***Interpretation of the cleaning-robot program***

This is a rule:

|  |  |
| --- | --- |
| at(P) :- pos(P,X,Y) & pos(r1,X,Y). | P, X and Y are variables and r1 is a constant. When this executes - will AgentSpeak fetch the value for pos(r1,X,Y) where X = 0 and Y = 0.  P gets the value r1 |

This is the goal:

|  |  |
| --- | --- |
| !check(slots). | this sets the goal – the one that we are going to accomplish (!) - slots could have another name on this constant, maybe square could be more expressive |

These are the plans:

|  |  |
| --- | --- |
| +!check(slots) : not garbage(r1) | this is a plan that will put check(slots) in the memory.  If this plan does not stumble on any problem the check(slots) will be put in memory (by the head +!check(slots) ). +!check(slots).                         puts check(slots) in the memory. |
| <- next(slot); | <- this is a body that will execute until the dot  next(slot) – note that slots and slot are two different names and thereby two different constants (s differs). The next(slot) cannot be change because it is coded in the Java-part |
| !check(slots). | The last means that !check(slots) is what will be accomplished and is also a call for check, a so called recursion. |

|  |  |
| --- | --- |
| @lg[atomic] +garbage(r1) : not .desire(carry\_to(r2)) | .desire -    checks if carry\_to is the same position as the one that is in the memory (a condition for the predicate). If carry on with the moving the garbage |
| <- !carry\_to(r2). | This means that the goal !carry\_to(r2) will be accomplished and is also a call to carry\_to with the constant r2. |

|  |  |
| --- | --- |
| +!carry\_to(R) | R becomes instantiate with r2 from the call |
| <-                 ?pos(r1,X,Y); | fetch position for r1 |
| -+pos(last,X,Y); | minus-sign in -+pos(last,X,Y) means that values in r1, X and Y are released to be able to add new values and position-call with last is getting the value of r1 position  (this means substitute)   last- is used to know where to go when the when the agent has moved to r2 |
| !take(garb,R); | take will take the garbage to r2 (- see more info about take below) |
| !at(last); | at is used to get the agent to go back to the position last - see  -+pos(last,X,Y); |
| !check(slots). | Means that !check(slots) will be accomplished and that it is a recursion since it calls itself. |

Unifying is carried out in two ways – either the variables must be in the same predicate (that is in the head of body before) or you get in by the call

For example - take calls take with garb and R (from the above mentioned predicate) – remember that R is instantiated - see above +!carry\_to(R)             
      !take(garb,R);         - garb is a constant and is coded in the Java-part. By garb you also have the coordination to R, r2.

|  |  |
| --- | --- |
| !take(S,L) : true | Here becomes S = garb and L = R (from the call)  - the condition is true |
| <- !ensure\_pick(S); | call to ensure\_pick with S that we have set to garb. Ensure\_pick is the goal that is going to be accomplised and a call to ensure\_ pick |
| !at(L); | after ensure has been executed – call the at with L, which is r2. Here at uses the first rule up in the top, first.  If not the same place, do the at in the bottom |
| drop(S). | drop garb |

|  |  |
| --- | --- |
| +!ensure\_pick(S) : garbage(r1) | S = garb from the call above  The condition is garbage(r1)  And if the predicate is fulfilled, ensure\_pick(S) will be added |
| <- pick(garb); | Pick garb is coded in Jave- part |
| !ensure\_pick(S). | means that ! ensure\_pick (garb) will be accomplished and that it is a recursion since it calls itself. |
| +!ensure\_pick(\_). | This add ensure\_pick. Note the (\_) which means that this is without value. (\_) is used to get the same number of variables in to ensure\_pick(S) |

Execute the program, preferably, with debugging feature!