CS5820 – Spring 2015 – Kaminski Assignment 1 – Robbie's World

OVERVIEW: Program contains 7 physically separate code files:

1) **dataBase** (DB) of facts about this specific, current state of the world (including robbie, table & objects)

2) **translator** rules to map KB & query rules (which specify **individual-property** predicates) to the DB facts' format (stated as **record-based** facts)

3) **knowledgeBase** (KB) of rules about how this general type of world works
- use my questions as a general guide to the types of concepts needed

4) **query** rules – Prolog queries which correspond to my English-language questions

*5) **questions** - **MY** English language questions – in Prolog-friendly format (part of userInterface (UI))*

*6) **runQuestions** - **MY** "local" controller (including part of UI) which automates running the questions & nicely formats the output*

7) **controller** (an expanded version of what I described as loadNGo)
- loads the above 6 files into Prolog
- re-routes all output to a file called Asgn1.txt (a text file) instead of (or besides) the screen output
- automatically starts running the questions (see runQuestions file for name of predicate)

NOTE: When naming files, use small letters to avoid the "capital means a variable" problem.

NOTE: I use "Robbie" in this document to mean either "the intelligent program" or "the hand you see in the picture". The context should make it obvious when I mean #1 or #2.

The DB of Facts

- include data about Robbie, the table and all of the objects
- data must be stored in record-based format (more like a traditional CS data file, but make it Prolog-friendly) rather than as individual property predicates (e.g., size, shape, madeOf, material, …)
- 2 types of facts:
  1) **static** (fixed) properties – things which can not change in this world – e.g., name, typeOfThing, color, size, material,…
  2) **dynamic** (relative) properties – a snapshot of this particular current state of the world – things which can change (e.g., if Robbie moved an object) – e.g., orientation, left, on
  [`left` means directly left of, with nothing in between & object is actually on the table
  `on` means directly on top of with nothing in between]
- Robbie and each of the objects have 2 facts: a static property fact and a dynamic property fact
  - the table just has a static property fact since it can not change
- Robbie and the table facts might have different number & order of parameters than the objects' facts do

The TRANSLATOR Rules

The KB & query rules never access the DB facts directly. They access translator rule-heads, which in turn have the translator rule-bodies access the relevant parameters in the DB facts.
Use predicates which specify property categories rather than property values – e.g., specify

```
size(Thing, TheSize) :-
        objectStaticProperties(Thing, _, TheSize, _).
```

rather than the less generic, less broadly usable:

```
big(Thing) :-
        objectStaticProperties (Thing, _, big, _).
small(Thing) :-         . . .
medium(Thing) :-        . . .
```

The DB vs. the KB

The DB is specific to 1) this particular world (static)
and 2) this current configuration of this world (dynamic) at this moment in time.
The KB is general, re-usable knowledge describing generic truths about how worlds like this one work, regardless of which particular objects exist and what their relative positions are now.
There must be KB rules about:
1) physics knowledge about how worlds of this general sort work,
2) vocabulary knowledge about the meaning of concepts for this type of world, including all terms which user queries might include, and
3) synonyms (e.g., `under` is the same as `below`) and antonyms (`below` is the opposite of `above`) which the user queries might use without having to re-phrase their question.

Where possible (and appropriate), store things as knowledge rather than data. It makes for a better system (more intelligent / robust / adaptable / able to learn / able to reason in new situations / . . .) if Robbie can use reasoning to infer (deduce) an answer to a question/goal (or a sub-goal) rather than looking it up as a fact. At this point we're not concerned that this may result in longer machine-run-time. At this prototyping stage we're more interested in system effectiveness and the ability to quickly add new knowledge (manually, or though Robbie generalizing and learning) than in execution-efficiency.

Do NOT put specific rules in the KB which combine concepts just because some question happens to ask about them together, e.g., hollowBlack things. A query should instead ask about each of the properties separately, e.g., things which are both hollow and black. In a future asgn, we'll want to automate the translating of English language questions into Prolog queries, and so have to deal with individual words as properties.

QUERIES & QUESTIONS: To test Robbie's apparent "intelligence", you'll give Robbie my exam. It consists of true/false questions and fill-in-the-blank type questions. (If there are several correct answers for the fill-in-the-blank questions, Robbie must give all answers that apply in order to get full credit). The completed program will be like a regular exam (i.e., batch processing) rather than an oral exam (i.e., interactive processing). However, during development of the program, you should interactively test Robbie on individual queries to make sure he's getting the correct answer for each. [However, unlike for real world exams: Robbie gets NO credit for just giving the right answer – he has to use the proper reasoning to get the answer! I, the grader, will look into what Robbie "knows" and how he reasons!]

I've supplied the natural language (English) questions (in file). But, since Robbie can't yet process natural language questions, the exam has to be translated (by you, as human programmer) into his native language, Prolog (i.e., query rules in the query file). Use wording in your query which closely mirrors the words and wording in my questions as if we had a machine translator program (which we might in the future) so we don't have to have you manually translate questions into queries.

To help Robbie do well on his exam (i.e., come up with the correct answers), use the specific meanings of words which I describe below and which we agree on in class.

There must be 1 query per question, appropriately numbered. But a single query might consist of 1 or more Prolog rules. The preferred format in Prolog is to use multiple rules rather than using OR's (;) in a single rule.

Also, ultimately, in the final demo version (which you hand in), the program code must write out the answer(s) itself to the file and not depend on Prolog to write out the answer (as it does interactively).