this is a hint!). An explanation may earn partial credit, but is not required.

Another hint: not succeeds if for **every** possible binding, the “argument” to the not would fail.

(and (member ?tallest (andy bob charlie dan))

(not (taller ?other ?tallest)))

3

The query we want is “Find the person for which every other person is not taller”. So, ?other should be unbound in the not (tallest person should be taller than **every** other person), but ?tallest should be bound (there is only one tallest person).

2 points. Partial credit assigned based on the query and the explanation (if there is one).

1. (3 points)  Write a rule for length. It should work as shown below. Additionally, you can assume you have a rule plus-one already written.

A description...

;;; Query input:                  ;;; Query input:

(length (a b c) ?x)               (plus-one 4 ?what)

;; Query results:                 ;;; Query results:

(length (a b c) 3)                (plus-one 4 5)

;;; Query input:                  ;;; Query input:

(length (a (b)) ?x)               (plus-one ?what 3)

;;; Query results:                ;;; Query results:

(length (a (b)) 2)                (plus-one 2 3)

(rule (length () 0))

(rule (length (?x  . ?y) ?result)

     (and (length ?y ?one-less)

          (plus-one ?one-less ?result)))

Rubric:

3 points - correct

2 points - missing base case, but otherwise correct

1.5 points - missing one of the expressions to the and, or tries using

a query match as if it was a function

1 point - shows some understanding

0 points - completely incorrect

1. (1+2+2 points) Louis Reasoner takes a look at the definition of list?

(assert! (rule (list? ())))

(assert! (rule (list? (?a . ?b))

(list? ?b)))

“It’s all wrong! The recursive rule is for a list with at least two elements, and the base case is for a list with no elements. There’s no case for a list with only one element!”

1. Explain what Louis has misunderstood.

It is not true that the recursive rule only works for a list with at least two elements – for a list with one element, ?a would be bound to the element, and ?b would be bound to ().

1. To prove that the rule works for one element lists, show all of the (successful) unifications that allow the logic programming language to derive the fact (list? (one)).

There are two successful unifications for (list? (one)):

Unifying (list? (one)) with (list (?a . ?b)) gives ?a •✇ one, ?b •✇ ()

Unifying (list? ()) with (list? ()) gives a successful unification without any bindings.

1 points for each unification. Be very lenient with the unifications – as long as they realize that there are two unifications, one for the recursive case with a one-element list, and one for the base case, they should get the 2 points.

1. Unfortunately, Louis has already added a new rule to deal with lists with one element:

(assert! (rule (list? (?x))))

Now, Ben Bitdiddle tries to use the list? rule, and sees the following:

;;; Query input:

(list? (is this a list))

;;; Query results:

(list? (is this a list))

(list? (is this a list))

Explain why (list? (is this a list)) is printed out twice.

There are two ways to deduce the fact (list? (is this a list)), and so it is printed out twice. In both cases we first use the recursive case to show that (list? (is this a list)) is true if (list? (list)) is true. We can then show (list? (list)) is true using either the same method as in part b, or using the new rule – each of these corresponds to one of the printed results.