Programing Directions for Assignment 2

CSC486/2506 - Knowledge Representation and Reasoning

Fall 2006

1 Platforms to be used

The platforms to be used are the same as the ones used for assignment 1.

2 Input Format

In Question 4, you should write normalization and classification procedures for building a concept hierarchy incrementally. Here, we will discuss a uniform way of representing the input of your program. You **must** use this representation.

First, concepts, roles and primitive descriptions are represented with atom names in *small letters* such us `thing, child` and `adult` respectively. Complex descriptions are built using lists where the first element is either atom `all` or atom `and`. Notice that all these atoms are in small letter because atoms in capital letters are variables in Prolog.

An input file will consist of set of pairs (C, D) where C is a concept name and D is its corresponding description. Each pair is represented as a list with its first element being the concept name (i.e., an atom) and the second element its corresponding description. The idea is that your program will take each pair (C, D), it will normalize D, and it will finally classify it into the (current) hierarchy. We shall use `(end)` and `[end]` to mark the end of the input file for Scheme and Prolog respectively. Examples of legal input files for both Prolog and Scheme are shown in figure **??**.

If you do not use Prolog or Scheme you should choose one of the two representations and state which one you use in your documentation.

3 What your program should provide

If you use Prolog or Scheme we provide template files (`classify.pl` and `classify.cl` respectively) which already have the code necessary for reading a set of concept-description pairs from a file. These two files define a top-level procedure called `buildHierarchy` whose only argument is the name of a file containing a set of clauses.

Procedure `buildHierarchy` reads one pair at a time and calls `classify` for each pair. Procedure `classify` should take the pair, normalize the corresponding description and introduce it into the current hierarchy. **It is this procedure `classify` that you have to implement (by modifying the template file) together with two extra procedures to show the results: `printConcepts` and `printHierarchy`.** Procedure `printConcepts` should print each known concept in the hierarchy together with its corresponding normalized description; and procedure `printHierarchy` should show the hierarchy by printing out a list of messages of the following form: "`Concept X is directly below concept Y`" for each known concept `X` (except for concept `thing`). For the example given in figure **??**, among other things, procedure `printConcepts` should print something like "`Concept c1 is adult`" (assuming `adult` is the normalized form of desription `(and adult)`); while `printHierarchy` should show the following:

```
Concept c1 is directly below concept thing
Concept c2 is directly below concept c1
Concept c3 is directly below concept c1
```

```
(c1 (and adult))
(c2 (and adult male))
(c3 (and adult female))
(c4 (and adult male rich))
(c5 (all child adult))
(c6 (all child (and adult female)))
(end)
```

(a) Scheme representation

```
[c1,[and,adult]].
[c2,[and,adult,male]].
[c3,[and,adult,female]].
[c4,[and,adult,male,rich]].
[c5,[all,child,adult]].
[c6,[all,child,[and,adult,female]]].
[end].
```

(b) Prolog representation

Figure 1: Two files containing a set of concept name-description pairs.

```
Concept c4 is directly below concept c2
Concept c5 is directly below concept thing
Concept c6 is directly below concept c5
```

In summary you should modify the template files by implementing three procedures: (i) `classify`, (ii) `printConcepts` and (iii) `printHierarchy`. Notice that the printing procedures are important because after classifying all concepts, procedure `buildHierarchy` will call both of them to show the results of your program. Also, by implementing yourself these two procedures you are free to use any representation you want to store your hierarchy incrementally.

If you are using Java or C, your program itself should be named `buildHierarchy` and it should take as its only argument the name of the file where it should read the set of concept name-description pairs. As there is no template file provided for these programming languages, you are responsible for programming the task of reading the pairs from a file as well as solving the classification problem for the set of concepts. After classifying all concepts, your program *should print the same information* as `printConcepts` and `printHierarchy` do. Also, remember to explicitly say in your documentation the *exact commands* used for compiling and running your program in either CDF or CS.

Finally, you should include with your program a small set of concepts with which you have tested your program for building hierarchies. At least provide one test example of a hierarchy with 10 (ten) concepts.

4 How to submit your program

You should submit your code and test files to `jabaier -AT- cs.toronto /dot/ edu` as an attachment file.

No matter what platform you choose, remember that your program has to run error-free on one of the CDF or CSLab machines. So, if you worked elsewhere, we recommend you try your program on either of these machines before handing in the code.