Overview

- Golog and Prolog
- The Taxi Problem
Algorithms in Logics
Language based on Situation Calculus
  - Situations
  - Actions
  - Predicates (atemporal, fluent)
  - Axioms (descriptive, precondition, successor state)

How to automatize inference in this language?
  - We have Prolog

Golog interpreter is written in Prolog
  - Both have similar syntax
Prolog

- Declarative language (relations)
  - Knowledge base (facts and rules)
  - Queries (can you prove/satisfy this?)

- User can provide a KB and ask a query.

- Prolog uses resolution (backward chaining) over KB to answer.

Why Prolog?

- First Prolog compiler by David H. D. Warren (Edinburgh)
- The “Edinburgh Prolog” dialect serves as the basis for the syntax of most modern implementations.
- Prolog has been used in Watson (IBM).
The Syntax of Prolog

- Predicates & constants start with lower-case.
- Variables start with upper-case.
- A **Term** is a constant, variable or composite from other terms.
- No quantifiers
  - variables in KB are implicitly universal
  - queries ask for satisfaction (existential-like)

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>,</td>
<td>conjunction</td>
</tr>
<tr>
<td>;</td>
<td>disjunction</td>
</tr>
<tr>
<td>:-</td>
<td>if (rev. implication)</td>
</tr>
<tr>
<td>.</td>
<td>end of sentence</td>
</tr>
<tr>
<td>=</td>
<td>unification (also prefix)</td>
</tr>
<tr>
<td>_</td>
<td>anonymous variable</td>
</tr>
</tbody>
</table>
Example of Prolog syntax

- KB:
  - horse(bluebeard).
  - offspring(charlie, bluebeard).
  - horse(H) :- offspring(H,X), horse(X).

- Queries:
  - horse(charlie).
  - horse(X).
  - lion(X).

":-" is only used in rules. Rules cannot be queries.
Create a KB file, e.g., demo.pl

Load the file in Prolog with command [demo].

Useful predicates:

- listing/0: list all facts in KB
- assert/1: save a fact into KB
- retract/1: remove a fact from KB
- halt/0: exit Prolog
You can run `../plan.sh sample-blocks.pl`
Runs on DICE machines or on Linux machines with installed SWI Prolog ("swipl").
Modify the program in order to find a solution of the Susman anomaly (sorry, no marks for this).
Prolog Demo

KB

edge/2.
edge(a,b).
edge(a,e).
edge(b,d).
edge(b,c).
edge(c,a).
edge(e,b).

Query1

edge(a,b).

a → e

↗
c

↓ ↙

b → d

Query2

edge(a,b), edge(b,c).

Query3

edge(a,b), edge(a,c).

Query4

edge(a,b); edge(a,c).

Query5

edge(a,X).
An atom in prolog is anything that can be represented internally by a single unit (e.g. string, variable, empty set)

Rule:
Atom :- Atom1, Atom2, ..., AtomN.

e.g.
\[
tedge(X,Y) :- \text{edge}(X,Z), \text{edge}(Z,Y).
\]

nodes X and Y have distance 2

\[
\text{path}(X,Y) :- \text{edge}(X,Y).
\]
\[
\text{path}(X,Y) :- \text{edge}(X,Z), \text{path}(Z,Y).
\]

there is a path from X to Y.
Numbers in Prolog

- Numbers are constants.
- Some arithmetic operators:
  
  \[ + \quad - \quad / \quad \ast \quad > \quad < \quad \geq \quad \leq \quad is \quad =:= \]

**KB:**

price(book,10).
price(coffee,3).

canBuy(Money,Item) :-
  price(X,P), Money\(\geq\)P, Item=X.

**Queries:**

canBuy(2,Item).

canBuy(6,Item).

canBuy(12,Item).
Create alternative definition of numbers:
Use a predicate “s” that stands for successor!

Rule:
\[
\text{can}(A, X) :- A = \text{move}, X = s(s( )); \\
A = \text{jump}, X = s(s(s( ))).
\]

Queries:
\[
\text{can}(\text{jump}, s(s(s(s(s(0)))))), \\
\text{can}(\text{move}, s(0)), \\
\text{can}(X, s(s(s(s(s(0)))))).
\]
Taxi Dispatch Agent
You are asked to:

1. Formalise the problem in the Situation Calculus
2. Implement in Golog
Start with: The first two chapters of “Learn Prolog Now!” by Patrick Blackburn, Johan Bos, and Kristina Striegnitz (online version);

Play with the block example;

Start with the simple tasks and build up.
Problem Description

Steps

1. Find and list the facts
2. Represent those facts in a compact representation

Facts

- Map is discrete
- 8 locations with streets connecting them
- Taxi is in a locations room (e.g. A)
- Passenger is static and its location is known
- Dispatcher can be reached at all times for updated information
Street map with pick-up and drop-off locations.
• Create a model using situation calculus
• atemporal predicates, fluents
• as compact as possible
  • remove unused predicates or fluents
Part 1: Formalise

- The environment:
  - which locations are connected?
  - where are passengers and taxis?
  - what is the state of the taxi?

- The actions:
  - what can the taxi do?
  - when can it do it? (preconditions)
  - how do actions change the environment? (effects)
Tips

- If locations $A$ and $B$ are bidirectionally connected the taxi can go from $A$ to $B$ and from $B$ to $A$
- If locations $A$ and $B$ are unidirectionally connected the taxi can go either $A$ to $B$ or from $B$ to $A$
- To pick up a passenger, the taxi and the passenger must be in the same location
Parts 2 & 3: Implement and extend

- **Part 2:**
  - Translate Part 1 into the syntax of Golog
  - Test on some initial states & goals

- **Part 3:**
  - Add more predicates
  - Change your axioms to suit the new predicates
  - Test
Assignment package

- 8 files
- A Golog-based planner - only available on DICE machines
- Domain and problem template files
- All text should go in answer.txt
- Blocks world example.
Tips

- Being concise will help you:
  - Remove unused predicates
  - Remove arguments in predicates that don’t need them.

- Balance abstraction:
  - Don’t add a new predicate for every action.
  - Don’t use same predicates for things that are essentially different.
All original files

- Part 1 answers should be in answer.txt
- Part 2
  - a domain file: domain-task21.pl
  - problem instances:
    - instance-task22.pl, instance-task23.pl, instance-task24.pl
- Part 3
  - domain-task31.pl, instance-task31.pl
  - domain-task32.pl, instance-task32.pl
  - domain-task33.pl, instance-task33a.pl and instance-task33b.pl
To conclude

- Worth 12.5% of the final mark
- Deadline: 4pm on Monday 28st March
- Electronic submission through submit software (available on any DICE machine)