Exercises for the Course

Logic Programming Engineering

(Author Dr. W. Nauber)
Dr. Paola Bruscoli, Peter Steinke, Amin Timany

Practical 11

Built-in Predicates II (I/O, dynamic features, memory)

Exercise 11.1

Write a Prolog program for the following game of guessing words:

- The aim of the game is to guess an unknown word with a minimal number of guesses.
- The player sees at the beginning of the game only the char * for each letter in the word to be guessed.
- At each round of the game the player has to guess a letter of the word: If this letter is present in the word, it will be shown in all correct positions in the word instead of the char * at all correct positions of the word.

The unknown word is automatically chosen from a database. The database can be read from a file, modified by adding a new word, and saved to a file.

For the control of the game, realize the following menu:

```
G u e s s  a  W o r d
---------------------

  l - list database
  r - read database
  a - add a new word
  s - save database
  w - guess a word
  e - end the game

Command:
```

An example of a session for a game is the following (obtained by giving the Command: w):

```
Please guess the word: ***
Please guess a letter: a
Your solution:         *a*
Please guess a letter: t
Your solution:         *at
Please guess a letter: c
Your solution:         cat

Congratulations!
```

Hints for realization in Prolog:

1) Copy the file *guesword.pl* from the LPE homepage to your home directory to save time during this practical session.
This part of program implements the shell, i.e. it produces the interface provided by the menu and understands the ending command of the game. You can use this program as a preliminary template and further modify it to accomodate the following steps of the activity.
REMARK: There are some new built-in predicates, read about them:

get_single_char, *put*, *name*

2) The file *words.pl* (again, available from the LPE homepage) is an example of database of words to be guessed, where words are encoded as facts $w(word)$. (You may refer to it or define similarly another database).
We need to define the predicates that allow to make the system interact with the database:

- Write a predicate *init_database* which loads a database from the file *words.pl*.
  REMARK: To erase all old facts $w(\_)$ in the memory, you may use the built-in predicate *abolish* (read about that).
- Define an auxiliary predicate $readfile(Name)$ which reads the clauses (facts) from the file $Name$ using the built-in predicates $exists\_file$ and *consult*.
- You shall initialize the database before you output the menu!
  For SWI-Prolog with version 3.4.0 and higher it is necessary to declare the predicate w (before reading it) as dynamic, by using the built-in predicate *dynamic*: the effect of this declaration is the possibility of changing the definition of our database during execution. If the file *words.pl* does not exist, please initialize the database with words at will, such as *cat* and *mouse*, using the built-in predicate *assert*.

REMARK: Several new built-in predicates are referred to in this part, read about them, they are mostly for managing I/O, memory management, modyfying declarations at run time, adding statement (and dually, removing) from a database:

abolish, $exists\_file$, *consult*, *dynamic*, *assert* (*asserta*)

3) To list a database which is in memory, you can collect all words in a list using built-in predicate $bagof$.

```
e.g.: database: [house, paper, radio, room, music, computer, pencil]
```

4) To read a new database, make the interface prompt for a file name, and then read the database using the predicate *readfile* you had from step 2). As an example for a different database, you may find the file *animals.pl* in the LPE homepage. The prompt may look like this:

```
filename of database (e.g. 'animals.pl'):
|: 'animals.pl'.
```

HINT: Use " in a Prolog string to get ' as an output.

5) Use the built-in predicate *asserta* to add a new word to the database in memory; the prompt may look like this and the new word is added in the database:

```
new word (e.g. dog.):
|: dog.
```

6) <u>To save a database in a file</u>, use the following built-in predicates:

tell, *listing*, *told*

You may have an interface then looking like this:

```
filename of database (e.g. 'animals.pl'):
|: 'animals.pl'.
Writing database to animals.pl ...
ready
```

7) This part has to do with guessing words and guessing letters in a word.

a) When you try to guess a word, select always the first word of the database (using $w(Word)$). For a more sophisticated task see 8).
Write an auxiliary predicate $genstartword(N, SWL)$ which generates a list SWL of length N of numbers 42 (that is the ASCII-code for the char '*').
Convert this list into a name using the built-in predicate *name*, and write it as the starting word in the screen.

b) To guess letters of the word in a recursive way, use a predicate $guessletters(WL, GWL)$, where WL is a list of ASCII-codes of the word to guess and GWL a list of ASCII-codes of the actual guessed word.
At the beginning, GWL is initialized with SWL. The process of guessing terminates when WL equals GWL.
Within the predicate *guessletters*, you shall prompt for the code of a new letter, and then use the predicate $newguessedword(WL, GWL, Letter, NGWL)$ (see part 7c)) to create the new list of ASCII-codes of guessed word $NGWL$.
If a letter was correctly found, then replace the code for '*' in the list GWL with the code of the new letter $Letter$. The new guessed word is then output, and you may start with the guess for the next letter.

c) To define the predicate *newguessedword*, you have to consider 3 cases:
1. lists WL and, GWL are empty,
2. the letter $Letter$ equals an actual element (head) of the list WL,
3. the letter $Letter$ is different from an actual element (head) of list WL,

Case 1. can be handled by defining a fact; for the other cases generate the actual element (head) of $NGWL$ and start with <u>the tails</u> of lists in a recursive way.

8) A more sophisticated way for selecting a word to guess, can prompt for its number K in the database (its index). To get the K-th word of the database, create a list LW of words using the built-in predicate *bagof*, and then use a predicate $kelem(K, LW, WK)$ to find the wanted word WK.
If the input was the number 0, the computer has to generate randomly an allowed number: you may try to use the built-in predicate *random* (or *cputime* and *truncate*)).

```
e.g.: Select a number (0.-7.): 5.
```