CPSC 312 Project 1 - Report
Prolog Expert System Shell

Name	Student ID	UGrad ID
Byron Duenas	34095117	v5e8
Tongli Li	15688112	w6d8
Brian Taylor	52311859	z2b0b

Question 1 - Domain Knowledge Base

Overview

We decided to create a knowledge base that would allow a user to identify a whale. Whales belong to the order cetacea, but there are two main suborders, odontoceti and mysticeti. Within each suborder there are families and then individual species, and within the balaenopteridae family there are two subfamilies. It was the hierarchical nature of whale classification that made this choice a good one. We did not create a comprehensive knowledge base using every species, but picked a few from each family.

We followed the style of rules in the bird knowledge base, for the most part. The nature of creating rules that would be turned into questions requiring a 'yes' or 'no' answer was a bit difficult. In some cases it would have been nicer to incorporate numbers (e.g. the number of blowholes, or the size of the whales), but this would not have fit into the schema. Distinguishing the two suborders was very easy (teeth or baleen, one or two blowholes), but we had to do a fair bit of investigation to try and determine simple qualities that distinguish each of the species we chose to include in the knowledge base.

Modifications

We modified `312-pess-grammar.pl` to include rules for nouns, adjectives, adverbs, and verbs in our new knowledge base so that loading the rules would function properly.

Examples

```
rule:
if   its family is physeteridae
then it is a "n:sperm whale".
```

Question 2 - Interpreter

Overview

We used Amzi's example of an interpreter shell to implement our interpreter loop[1]. It provided the basic command loop implementation that allows flexible pattern matching of commands to execute other prolog predicates.

From there on, we followed its pattern and extended the available commands according to the assignment. For each command, we first check some conditions and write some informational

[1] http://amzi.com/ExpertSystemsInProlog/02usingprolog.htm#asimpleshell

output to the user using a helper predicate before calling the main predicate that does the heavy lifting. For example, `solve.` checks that rules other than the "top_goal" have been set, otherwise it alerts the user that no rules are present and therefore it can't come to any conclusions.

In addition to the commands assigned in the project, we added an additional `clear.` command to clear the currently loaded rules as a way to improve the user experience. It made testing much easier, too.

Examples

```
?- main.
This is the CPSC312 Prolog Expert System Shell.
Based on Amzi's "native Prolog shell".
Type help. load. goal. solve. rule. list. clear. or quit. at the prompt.
Notice the period after each command!
> help.
Type help. load. goal. solve. rule. list. clear. or quit. at the prompt.
Notice the period after each command!
> |: quit.

true .
```

Bonus Attempted - "list" Command

The `list.` command is handled by the `list_rules/0` predicate, which uses the built-in predicate `current_predicates/1` to check that rule-predicates `rule/2` have been defined.

If they are, it then further checks that those rules do not contain "top_goal". If so, the `list_rules_exist/0` predicate will gloss and write as a sentence all rules that do not contain "top_goal" by backtracking, and end when the only rule remaining contains "top_goal".

If there are no rules predicates `rule/2` defined, or the `rule/2`s defined contain "top_goal", it will write "`No rules are loaded.`".

To use the command, type "`list.`" in the interpreter loop.

Question 3 - Specify The Goal

Overview

To allow the user to specify the top goal, we created `set_top_goal/1` which replaces the current top goal if there is one and calls `set_top_goal_helper/1`. Its single parameter is the current top goal in plain English. We created `set_top_goal_helper/1` which waits for user input, calls `set_top_goal/2`, and glosses the top goal for the single parameter with `plain_gloss/2`.

We created `set_top_goal/2` which uses existing grammar rules, and grammar rules that we've added to turn the user's sentence into a new top goal.

Modifications

We modified `312-pess.pl` in the following ways:

- We changed "top_goal" to return the "attr" structure, rather than just a single atom belonging to the top goal.
- Solve now uses `plain_gloss/2` to convert the "attr" structure of "top_goal" into a proper sentence.

Examples

After manually adding "goal: does it eats insects." to the file `bird.kb`, the `solve.` command will solve for the user specified goal after the knowledgebase is loaded, as follows:

```
> load.
Enter file name in single quotes, followed by a period: |: 'bird.kb'.
Understood:
if it has external tubular nostrils and it lives at sea and it has hooked bill then
it has order that is tubenose
...
if it is brown and it is swan then it is brown swan
Rules loaded.
> |: solve.
Would you say that it eats insects ?> |: yes.
The answer is: it eats insects
```

One limitation of our system is that it cannot handle "eat", which is why the goal had to be written with "eats" since this feature only works for words that are defined within a given knowledge base.

Bonus Attempted - "What the heck is THAT"

The interpreter will understand the goal "`what the heck is THAT.`" as "what is it" based on pattern matching.

To use the command, type "`goal.`" in the interpreter loop, then enter "`what the heck is THAT.`" as the goal.

The shortcoming of this method is that it can only parse the above mentioned specific words; however, we handle any variation of capitalization by converting all input to lowercase.

Question 4 - "goal" Command

Overview

This was easy to implement after completing question 3 because we simply had to pass the input value to be parsed by `set_top_goal/1`.

Modifications

However, we did have to modify our implementation of `set_top_goal/1` from question 3 to now pass up the new top goal in plain english text because question 4's requirements required us to inform the user of the new "top_goal".

This required `set_top_goal/2` to pass up the top goal in its "attr" structure, `set_top_goal_helper/1` to gloss the the top goal with plain_gloss, and for "top_goal" to have its single parameter return the new top goal in plain english text.

Examples

Example goals:

```
what is it
it is a hooked what
is it a long "n:pygmy right whale"
does it feeds
it feeds "n:in coastal waters"
```

Setting a new goal and solving for it:

```
> load.
Enter file name in single quotes, followed by a period: |: 'whale.kb'.
Understood:
if it has teeth and it has one blowhole then it has suborder that is odontoceti
...
if it is pointed then it is hooked fin whale
if it is pygmy right whale then it has tail
Rules loaded.
> |: goal.
Enter the new goal, followed by a period: it is a hooked what.
Understood goal: it is hooked what
> |: solve.
Would you say that it is pointed ?> |: yes.
The answer is: it is hooked fin whale
```

An alternative example:

```
> rule.
Enter a new rule, followed by a period: if it is brown and it is a swan then it is a
brown swan.
```

```
Understood rule: if it is brown and it is a swan then it is a brown swan
> |: goal.
Enter the new goal, followed by a period: it is a brown swan.
Understood goal: it is a brown swan
> |: solve.
Would you say that it is brown ?> |: yes.
Would you say that it is swan ?> |: yes.
The answer is: it is a brown swan
> |: goal.
Enter the new goal, followed by a period: it is a brown what.
Understood goal: it is brown what
> |: solve.
Would you say that it is brown ?> |: yes.
Would you say that it is swan ?> |: yes.
The answer is: it is brown swan
```

Question 5 - "rule" Command

Overview

To allow the user to add rules to the knowledge database, we read their input, added new words encountered that were not in the vocabulary, and processed the input in a list that corresponded to the rule format. To tell the user we understood their rule, we wrote out the parsed rule into the prompt.

The `rule.` command is handled by the `add_rule/0` predicate, which uses `read_sentence/1` to get the user's input.

It checks that all of the words in the input have been previously added to the vocabulary using the `add_unknown_words/1` predicate, and then adds the words to a list of words of the format `['rule:',words,in,the,input,rule]` to be processed by the provided `process/1` predicate, which uses the unchanged clause originally provided to us.

Examples

Adding a new rule and solving for it:

```
> rule.
Enter a new rule, followed by a period: if it has teeth then it is a shark.
Understood rule: if it has teeth then it is a shark
> |: list.
if it has teeth then it is shark
> |: goal.
Enter the new goal, followed by a period: what is it.
Understood goal: it is what
> |: solve.
Would you say that it has teeth ?> |: yes.
The answer is: it is shark
```

There are some shortcomings to our implementation of this. Vocabulary that is new must be hard-coded by the user as he or she writes the rule. For example, by using the syntax of putting the new word or phrase in double quotes, preceded by the part of speech (e.g. `“n:stellar’s jay”`).

Question 6 - WordNet / Morph

Overview

Using the `add_unknown_words/1` predicate, we can handle words that are not defined in the vocabulary. The predicate does this by calling `add_word/1` on the entire list of unknown words. `add_word/1` will add a word if it's not already added. It will then check if it is a valid noun, adjective, adverb or verb. It does this by calling `noun/1`, `adjective/1`, `verb/1`, and `adverb/1`.

The predicates `noun/1`, `adjective/1`, `verb/1`, and `adverb/1` all morph the word using `morph/2`. The predicate `morph/2` calls `morph_atoms/2` in `pronto_morph_engine.pl` to give us all the possible morphs of a given word just in case the original word is not in WordNet (plurals for example). Then `noun/1`, `adjective/1`, `verb/1`, and `adverb/1` will assert `v/1`, `n/1`, `adj/1`, and `adv/1` rules for each valid word and its corresponding morphs using `assertz/1`.

We chose to use expand the vocabulary this way so that we only process (their part-of-speech and assertions) WordNet words into the vocabulary database when we encounter a previously undefined word, instead of all at once. We made this decision because we wanted to dynamically create rules for words that we are currently using and not initially load the entire database of words at initialization.

Examples

Adding a new rule that uses vocabulary not contained in the `312-pess-grammar.pl` file:

```
> goal.
Enter the new goal, followed by a period: what is it.
Understood goal: it is what
> |: rule.
Enter a new rule, followed by a period: if it has a brain and a soul then it is
human.
Understood rule: if it has a brain and a soul then it is human
> |: list.
if it has brain and it has soul then it is human
> |: solve.
Would you say that it has brain ?> |: yes.
Would you say that it has soul ?> |: yes.
The answer is: it is human
```

Shortcomings

General Limitations

There are some more general shortcomings of our system. It is set up only to prompt the user with yes or no questions. Ideally, an expert system would be able to ask questions such as “What colour is it?” or “What is its mass?”. A feature that we toyed with implementing was a menu system where the user would be given a choice of suitable answers to a question. For example, to answer the question “How many blowholes doe it have?” the user would be prompted with a menu listing the choices “One”, “Two” and then would enter one of the choices.

Another limitation of our expert system is the black and white nature of the identification system. If we were trying to design a diagnostic system we would have had to implement some sort of likelihood function so that after answer a series of questions the user would be presented with a list of possible diagnoses in declining likelihood.

Goal Matching

There are definitely some shortcomings with our implementation. For example, it will only handle goals of a certain type in a certain format because goal setting is based on pattern matching. If the goals are not written in a prescribed way the results are somewhat arbitrary as the following example shows.

Known scenarios: if a user writes a goal such as "what is that", the system will not know what "top_goal" to to match the input with.
Attempt to fix: we attempted to determine how rules can be written in various ways, but was unable to come up with a method to generalize differing rules into specific ones that the system can understand and solve for.
Result: unfixed.

An example of erroneous parsing of a goal:

```
> goal.
Enter the new goal, followed by a period: what is that.
Unable to understand goal, goal set to "what is it".
> |: goal.
Enter the new goal, followed by a period: what does it do.
Understood goal: attr(does,do,what)
> |: rule.
Enter a new rule, followed by a period: if it is human then it do things.
Understood rule: if it is human then it do things
> |: solve.
Would you say that it is human ?> |: yes.
The answer is: it do things
```

Words Outside Of Vocabulary And WordNet

One shortcoming has to do with vocabulary. If a user enters a word in a rule that is not recognized it will cause a parsing error.

Known scenarios: if a user writes a rule such as "If it eats worms then it is a wormeater.", the system will not know what to do with 'wormeater'. Instead, the user has to enter the novel word as "n:wormeater", thereby adding it manually to the dictionary. Additionally, a noun phrase might have to be entered similarly in double quotes, for example "n:two arms".
Attempt to fix: the scenario for unknown words could possibly be fixed by changing `add_word/1` to add a new word when it exhausts morphs/WordNet, however we are leaving it as is because it mostly works.
Result: unfixed.

"load." Command With Non-existent File.

Known scenario: the entire interpreter errors out of execution if the filename given does not point to a file.
Attempt to fix: investigate how see/1 works, and how error handling in Prolog works. However due to time, we did not implement error handling for this issue.
Result: unfixed.