Programming Assignment – Zootson

(MAT-75006 Artificial Intelligence 2014)

Instructions, Version 1.2

Update history of the instructions:

Feb 11, 2014	Version 1.2: Text added concerning the needed vocabulary. Python version corrected. Figure illustrating the grading process added. Some minor clarifications.
Feb 5, 2014	Version 1.1: the answer format table was modified and animal class names added.
Feb 4, 2014	The instructions were published.

General

Getting the programming assignment described in this document accepted is a mandatory part of passing the implementation of the course MAT-75006 Artificial Intelligence lectured in the spring 2014. The assignment consists of programming – within the guidelines and requirements presented in this document – a simple "expert system" capable to answer questions asked in natural language and documenting the resulting implementation.

The assignment is to be carried out individually; any coding co-operation, code passing or code exchange is NOT allowed. It is forbidden to use code not written by yourselves (excluding libraries, modules etc. used for "basic functionalities of a programming language"). All the "real work" ought to be made by your own code, not some fancy, already existing component.

You do not need to register for the assignment anyhow. There are no intermediate checks of your progress or work status, so you must take care of starting the planning and implementing early enough by yourselves. For getting your work evaluated it is enough to submit it as instructed.

You can implement your system wherever you like and using the programming language of your choice. However, as it should be easy to get your code interpreted or compiled for the evaluation, the language versions, libraries, etc. are restricted to those found in the *korppi* cluster (korppi.cs.tut.fi) of Birdland (Lintula). For instance, Python 2.6.4 could be a good choice for the language, and before submitting your work you should verify that your code works in the *korppi* environment. If you do not have a Birdland account, you can try to get one by following the orders here.

Naturally, it is possible that some important matters are accidentally missing from these instructions. If you think that something that is not mentioned should be included, or if you find contradictions, please notify teemu.heinimaki@tut.fi for getting possible errors and omissions corrected – preferably as soon as possible.

Changes for these instructions can be made during the course, should a need arise. Hopefully there will not be any need for major changes, but you should check for possible updates regularly.

Zootson

The Zootson software you are to implement is to be a natural language–processing expert system (although only with rather limited capabilities and expertise). The idea is that Zootson should be able to read a text file containing simple questions in plain English (one question per line), and to generate another text file containing the corresponding ("right") answers based on its knowledge.

Sources of Knowledge

The basic knowledge of Zootson is to be taken from the 'Zoo' dataset from UCI Machine Learning Repository [1]. It (and its description) can be downloaded from here: http://archive.ics.uci.edu/ml/datasets/Zoo. The possible 'type' attribute values, ranging from 1 to 7, can be mapped to animal class terms 'mammal', 'bird', 'reptile', 'fish', 'amphibian', 'insect', and 'invertebrate', respectively.

As additional data, we offer two other text files:

File	Description
continents.txt	In csv-formatted continents.txt each line represents one animal. The first value of each line corresponds to an animal name in 'zoo.data' (the actual data file of the 'Zoo' dataset), and the following values are associated to different continents in the following order: Africa, Europe, Asia, North America, South America, Australia, and Antarctica. Value '1' means that there are animals of the species/type under scrutiny living (in most cases indigenously or as major invading species) on the corresponding continent or its vicinity, and the value '0' means that this is not the case. Some approximations, guesses, simplifications and more or less arbitrary decisions had to be made when composing the data, for several reasons. For instance, the set of animals was rather nonhomogeneous, some terms ("animal names") representing a multitude of taxonomical families and even suborders, and others only a single species). The terms are also used differently in the different areas. Also the model of seven continents used does not fit perfectly for our needs (but it was chosen to be used, for the other common models also have their own problems). E.g., in our dataset, New Zealand is considered to be a part of the continent of Australia, as Australia happens to be the nearest continent. (Therefore kiwis and tuataras are considered in this case to live in Australia...) These problems demonstrate that dealing with natural language and imperfect existing models can be tricky. However, for the purposes of this exercise, the data can be assumed to be valid and "right". (You can, though, suggest corrections, if something seems to be terribly wrong.)
facts.txt	The file facts.txt contains additional facts about animals in plain English. There is one "simple" fact per line.

The Requirements for the Implementation

As mentioned, Zootson ought to be implemented with a language and environment compatible with some of those found in korppi cluster of Birdland. The recommended language is Python 2.6.6 (or whichever version you can find in korppi). Even if the version is not the latest one, the features needed for this assignment are all there. If you have a good reason to use some language or version not usable in Birdland, please contact teemu.heinimaki@tut.fi – special arrangements are possible, given they do not complicate fair evaluation of the work too much.

It should be easy to run the program. If you wish, you may implement support for command line parameters (and different modes and functionalities), but the program should work without any parameters as described in this document.

When Zootson is run, it reads the files zoo.data, continents.txt, and facts.txt. Based on these inputs, it constructs its knowledge in a suitable way. Finally, it reads a question file, 'questions.txt' and outputs a file containing corresponding answers, 'answers.txt'. All these input and output files should be located in the same directory with the Zootson binary/script itself.

An example question file can be found here – there is one simple question per line. The generated answer file should also have one answer per line, corresponding to the questions. The answers should be of the following format:

"yes"|"no"	for the questions expecting an answer of Boolean nature,
"none"|animal_name[", "animal_name]* (e.g., "cheetah" or "pike, vampire, lion"	for the questions asking for animals,
digit[digit]* (e.g., 708)	for the questions asking for numbers,
"nowhere"|continent[", "continent]* (e.g., "Africa, Asia")	for the questions asking for locations,
description[", "description]* (e.g., "big, hairy, fast, beautiful" – typical descriptions used with Zootson are adjectives)	for the questions asking for descriptions, and
"no idea"	for the questions the system cannot answer.

Zootson should be able to produce its answers for the example questions – and any question set with similar number of any questions of corresponding structure – without any significant delay; after starting the execution, the answers should be ready rather "immediately" in the human terms.

Vocabulary

The goal is not to make Zootson a dictionary application; the vocabulary needed for understanding facts and questions is meant to be deducible based on the given examples and the datasets. As only very limited vocabulary is necessary, there is no need (and it is not allowed) to use any existing, external word lists or such. (Basically recognizing few adjectives (with their opposites), few

essential verbs, numbers, given animal classes, animal parts, and few other keywords should be quite enough. And the animal names and continents, of course.)

The Grading

You can gain from 0 to 6 points to the overall course score from this assignment. Getting 0 points means that the work is not accepted and thus you cannot pass the course. Getting 1–6 points means that the work has been accepted, and these points are added up with the exam score, essay score and possible presentation score for the final course grading.

The minimum requirement for passing the assignment is to submit a Zootson implementation capable to answer correctly for at least 75% of the questions in a test question set corresponding structurally to the given example question set without any wrong answers. (There can be, however, some "no idea" answers, which are not considered to be wrong in this test, even if the system "should know".) If the submission has been made as instructed, the documentation is OK, and this test is passed, you pass the assignment and are given at least one point.

For distributing additional points, the passed implementations are tested with larger sets of facts and questions. There can be slightly different types of questions, but they are not intended to be much more complex. All the facts and questions are presented in present tense. Based on this more comprehensive test, the implementations are given "tournament points" (TPs): each correct answer yields one TP, each "no idea" answer given in the situation, in which the program should have been able to know the answer, yields –0.5 TPs, and each other kind of wrong answer yields –2 TPs. The assignment points for the implementations are given based on their total TP scores.

The total score of a program can also be affected by the quality of the documentation. All the matters instructed should be present, the length should be correct, the language should be understandable, and so on. An OK documentation does not affect the number of points awarded, but if there is something wrong with the documentation, points may be taken away. Also the quality of the code may affect the grading, so you should comment your code and follow good conventions, but the emphasis on this course is on the functionality of your system and the ideas behind it, not the code as such. Moreover, it is easy to get penalized by not following the submission orders strictly. Your agent should be of high quality and it should not crash in any situation. If it does, however, it may be penalized harshly. The basic idea of the grading process is illustrated in Figure 1.

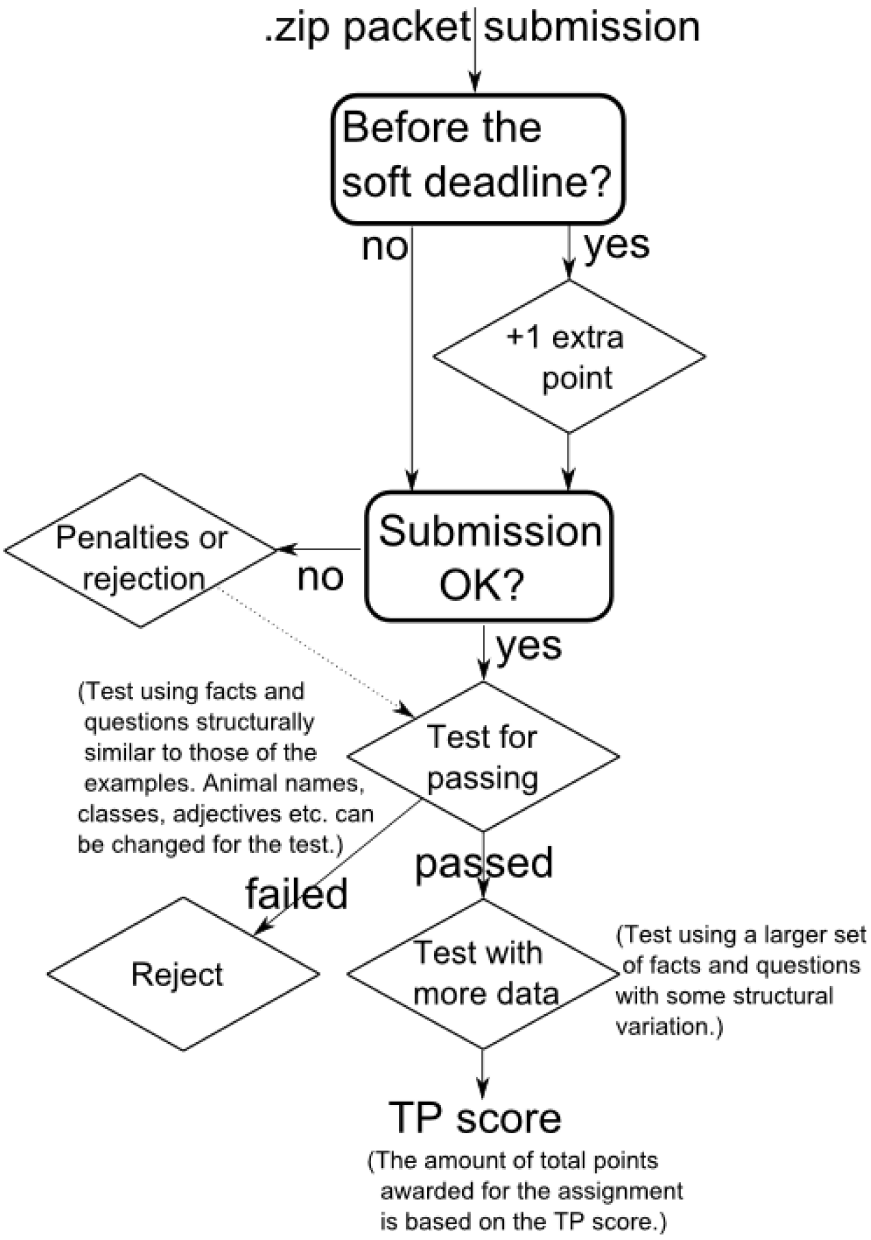

Figure 1: The approximated flow of the grading process.

Submitting the work

The hard deadline for submitting your work is March 31, 2014 – your work must be **successfully received** during that day or before it in order to be able to pass the exercise. There is also a soft deadline, March 10, 2014; you can obtain one extra point ("early bird bonus") in grading, if your submission is received at latest during that day and accepted. (You can still resubmit before the hard deadline for getting another version graded normally. The better score will prevail.)

The work is submitted via e-mail, as a single .zip file attachment. The .zip packet should contain a single directory, and the name of the directory should be your student number. It should contain

- the needed self-written code file(s),
- a documentation file 'documentation.pdf',
- a text file 'own_facts.txt' containing three facts (somewhat similar to those of facts.txt),
- a text file 'own_questions.txt' containing three questions (somewhat similar to those of questions.txt), and
- a text file 'own_answers.txt' containing the correct answers for the questions of own_questions.txt.

The documentation shoud be in PDF format. A suitable length for it is 2–4 pages. The questions of docu.rtf should be answered in your documentation – you can use the file as a template, add your answers, and finally convert to PDF.

The files own_facts.txt and own_questions.txt may be used as additional material for evaluating the implementations. A fixed percentage of all the submitted implementations should be able to answer a question (that can be based on the knowledge offered by the additional facts, or not) correctly for it to be accepted as possible testing material, so do not tailor the questions too much only for your own implementation. Basically the vocabulary needed for understanding the given examples should also suffice for understanding these additional facts and questions. The sentence structures used in own_facts.txt and own_questions.txt may differ from the structures used in the examples, but they should still be "somewhat easy", and the answers should be unambiguous.

The submission e-mail is to be sent to the address teemu.heinimaki@tut.fi. The name (before the filename extension) of the .zip package is to be your student number, so student 123456 ought to name his/her packet, containing the directory 123456, as 123456.zip. The size of the packet should be less than 500 KB. The subject of the submission e-mail message is to be “AI, Zootson submission, *X*” (or in the case of resubmitting your work “AI, Zootson resubmission, *X*”), in which *X* is to be replaced by your student number (like “AI, Zootson submission, 123456”). You do not need to write anything to the actual message part of the e-mail.

There will not be any automated reply, but an e-mail confirming the reception of your submission will be sent within a reasonable time. Please contact teemu.heinimaki@tut.fi, if the reception is not confirmed within the time of two days.

References

[1] Bache, K. & Lichman, M. (2013). UCI Machine Learning Repository [http://archive.ics.uci.edu/ml]. Irvine, CA: University of California, School of Information and Computer Science.