CS312 Prolog Project Report

1. **New Knowledge Base:**

Looked up different dog types organized by their traits. This information was then encoded in the dog.kb file, following closely the patterns in bird.kb.

1. **Interpreter Loop:**

The Interpreter Loop was implemented following Amzi’s user interface design. A user begins the program (after consulting ‘312-pess.pl’) by entering ‘main.’

From there, a greeting is provided (constructed using the write/1 and nl/0 commands) that instructs the user of the available commands that they can enter. The design of the interpreter loop is centered around a repeat/0 and do/1 pattern where additional commands can easily be added in a modular fashion. This is done by creating the necessary function (ie goal) and creating an associated do(goal):- goal, !. near the user interface so that it calls the correct function when those are entered at the command line.

Lastly, if none of the commands match that which was entered by the user, an error will be written that ‘X is an invalid command’ where X is the command that the user entered.

1. **Allow user to specify the goal in knowledge base**

We needed to add a new process/1 function (a list argument with the first element as ‘goal:’) and then passes the remaining elements of the list into goal\_parse/3. Our goal\_parse function parses the goal by passing it to the parse\_goal/1 function which parses it according to it’s sentence structure (see comments in the file). From there, it can build the goal up and then it sets it as top goal.

\*\*Bonus: What the heck is that:

When you pass in the goal: “what the heck is that”, it parses the property that the first element is ‘what’ and second element is ‘the’ and so on for all the words. This then builds the goal by passing it through build\_goal/1 which copies the result of ‘what is it’ and assigns that as the top goal. It can then solve the goal from there.

1. **Allow user to set up top-level goal from interpreter prompt**

We created an additional interpreter procedure called ‘goal’ which, when the user enters this at the prompt, will then request a new goal to solve. The goal that’s entered is then read and the sentence is transformed into an atomic\_list which can they be passed into process/1 with the first element of the list as ‘goal:’. In process/1, we also added the retractall/1 procedure to remove any previously added goals.

Whenever you run ‘main’ the default top goal is assigned then so that if no goal is specified by the user, it uses that for solving.

1. **Assert new facts/rules at interpreter prompt**

We built an additional command for the interpreter called ‘assert’ which, when entered by the user will then call a new function, assertf/0, which requests a rule/fact from the user to add it to the database. Atomic\_list\_concat/3 is used for breaking the words down into individual words and then it is processed based on the ‘rule’ pattern.

1. **Use WordNet and Pronto\_Morph to increase vocabulary**

After ‘assert’ command is called from the user interface with a new fact/rule to be added, the system will first check each word if it’s in the database. If so, it won’t add it again and will skip that word. Otherwise, it will call get\_stems/1 which then calls morph\_atoms\_bag which will return the list of stems of the word/s that were passed in. This list will then be flattened and passed into check\_word/1 which then checks each word for it’s base word in Wordnet (check\_in\_wordnet/1). If the word is in WordNet, it will assert the word into the database using the word itself and it’s type (which was obtained from WordNet).

Ie. Get\_stems([flies]).

Since ‘flies’ is not in the database originally, after that function is called, it will break the word into all it’s stems (fly, flies, etc) and after checking in WordNet, will add n(flies) to the database.