Quiz 13b

1. (4 points) Define the rule substitute. It should work as follows:

;;; Query input:

(substitute (a b a c d) a q ?what)

;;; Query output:

(substitute (a b a c d) a q (q b q c d))

;;; Query input:

(substitute (r s t u) s s ?what)

;;; Query output:

(substitute (r s t u) s s (r s t u))

;;; Query input:

(substitute (s (s) s s) s z ?what)

;;; Query output:

(substitute (s (s) s s) s z (z (s) z z))

input old new result

(assert! (rule (substitute (?a . ?y) ?a ?b (?b . ?s))

(substitute ?y ?a ?b ?s))

(assert! (rule (substitute (?x . ?y) ?a ?b (?x . ?s))

(and (substitute ?y ?a ?b ?s)

(not (same ?x ?a)) )))

(assert! (rule (substitute () ?a ?z () )))

(assert! (rule (same ?x ?x)))

Rubric:

4 points - correct (don’t take off points for not defining same)

3 points - missing base case, or will lead to correct answer output plus other outputs

2 points - tries using a query match as if it was a function, otherwise close to correct

1 point - shows some understanding

0 points - shows no undersatnding

1. (4 points) Write the rule for list-ref. (list-ref ?x ?y ?z) is the fact “The element of ?x at index ?y is ?z”. We will use the following representation for the index:

0 -> zero, (s 0) -> one, (s (s 0)) -> two, (s (s (s 0))) -> three, and so on.

Here are some examples:

;;; Query input:

(list-ref (she loves you yeah yeah yeah) (s (s 0)) ?what)

;;; Query results:

(list-ref (she loves you yeah yeah yeah) (s (s 0)) you)

;;; Query input:

(list-ref (she loves you yeah yeah yeah) ?num yeah)

;;; Query results:

(list-ref (she loves you yeah yeah yeah) (s (s (s (s (s 0))))) yeah)

(list-ref (she loves you yeah yeah yeah) (s (s (s 0))) yeah)

(list-ref (she loves you yeah yeah yeah) (s (s (s (s 0)))) yeah)

Hint: If you match a number to (s ?x), ?x will be that number minus 1.

(assert! (rule (list-ref (?car . ?cdr) 0 ?car)))

(assert! (rule (list-ref (?car . ?cdr) (s ?rest) ?result)

(list-ref ?cdr ?rest ?result)))

Rubric:

4 points - correct

3 points - missing base case, but otherwise correct

2 points - Tries using a query match as if it was a function

1 point - shows some understanding

0 points - completely incorrect

1. (2 points) Recall the code for the reverse rule:

(assert! (rule (reverse (?a . ?x) ?y)  
 (and (reverse ?x ?z)  
 (append ?z (?a) ?y) )))  
  
(assert! (reverse () ()))

This rule cannot “run backwards”. For example, the below query results in an infinite loop.

;;; Query input:

(reverse ?what (a b c))

Why does this infinite loop? Be as specific as possible (trace through the steps!).

* The system finds that the reverse rule is applicable; that is, the rule conclusion (reverse (?a . ?x) ?y) successfully unifies with the query pattern (reverse ?what (a b c)) to produce a frame in which ?a is bound to the car of ?what, ?x is bound to the cdr of ?what, and ?y is bound to (a b c). So the interpreter proceeds to evaluate the first part of the rule body (reverse ?x ?z) in this frame.
* At this point, we need to match (reverse ?x ?z), where none of these variables have any real bindings. But to match this rule, we need to match another call to reverse...we never make any progress since there are no meaningful bindings.

The system is now in an infinite loop.

**Rubric:**

2 points - correct

1.5 points - has the right idea, minor erros

1 point - has the big idea picture but does not explain how the matching works

0 points - Completely off, shows little to no understanding