

CMPT310 - Project

Overview

The goal of this project is to create a simplified Prolog implementation of the SHDRLU system (sans graphics).

There will be three sub-systems (or parts) to this diagnostic system:

- The blocks world domain model and STRIPS representation.
- A general purpose planner.
- A natural language interface.

You will create a team of three people to work together on this project. Each person will be responsible for a different sub-system, and for parts of the other deliverables (see below). As a last resort, if you cannot form a team of three, please contact the instructor to arrange for other options.

You must create your assignment in electronic form using the accepted file formats, and including all assignment parts specified in the Deliverables sections below.

Note: Be sure to read the Deliverables section before you begin working on the assignment.

Important Note: Students must work within their groups on this project. You may **not** discuss the specific problems in this project, nor their solutions with students who are not in your registered group. You may **not** provide or use any solution, in whole or in part, to or by another student who is not in your group.

You are encouraged to discuss the general concepts involved in the project in the context of completely different problems.

Description

The purpose of this project is to develop a Prolog version of the SHDRLU system <<http://hci.stanford.edu/~winograd/shrdlu/>>. The core features of this system were a blocks world domain, a problem solver, and a natural language interface.

Each sub-system interacts with the other to create the full system. They are detailed below.

Representation – Blocks World Domain

The blocks world domain is composed from a collection of mobile blocks on a fixed table. The table has finite size and blocks must be placed so that they are entirely on

the table, so blocks cannot fall off the table. There are several types of blocks: cubes, rectangles, pyramids, triangles, and boxes. Blocks have attributes: colour, shape, and position. The colour may be red, green, or blue. The shape of a block is determined by a base rectangle and a top rectangle. The position is a point in 3D space $\langle X, Y, Z \rangle$ (where X, Y, Z are integer values). The physics model of the blocks world is very simple; blocks cannot fall off other blocks. Blocks must rest on another block (or blocks) or on the table. A block is resting on another block when the base rectangle of the top block is at the same height as either the top or base rectangle of the lower block, and the two rectangles intersect.

Different blocks have different attributes:

- Pyramid: The top rectangle is a point (e.g., has zero width and length). Do not place blocks on top of a single pyramid (even though the physics model allows this). Pyramids with their tops at the same height can form a support for another block, provided that the base rectangle of the top block intersects with three or more of the pyramids top rectangles.
- Triangle: The top rectangle is a line (e.g., has zero length, but the same width as the bottom rectangle). Similar to the pyramid, do not place blocks on top of a single triangle (even though the physics model allows this). Two or more triangles with tops at the same height can support another block.
- Box: It is possible to place items inside a box, provided that the inner block is smaller than the box, and is allowed to sit atop any block already inside the box.
- Rectangle: A rectangle can be rotated so that that it is placed on its side, or on its base.

It is possible to create an arch from three blocks, such that a block rests on two stacks of blocks which do not touch each other. A block can only rest on multiple blocks if the supporting surface has uniform height. Multiple blocks can sit on a supporting block, provided the physics rules are satisfied.

Blocks are moved by a manipulator arm which can only move one block at a time, and can only move the top-most block in a stack.

The KB for the domain model must contain rules and facts which:

- Specify the attributes of each block as specified above.
- Determine the user visible size attributes of each block (e.g., big, small, medium).
- Specify the relationships between blocks (e.g., supports, above, below, on, under).
- Contain predicates which determine which blocks form an arch.
- Specify the currently selected block (or blocks).
- Specify the block movement rules in STRIPS format.

The rules and STRIPS representation should be sufficiently general so that additional blocks can be added to the domain (or removed). The initial domain should be populated with a sufficient variety of blocks so that the domain includes all types of

blocks of a variety of sizes (> 10 blocks). The table should be small enough to necessitate some block stacking, on occasion, when moving blocks, but large enough to construct interesting block configurations (e.g., arches).

The user of the system will not have access to the <X,Y,Z> position, or size rectangle attributes of the blocks, nor the constant which identifies a block. It should be possible to query for a block based on its other attributes and relations (e.g., colour, supports, above, below, on, under, big, small, medium). The user identifies blocks via these other attributes and relations.

One team member will be responsible for the domain model representation. They should work with the reasoning team-mate to ensure the KB is written in the form required by the planner. They should also work with the natural language team-mate to determine which nouns, verbs, adverbs, and adjectives are needed to describe the domain.

Reasoning – General Purpose Planner

The general purpose planner takes a description of the goal state and determines the sequence of moves that will achieve the goal state, if possible, starting with the current state.

The planner is general purpose in that it is not dependant upon the given domain, but could work with any domain specified in the STRIPS representation.

The planner should create its plans without altering the KB (i.e., it should represent the domain state in a list, and work with the list while planning). After a plan is constructed and accepted, the planner should be able to execute the completed plan so that the KB is updated.

Hint: To update the KB, use the predicates: `assert`, `asserta`, `assertz`, `retract`, and/or `retractall`.

The planner will also need to determine the minimal set of descriptions needed to specify a block. For example, it may be possible to uniquely identify a block in several ways:

- by all its colour.
- by its shape.
- by its colour and shape.
- by its colour, shape, and size.
- by the blocks it is on.
- by all its attributes and the stack of blocks it is above.
- etc.

The planner should support a predicate that returns a minimal description (i.e., list of

attributes and relations) for a given block. The natural language interface can use this to represent a block instance to the user (after translating the attributes into a natural language sentence).

Hint: The `findall/3` predicate can be used to query for all blocks that match a given query.

One team member will be responsible for the planner. They will need to work with the domain model team-mate to work out the predicate and STRIPS representation of the domain. They will also need to work with the natural language team-mate to provide the predicates which perform plan construction and execution.

Recognition – Natural Language Interface

The domain model and planner are both hidden from the user of the system. Instead, all queries, and instructions are entered using a restricted domain of English natural language.

All output, such as query results, questions, or answers, are displayed in natural language (instead of predicates and variable assignments). The natural language system must perform this conversion.

All input from the user is entered in natural language, and is then converted into the form that the underlying planner or domain model requires. In essence, the natural language interface sub-system constructs the Prolog queries from natural language user input. These queries will either be for the domain model sub-system, or for the planner sub-system. The output from these queries is displayed to the user in a natural language form.

The goal state is specified in terms of user attributes. This means that the goal state may be ambiguous. When there is ambiguity over which blocks the user is referring to, the natural language interface should ask the user to clarify. Ambiguity over the exact position of the block in $\langle X, Y, Z \rangle$ space is ignored. The planner determines the actual position based upon the goal relationship (e.g., red pyramid on top of green cube).

The lexicon should be sufficiently large to describe the user level attributes of blocks and their relationships between each other. It should contain all determiners, verbs, adverbs, nouns, pronouns, and adjectives needed to form the types of sentences the system will support as both input and output.

Some example sentences and system interactions are shown below (domains are not consistent between examples):

```
user> where is the red pyramid?  
computer> it is on a small green rectangle, and a blue cube.
```

user> where is the blue block?
computer> which blue block? there is a small blue cube in the red box, and a big blue triangle on a green triangle.
user> where is the small blue cube?
computer> it is in the red box.
user> move it to the table.
computer> done.

user> where is a box?
computer> there is a big red box on the table.
user> where is the box?
computer which box? there is a big red box on the table, a small blue box inside the big red box on the table, and a small red box on a medium blue cube.

user> which blocks are not above the table.
computer> no blocks are not above the table.

user> which blocks form an arch?
computer> an arch is formed by a triangle on top of a red rectangle on the table, and a blue cube on a green cube on the table.
user> move the green cube on the table to the red box on the table.
computer> done.
user> how did you do that?
computer> i moved the triangle to the table, moved the blue cube on the blue rectangle, and then moved the green cube into the red box on the table.

user> which block is bigger than the red block on the blue block.
computer> the green triangle is bigger than the red block on the blue block.
user> move the green cube on top of the big green triangle.
computer> cannot move the green cube on top of the big green triangle.

Note: These examples are just to give a flavour of the type of possible interaction. The natural language interface must allow the user to discover the blocks that make up the domain, discover their relative positions, and specify where to move the blocks.

Create a list of exemplar sentences for your system before you begin. From these examples, create natural language sentence templates which can handle a variable number of objects and attributes. Start with simple sentences and constructs, then add more the complex capabilities (i.e., pronouns, ensuring blocks are uniquely identified, etc.) into the natural language sub-system.

One team member will be responsible for the natural language interface. They will need to work with their other team-mates on such issues as the lexicon requirements, and the predicate interface for the domain model and the planner sub-systems.

Other System Requirements

You should strive to create an extensible system that can work for other domains.

It should be possible to swap out one domain system for another (e.g., swap the blocks world domain for a diagnostic domain), while keeping the domain independent planner, and still have a functional domain model.

At very least the natural language interface should have an extensible lexicon. Ideally, it could also build the lexicon automatically from the domain model.

Documentation

Together you must create documentation for this project. For each sub-system there will be:

- A short summary (or overview) of its capabilities, and design. (a paragraph or two).
- A detailed write-up on the capabilities, design, and implementation details. (a page or two).

Each team member must write one short summary and one detailed write-up. You must not write about the sub-system you implemented. Instead you must choose one of the other sub-systems for the summary, and the other sub-system for the detailed write-up. Furthermore, you must do so in such a way that each sub-system has one summary, and one detailed write-up.

Note: The detailed write-up should provide sufficient information to allow any other Prolog programmer to extend or maintain your code.

Bonus

The JLog Prolog in Java applet <<http://jlogic.sourceforge.net/>> supports graphics and animation. Create a modified version of your project to run on JLog, and add a graphical presentation of the blocks world in the Animation pane. The Animation pane should remain updated with the current blocks world state, by showing the blocks moved, and the manipulator arm that moves them. The Animation pane view should also show the current 'it' blocks (i.e., currently selected blocks) as highlighted.

The JLog version should include all the necessary source code in one file (separate from the sub-system code for geared for SWI-Prolog).

Followup Questions

Each group member must answer the following question independently, based upon their original solution in Assignment #1. Make sure that the author of each answer is clearly labeled.

0) In Assignment #1, Question 0, you were asked to construct a classification scheme for computational intellects to determine which of three rights categories they should belong to.

In two or three concise paragraphs, critique and revise your original solution based upon the concepts and methodologies covered during this course. If your position has changed, explain why. If not, explain how the course topics support your solution, or explain why that are not relevant.

Note: Please answer this question after you have completed the bulk of this project, and the course is nearly completed.

Deliverables

Register Group

When you formed your group, please send an email to the instructor as notification. Include the names and email address of all students belonging to that group.

Final Product

You must create your assignment in electronic form.□The accepted text formats are:

- ASCII Text (Unix linefeeds) for Prolog source files.
- HTML for documentation.
- PDF (Portable Document Format) for documentation.

And the accepted image formats are:

- GIF
- JPEG
- PDF

All source files should work using SWI-Prolog on the department machines. You may work from home, and in your preferred Prolog environment; however, you must ensure that the final product works without error on the department machines.

Deliverables will include:

- Source file for domain model sub-system.
- Source file for planning sub-system.
- Source file for natural language interface sub-system.
- Answer to follow-up question 0 from each student.
- Documentation, including three detailed, and three summary, sub-system write-ups.
- The complete cover_sheet_group.txt file.
- Optional: bonus JLog source file for entire system, with a graphical domain display.

Note: Only *one* person in the group should perform the hand in. They should however, have all the files created by the group.

Every member of the group should understand the work performed by other group members.

You **must** complete the `cover_sheet_group.txt` file and include it with your other files. This file is available from the course main page:

<http://www.cs.sfu.ca/CC/310/gholst/>

All files should all be labeled appropriately so that the markers can easily follow and find your work.

Create an archive (either `*.zip` or `*.tar.gz`) containing all relevant assignment files, plus the cover sheet.

When you are done, go to the Assignment Submission Page:

<https://onara.cs.sfu.ca/>

And follow the instructions there.