Computer Science

CompSci 367 S2 C
- ASSIGNMENT Three -

The work done on this assignment must be your own work. Think carefully about any problems you come across, and try to solve them yourself before you ask anyone else for help. Under no circumstances should you work together with another student on any code used in this assignment. Any code you reuse from any source MUST be referenced. You should use SWI Prolog, it is freely available to be downloaded for all platforms. Your system will be marked using this version of Prolog.

Assessment
Due: Monday 21st October 2013 11.59 pm
Worth: 12% of total 367 marks

Aim of the assignment
The main aim of this assignment is to give you some experience modeling a planning domain. The particular planning domain is that of RushHour®. RushHour® is a commercial game and the url for its instructions: http://www.puzzles.com/products/rushhour/rhfrommarkriedel/jam.html#instr). There is an online version you can play to get a feeling for the rules at: http://home.kpn.nl/fredvonk/rushhour.htm The specific problem will be determined by the specific initial state. The goal is the same for all RushHour® problems, namely, to enable the red car to exit the grid.

Modeling "RushHour®"
Modeling a domain involves deciding how to model states of the world (in this case, the RushHour® board) and how to model actions that alter these worlds. The class lectures on "Domain Modeling" should give you the required background for modeling a domain. There is an additional document associated with this assignment entitled "Design Notes", you should read this and go back over the lecture notes before starting this assignment.

What is Required?
You will need to do the following things for this assignment:

- Pick the domain predicates that you will use to represent the world states and the domain actions. You will need to decide the type of each predicate. A domain predicate can only be of the following four types: metaLevel, derived, static, or fluent. These predicates should be "typed" and documented in the **ontology.pl** file in the **rushHour** directory. Fluent predicates do not need to be explicitly "typed" (predicates are fluent by default) but should be documented.

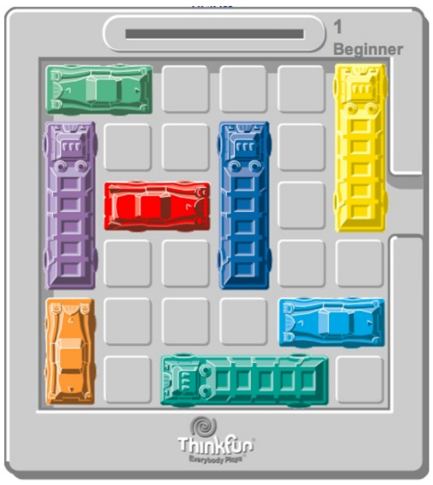
- You will need to have derived predicates and will have to define their meaning.
- You will need to have static predicates.
- You will need to have metaLevel predicates and will need to define their meaning in Prolog. One metaLevel predicate you should think of creating is an "is" metaLevel predicate for doing arithmetic. It would look like the following: *is(Variable, ArithmeticExpression)*, for example "*is(X, 4 + 2 * 7)*", which would evaluate the arithmetic expression and bind its value to the variable X. The existence of an "*is*" predicate may ease the effort of modelling the RushHour® domain.
- Design all the domain actions that alter the states. Note that all vehicles can ONLY move one position per action! These actions are stored and described in the **ops.pl** file in the **rushHour** directory.
- Since all RushHour® puzzles have the same goal, you need to come up with the goal expression for that goal. Assume the exit square is always the same as shown in the RushHour image shown below (i.e., the 3rd row from the top and the 7th column from the left-hand side) and that it is sufficient to get the nose of the car to the 3rd row, 6th column to consider the problem solved. Note: this requires the red car to always be on the 3rd row for the problem to be solvable.
- Write a Prolog predicate that translates "iconic" representations of RushHour® states into problem descriptions using your domain representation. This predicate will have the following signature: *icon2symbolic(IconicRep, problem(SymbolicInitialState, SymbolicGoal)),* this predicate will be stored and documented in the **icon2symbolic.pl** file in the **rushHour** directory.
- The planner expects all state descriptions to be ordered sets (*ordset)*. This means the initial state description produced by your translator needs to be made into an ordset list using the predicate *list_to_ord_set/2* from the SWI-Prolog ordset library.

Given a valid iconic description of a RushHour® initial state, e.g., **prob01.pl**, your code translates it into the corresponding problem represented in your ontology (predicates). The problem can be stored in a file in the rushHour **problems.d** directory and it can be solved using the solve/4 predicate in the solve.pl file in the top directory. Assuming you save your problem description in the file **probMine01.pl**, then you would enter the following to ask Prolog to solve the problem: *solve(rushHour, zero, probMine01, Solution)*, where **zero.pl** would be the name of the file containing the *zero* heuristic.

Iconic RushHour State Representations

Conceptually, a RushHour state is a 6 x 6 matrix, where the value of each element represents the contents (i.e., the car) of its corresponding square. However, RushHour states are actually represented "iconically" as 36 item lists. The first 6 items correspond to the top row of the RushHour conceptual matrix, the 2nd 6 items correspond the next row down, etc. Each item in the list is a number. If there is no vehicle in the corresponding square then the corresponding number is zero. If the vehicle in the corresponding square is the red car to be driven out of the grid then the corresponding number is one. All other vehicles can be numbered arbitrarily (as long as no two distinct vehicles have the same number and are integers greater than one).

So, for the RushHour state shown below, it could be represented as the following list: [2,2,0,0,0,3,4,0,0,5,0,3,4,1,1,5,0,3, 4,0,0,5,0,0,6,0,0,0,7,7,6,0,8,8,8,0].

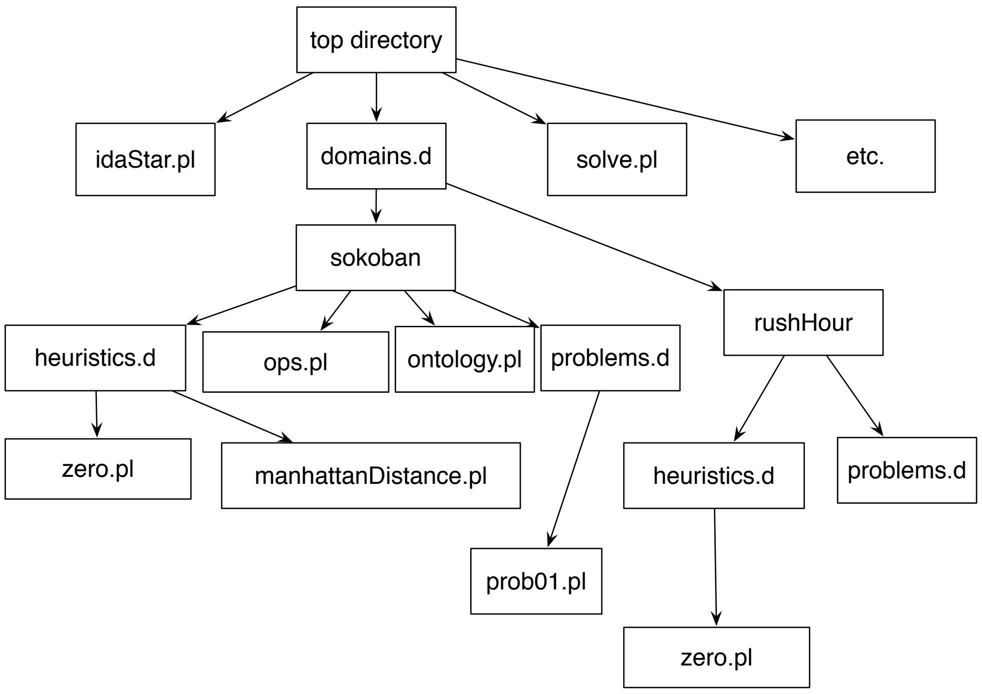


Your translator will need to take the list shown above and translate it into the appropriate problem description using your ontology. It should be a problem description that this assignment's planner can take and solve.

Iconic representations will be encoded as: *iconic(<LIST>)*, where *<LIST>* is a list as described above. The above iconic representation of this problem can be found at **domains.d/rushHour/problems.d/prob001.pl** in the assignment archive.

Directory Structure

The diagram below shows the existing directory structure. You will be adding the appropriate files below the **rushHour** domain directory.



Resources

For everything you are being asked to write for the RushHour domain, there are corresponding examples in the sokoban domain in the assignment archive. I would recommend that you look at them.

Marking Guide – 12 Marks in Total

Things you will be marked on:

- Defining and using a reasonable derived domain predicate [1 mark]
- Using a reasonable static domain predicate [1 mark]
- Implementing and using a reasonable metaLevel domain predicate [1 mark]
- Define reasonable domain actions [2 marks]
- Create an appropriate goal expression for RushHour® [1 mark]
- Write prolog code to translate my iconic representation of a state into a problem description using your domain ontology [2 marks]
- A translation of **prob01.pl** (the iconic form of the RushHour problem shown above) into the appropriate problem representation (stored in the file **probMine01.pl**) using your ontology [1 mark]
- Whether you have created a working representation for RushHour®. In other words, the assignment's planner solves the problem stored in **probMine01.pl** [2 marks]
- Clear in-line documentation of all the above [1 mark]

Submission of Assignment

Archive your rushHour directory as rushHourArchive.zip and submit via the Computer Science Assignment Dropbox (https://adb.auckland.ac.nz/Home/) before the deadline.