Dept. of Informatics and Media 2018-08-29
Uppsala University

Course information
Declarative Problem Solving Methods
Fall 2018

Course Plan

Goals
Regarding knowledge and understanding the student is, after the course, expected to be able to:
- Describe how declarative programming languages can be built-up and relate these to logic
- Explain and utilise the concepts in programming languages based on logic
- Describe how an interpreter for a logic-based language functions
- Describe different declarative problem solving methods

Regarding skills and abilities the student is, after the course, expected to be able to:
- Solve provided tasks in a logic programming language
- Apply different declarative problem solving methods to solve provided tasks

Regarding judgement and approach the student is, after the course, expected to be able to:
- Discuss and evaluate different declarative problem solving methods in relation to different types of assignments
- Relate declarative programming languages to other types of programming languages

Contents
Introduction to declarative programming languages

Logic programming:
Definitions, programs, databases, rules, facts, recursion, recursive data structures, program structures, built-in predicates, negation through failure, set expressions, search spaces, efficient limitation of the search space, tail recursion, accumulator pairs

Problem solving methods:
Divide-and-conquer, generate-and-test, meta-programming

Instruction
Lectures, lessons, seminars and laboratory work.

Examination
Examination, assignments and attendance to lectures. As grades on the course, one of the expressions passed with distinction, passed and failed is used.

Course literature
Bratko, Ivan : *Prolog programming for artificial intelligence*

Reference literature
Blackburn, P., Bos J., Striegnitz, K: *Learn Prolog Now*,
See free online version at http://www.learnprolognow.org/

Teachers
Anneli Edman room A320, tel 018-4711025, anneli.edman@im.uu.se
Mudassir Imran Mustafa, mudassir.mustafa@im.uu.se
Juan Torres, juan.torres@im.uu.se
Ruth Lochan Winton, ruth.lochan@im.uu.se

Guest teacher
Lars-Henrik Eriksson, lars-henrik.eriksson@it.uu.se

Guidance to the compulsory assignments
Mudassir is responsible for the supervising.

Prolog Interpreters
The recommended Prolog interpreter for the course is *SICStus Prolog*. It is available on the Windows workstations at Ekonomikum in the lab rooms in corridor F, level 3. There is a 30-day trial version available for download (https://sicstus.sics.se/eval.html) if you would like to work elsewhere. Practicing at home is highly recommended!

It is also possible to use SictusProlog on distance through the department's servers. Go to
http://www.ekonomikum.uu.se/service/itsupport/remotedesktop.htm?l=en
for information and then choose ts.im.uu.se

There are other Prolog interpreters available, some of which are free. If you are interested, the Wikipedia page for Prolog should point you in the right direction. *SWI-Prolog* (http://www.swi-prolog.org/) is an example of a free and stable interpreter. Keep in mind that we will probably not be able to help you with code written for interpreters other than *SICStus Prolog*.

Note: We require all submissions to be fully working with the version of *SICStus Prolog* at Ekonomikum.

Examination
The examination comprises

- Five assignments.
- A project.
- A project meeting and a seminar related to the project. Each group presents another group's task and act as opponent to this task at the seminar. The opposition should also be written down and handed in.
- Attendance to lectures and labs (denoted compulsory in the schedule).
- A written exam.

The written exam is worth 4 points. Assignments, project and compulsory lectures/labs are worth 3.5 points together.

Student portal
Material from lectures and some code related to lessons and laboratory work will be available in the Student portal. You will also hand in your assignments, the project and the opposition in the portal. Your ordinary password is needed to log in. Important messages, e.g., changes in the schedule, will also be given in the Student portal or through email sent from the student portal. The address is https://studentportalen.uu.se. You can follow the examination process in the folder Progress in the Student portal.

Don't forget
You need to register in time for written exams, which means that you register through the Student portal *12 days before the exam*. See the regulation below, which is much stricter than earlier.

> **The regulation from fall 2015:**
> "Only candidates who have registered for the examination at the time and pursuant to the arrangement determined by the department may take the examination. It is not possible for candidates to register for the examination in the examination hall. "

Compulsory assignment 1

Task 5 a)-b) from lab 1
Describe your family using facts and then define rules according to the description in the lab.

The tasks should be solved individually.
Submit your program in the Student portal not later than *7th of September.*

Compulsory assignment 2

Task 3 from lab 2
Write a program that calculates whether a given year is a leap-year. See the description in the lab!

Task 8 from lab 2
Write a Hilo-program that gives the player a possibility to guess numbers without posing a question to Prolog for every guess. The program should be solved according to the description in the lab.

The tasks should be solved individually.
Submit your programs in the Student portal not later than *12th of September.*

Compulsory assignment 3

Task 3 from lab 3
Write a program that search for journals based in a set of subjects and a small database with information about the journals. See the lab for more information.

Task 4 from lab 3
Write a program that calculates the number of positive integers, negative integers, and zeros in a list of integers. See example in the lab.

The tasks should be solved individually.
Submit your programs in the Student portal not later than *17th of September.*

Compulsory assignment 4

Task 3 from lab 4
Define a relation between two lists where one list should contain all the elements in the first list that starts with a given combination of letters. See an example in the lab.

Task 4 from lab 4
Write a program that deletes elements in a list using no cut, green cut and red cut. See the lab for a description of the task.

The tasks should be solved individually.
Submit your programs in the Student portal not later than *20th of September.*

Compulsory assignment 5

See lab 5 and 6 for a description of the tasks.

The tasks should be solved individually.
Submit your programs in the Student portal not later than *24th of September.*

The project

Wanted – A Game of Deduction is the assignment. The description is given below.

Getting started

This assignment is solved in groups of three people. We will together discuss how these groups should be formed.

Submission

- Attend the *compulsory* project meeting on *Friday, September 21* to show what you have done so far with the code. The following parts of the documentation should be also presented: draft project plan, the program purpose, and the program structure.
- Publish the complete solution in the Student portal no later than *Thursday, September 27, at 10.00***.** Make sure that the names of all group members can be found in source files and documentation. All uploaded projects can be found in the Student portal 10.15 the same day.
- The assignment is presented at the seminar on *Friday, September 28*, where attendance is *compulsory*. At the seminar, every group presents the assignment of their opponents and perform an opposition. The opposition should also be uploaded in the Student portal.

The form of the program

The program should be well structured, well documented, easy to use and it should handle incorrect input sensibly. Principles of good programming should be applied (correctness, user-friendliness, efficiency, transparency/readability, modifiability, robustness, and documentation).

Documentation

In the documentation, a project plan, a programming manual, and a user manual should be included in one pdf file. The file should contain the group name, the names of all group members, and the date. A table of contents, header with the group name, and footer with page numbers should also be added. All figures and tables (if exists) should be labelled and it is preferred to have a list of figures and a list of tables after the table of contents.

1. Project Plan

A plan describing how the work has been divided between the group members where each task has a date, estimated working hours, and name of group member(s).

2. Programming Manual

2.1. Program Purpose: An introduction and simple description of the game and the purpose of the program

2.2. Program Structure: A diagram/flowchart of the program structure with the subparts.

2.3. User Interface: A simple description of the user interface, main options, inputs and outputs. More details with screenshots of the implementation can be included in the User Manual.

2.4. Program Code: A print-out of a well-documented and commented program.
The code can be included in this section of the programming manual or can be uploaded as a separated file. A description of the related file (if exists) should be included in this section of the document.

In the code file, group information should be included in the beginning, and comments that contain a description of the task and its input and output should be placed above each predicate. The predicate comment can be in the following format:

```
% a simple description of the task
% predicate_name(+InputArg, -OutputArg,…)
% description of input argument(s) +InputArg
% description of output argument(s) –OutputArg
% …
```

Example:
```
% Calculate the length of a list
% count_list(+List, -Count).
% +List     is the given list
% -Count  is the length of the list

count_list([], 0).              % base case
count_list([_|Rest], N) :-      % recursion
     count_list(Rest, N1),
     N is N1 + 1.
```

2.5. Testing: A report regarding the tests that are performed and their results. Tests regarding whether or not the program performs the main requirements and meets the objective of the project should be performed. In addition, tests should ensure the robustness of the program and that it is able to provide an appropriate output for any errors that might occur in the system or any user-related errors such as incorrect inputs.

Each test case should have the following info:
Id, date, tester name, test description, user input, expected result, actual result, whether it passed or failed, severity of the defect, summary of the defect and how it was resolved.
Screenshots can also be added to each test case if necessary.

3. User Manual
A user manual with instructions on how to run the game and examples of program executions where it is clear how the user could communicate with the program. Screenshots of the implementation should also be included.

Opposition
In the written opposition the following should be included:
1. Name of the opponents and the name of the group members that have done the assignment.
2. A short description of the assignment.
3. A description of the user interface (the chosen solutions, advantages, disadvantages, possible alternatives) and how easy it is to use.

4. A report regarding the program code (the solutions chosen, advantages, disadvantages, possible alternatives) and an evaluation of how easy it is to follow the code.
5. An evaluation of the program's function (do all the subparts work well, does the program cope with incorrect input).
6. An evaluation of how well the program has been tested.
7. An evaluation of the whole program documentation.

In the oral opposition the group should choose to present some of the given points given above, points that are important in relation to the current program.

Wanted - Game Description

A murder has been committed! The player's job is to find out who the murderer is.

At the start of the game, the program randomly selects a person to be the murderer, and the murderer is kept secret until the end of the game. The game begins. A new person is randomly selected and presented to the player in one of two ways: either as a "suspect" or as an "innocent." This process is repeated until the player is ready to guess who the murderer is.

Each person has a unique combination of attributes, which the program bases its decision on:

- A person is considered a "suspect" if *at least one attribute* matches the attributes of the murderer.
- A person is considered "innocent" if *no attributes* match the murderer's.

As soon as the player understands which combination of attributes the murderer must have, the player should input a guess indicating the appropriate attributes. The game ends, and the player is presented with a message saying that the guess was either correct or incorrect.

Attributes

Each person consists of a unique combination of three attributes:
- age_gender (oldman, oldwoman, youngman, youngwoman)
- colour (blue, brown, green, yellow)
- weapon (gun, knife, poison)

Defining the values of attributes should be dynamic, which means that it should be easy to add/remove values without affecting the program.

Examples of combinations:

- youngwoman dressed in blue with a gun
- oldman dressed in green with poison

Once a particular person (i.e. a unique combination of attributes) has been randomly selected and presented it should not be selected again in the future! This means that once a particular combination has been selected and assigned either "murderer", "suspect" or "innocent", this combination should never again be generated or used elsewhere. For instance, the combination of the murderer must never later in the game show up among the innocents. A person who has been assigned to the suspects cannot later turn out to be the murderer, and so forth. Every person is unique in combination (age_gender, colour, weapon) and can be assigned only one of the following: murderer, innocent, suspect.

Requirements

The player should be always able to:

- Query the program for a new person,
- Guess the combination,
- Check the latest randomly selected person and to which category it was assigned,
- Check the innocents so far,
- Check the suspects so far.

After the game finishes, the number of randomly selected persons (the murderer does not count) should be displayed. This number is interesting because it represents the number of "tries" it took the player to deduce the murderer's combination. The aim of the game is of course to successfully deduce who the murderer is in as few randomizations as possible!

The persons (the combinations of attributes) must not be hard-coded into the program, but instead automatically generated at runtime from the possible attributes listed above. This way, adding new attributes should be as simple as adding new elements to a list and restarting the game.

Example Run

Note! The following should not be interpreted as a screen shot from a real program execution, but rather as an example to further illustrate the logic and flow of the program.

The game starts and the murderer is selected: [oldwoman, yellow, knife]
(This is only shown here for reference -- the murderer is otherwise always kept secret from the player!)

Person 1: [youngman, brown, gun] - innocent!
Person 2: [youngman, yellow, poison] - suspect!

Now we know that the murderer must have at least one of yellow or poison, simply because youngmen are innocent according to Person 1.

Person 3: [oldwoman, brown, gun] - suspect!

Now we know that the murderer is an oldwoman because both brown and gun are among the innocent (Person 1).

Person 4: [youngwoman, blue, gun] - innocent!

Now we can conclude that the murderer cannot be dressed in blue.

Person 5: [oldman, blue, poison] - innocent!

Now we know that the murder's weapon is a knife because both poison and gun have been declared innocent. Since we know that we are after an oldwoman with a knife, looking back at Person 2 leads us to the conclusion that the murderer must be dressed in yellow.

"The murderer is an oldwoman dressed in yellow with a knife!"

Congratulations! Your deduction required 5 persons!

Tips and Tricks

If a random number generator is needed, SICStus Prolog can provide you with a solution by adding this line to the top of your program:

```
% Include the necessary predicates.
:- use_module(library(random)).
```

You use this through the predicate random/3:

```
random(Min, Max, RandomNumberBetweenMinAndMaxMinusOne)
```

Example:

```
random(0, 10, R).   % R becomes 0-9.
R = 6
```

Good luck!