Artificial Intelligence Assignment 1 Readme Document

Author:
Joshua Wang
UNI: jw3123

1. The programming language in the project is Lisp.
2. The version of programming language is Common Lisp.
3. Development environment is:
   Application: LispWorks
   Operating System: Windows 8
   Processor: Intel i7-4702MQ
   Memory: 8GB DDR3
4. How run and compile on LispWorks:
   (1) Open LispWorks.
   (2) If there is no listener, click Tools->Interface->Listen. A listener's window will appear.
   (3) Click on listener's window. Then click File-> Compile and Load. Select the file jw3123.lsp. The file will be complied and loaded.

5. The procedure to compile and execute the program:
   (1) After compiling, press space or click the Listener tab in the Listener window.
   (2) In the listener, it will appear CL-USER 1 >
   (3) Type the instruction. That is, match "pattern" "data". For example, type:

```
match '(?x ?y) '(1 2)
```

   The program will return the association list

```
((?Y 2) (?X 1))
```

6. A top- down explanation of how the program functions:

(1) The function *match "pattern" "data"* has a sub-function *subpm "pattern "data" "resultant list of association lists"*. *Subpm* is designed to return every possible binding lists. I.e. return a list of association list. Function *match* handles the return value of *subpm*. Possible situations are:

(a) The list of association lists is nil, then return nil.

(b) The list of association list contains a T. This is checked by function *hast "association list"*, by recursively checking if at least one element in the list of association list is T. If there is, return T, else return nil.

(c) By checking whether the second element of the list of association list is nil, the function can know whether there is only one association list. If this is the case, return the first element of that list, which is the only association list.

(d) By default, which should be the case that there are multiple association lists, return that list of association lists.

(2) The main structure of *subpm "pattern "data" "list of association lists"* is a conditional statement. There are total 11 cases, illustrated as follows:
(In the following, p stands for the list of pattern, d for the list of data, a for an association list, (a) for list of association lists)

(a) If p and d are nil simultaneously, which implies a successful match. In this case, if (a) is not same as input, which is (( )), it means there is at least an association list, thus (a) is returned. Else (t) is returned.
(b) Before (c), we must check whether car p is * when d is nil. In this case, we advance p to the next element.
(c) If either p or d is nil but the other is not, the match fails and return nil. (Though we use OR, it cannot be both p and d are nil because that case is satisfied in case (a))
(d) If car p is ?, since ? matches everything, simply pass cdr p, cdr d, a, to the next recursion of *subpm*.
(e) The *isv* function checks whether car p is a variable. (illustrated in section (3)). If it is, update (a) by calling *matchvar* (illustrated in section (4)) and pass the new (a) with cdr p, cdr d, to next recursion of *subpm*.
(f) If car p is a *, append the returned (a)s of advancing p and d, respectively.
(g) The *isex* function checks whether car p has a "!" at the front. (illustrated in section (3)). If it is, update (a) by calling *exclude* (illustrated in section (5)), with the first symbol of car p (= "!") changed into "?", for the convenience of processing. Then pass the new (a) with cdr p, cdr d, to the next recursion of *subpm*.
(h) The *isgreater* function checks whether car p has a ">" at the front. (illustrated in section (3)). Also check whether car d is a number. If these conditions are satisfied, update (a) by calling *greater* (illustrated in section (6)), with the first symbol of car p (= ">") changed into "?", for the convenience of processing. Then pass the new (a) with cdr p, cdr d, to next recursion of *subpm*.
(i) The case of encountering "< "is similar to case (h), except that the *isgreater* function is replaced by *issmaller*, *greater* by *smaller,* and ">" symbol by "<".
(j) When encountering a list. It might be a "constraint case", with & at head and several variables, correspond to one same datum. Or sub-list case, which does not have &, and different variables correspond to different data. i.e., the same as we are

doing in the *subpm* function. In the constraint case, we call function *bindhandler* (illustrated in section (7)) , with car (car p) and car d and (a), to update (a), then pass new (a), with cdr p and cdr d to the next *subpm* recursion. In the case of sub-list, we call subpm, with car p and car d and (a), to update (a), then pass the new (a), cdr p, cdr d, to the next *subpm* recursion.

(k) By default, if car p does not match any condition above, we assume it is a symbol. Thus if it is the same as car d, then pass cdr p, cdr d, and (a) to the next recursion. Else return nil.

(3) *isv x, isex x, isgreater x, issmaller x* : check whether x has "?"," !"," >","<" before some symbol, respectively. They first check whether x is a symbol, then whether they have the corresponding symbols ("?"," !"," >","<") at the front of x, then whether the rest part of x is not empty. Return t if all of these conditions are satisfied, else return nil.

Note: this part of code is referred from notes from the previous course of AI:
URL: http://wsh4346.googlecode.com/svn/trunk/AI/proj1/trunk/sw2583.lisp

(4) *matchvar carp card* (a) (where carp and card refer to car p and car d). Initially If (a) is (()), it means this is the first time the program encounter a binding, so we bind carp and carp, and return (((carp card))). Else it inspects every a in (a). If carp is not seen in this a, add (carp card) to the current a. Else if (carp card) accords with previous binding in a, it retains this a by merging this a, with the return value of following recursions, Else only return the following recursion, which in effect deleted the current a.

(5) *exclude carp card (a),* works conversely as *matchvar* in that it deletes the a when discovering a binding, and retains a if card is not equal to the datum carp is binded with. We set the return value nil if carp is unseen. (Note that carp is modified as a variable before inputting. E.g., from !x to ?x.)

(6) *greater carp card (a)* (or *smaller carp card (a)*) work similar to *exclude*. They retain the a if card is greater (or smaller) than the datum carp is binded with, and delete the a otherwise.

(7) *bindhandler carp card (a)* (where carp is actually a list) recursively test whether each element in carp satisfied the relation with card, and update (a) correspondingly. Four possible relations' signifiers are *isv, isex, isgreater, issmaller,* and the

corresponding functions to update (a) are: *matchvar, exclude, greater, smaller.*

Reference:

1. The overall structure of the project is inspired by the note on the website of the course.

URL: https://www.cs.columbia.edu/~jvoris/AI/notes/assignment_1/assignment_1.htm

2. Syntaxes reference:

   (a) Slides in the lecture:

   URL:https://www.cs.columbia.edu/~jvoris/AI/notes/LISP_at_high_Speed.pdf

   URL: https://www.cs.columbia.edu/~jvoris/AI/notes/m3-search_01.pdf

   (b) The Common Lisp Cookbook.

   URL: http://cl-cookbook.sourceforge.net/index.html

3. Functions of *isv* , *isex, isgreater* , *issmaller :*

   Notes from previous class of AI:

   URL: http://wsh4346.googlecode.com/svn/trunk/AI/proj1/trunk/sw2583.lisp