CS761 Artificial Intelligence

Tutorial 1:
Agent Driven by Propositional Logic
&
A Basic Tutorial for Prolog

Agent Driven by Propositional Logic

Recall: Knowledge-based Agent

The wumpus world scenario may be realised by a knowledge-based agent.

Knowledge-based Agent Program Framework

INPUT: *percept*, background knowledge *KB*, clock *t* (initially 0)
OUTPUT: An action

$Tell(KB, MakePerceptSentence(percept, t))$
$action \leftarrow Ask(KB, MakeActionQuery(percept, t))$
return $Tell(KB, MakeActionSentence(action, t)$
$t \leftarrow t + 1$
return *action*

Two important operations:

- *Tell*: Add a sentence to KB.
  - We introduced proposition logic as a knowledge representation language.
  - A clause is a proposition of the form

$$(h_1 \vee h_2 \vee \cdots \vee h_m) \leftarrow (\ell_1 \wedge \ell_2 \wedge \cdots \wedge \ell_k)$$

  - A propositional knowledge base is a set of clauses.
- *Ask*: Reason if a sentence is entailed by the KB
  - A model of a propositional knowledge base KB is an interpretation that satisfies KB.
  - A proposition g is called a logical consequence of a knowledge base KB, written as

$$\mathrm{KB} \models g$$

    if g is true in every model of KB.
  - An inference engine decides for any KB, a set of percept atoms Percept, and proposition g, whether

$$\mathrm{KB} \cup \mathrm{Percept} \models g$$

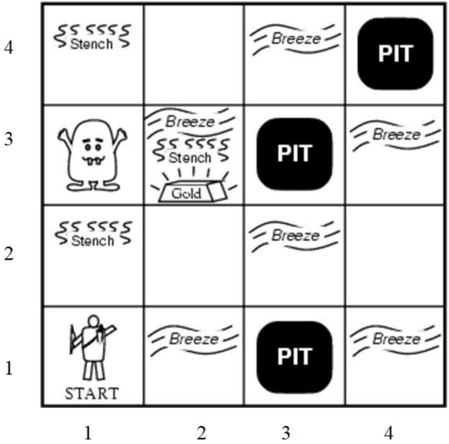
A Propositional Wumpus World Agent

We now bring together all points above in order to construct a wumpus world agent using propositional logic.

We need to handle the following questions:

- World model:
  - How does the world evolves independently of the agent?
  - How do the agent's actions affect the world?
- Internal state of the agent:
  - How does the agent perceive the environment?
  - How does the agent update its knowledge base?
- Inference:
  How does the agent use logical inference to make decisions?

The wumpus world:



States: Static knowledge

Define the atomic propositions: For any $i, j \in \{1, 2, 3, 4\}^2$

- $B_{i,j}$: Breeze can be sensed at location (i, j)
- $P_{i,j}$: Location (i, j) has a pit in it
- $W_{i,j}$: Location (i, j) has a wumpus in it

In KB, add the static knowledge axioms:

- Starting location: $\neg P_{1,1}, \neg W_{1,1}$
- $B_{1,1} \leftrightarrow (P_{1,2} \vee P_{2,1})$
- $S_{1,1} \leftrightarrow (W_{1,2} \vee W_{2,1})$
- There is at least one wumpus:
  - $W_{1,1} \vee W_{1,2} \vee \cdots \vee W_{4,3} \vee W_{4,4}$
- and others ...

States: Dynamic Knowledge

Define the atomic propositions: For $t \in \mathbb{N}$, $i, j \in \{1, 2, 3, 4\}$,

- $WumpusAlive^t$: Wumpus is alive at time t
- $OK^t_{i,j}$: It is OK to move to position (i, j) at time t
- $L^t_{i,j}$: Agent is located at (i, j) at time t
- $FacingEast^t$: The agent is facing east at time t

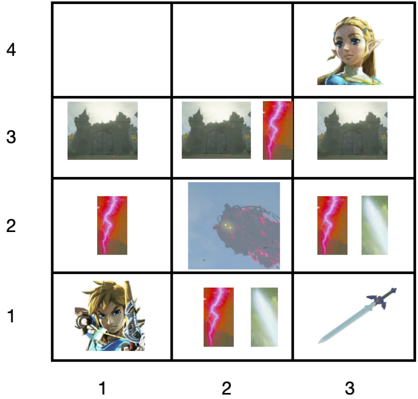
In KB, add the following dynamic knowledge axioms:

- $L^0_{1,1}$
- $FacingEast^0$
- $OK^t_{i,j} \leftrightarrow (\neg P_{i,j} \wedge \neg(W_{i,j} \wedge WumpusAlive^t))$
- and maybe others

Q2 in Assignment 2

For any $i \in \{1, 2, 3\}$ and $j \in \{1, 2, 3, 4\}$

- $C_{i,j}$: Calamity can be sensed at location (i, j)
- $G_{i,j}$: Ganon is at location (i, j)
- $S_{i,j}$: Master sword is at location (i, j)
- *HoldingSword*: Link has picked up the master sword



Questions:

(a) Use atomic propositions given above to represent the following static knowledge axioms:

  (1) Ganon must be at one location. ***(You do NOT need to assume whether Ganon is unique or not.)***
  (2) Ganon and master sword are not at the same location.
  (3) Calamity cannot be sensed at two adjacent locations at the same time.

(b) Use atomic propositions given above to define dynamic knowledge axiom $OK_{i,j}$ that denotes that it is safe to move to location (i, j).

Note: The explanation for how a formulas is derived is NOT required.

A Basic Tutorial for Prolog

Imperative v.s. Declarative Programming

Imperative programs: describe the steps of computation, i.e., how to produce an output.

Example. The procedure to sum a list of integers.

```
int sum(int[] list){
    int result = 0;
    for (int i=0; i<list.length;++i){
        result += list[i];
    }
    return result;
}
```

Declarative programs: describe what output you want, rather than how to produce the output.

Example. The definition of the sum of a list of integers.

```
sum([],0).       % sum of an empty list is 0
sum([FirstItem | Rest], Sum) :-
    sum(Rest,SumRest), Sum is FirstItem + SumRest.
                 % sum of a non-empty list is the first item +
                       sum of the rest of the list.
```

Logic Programming

Logic programming is a declarative programming language paradigm that aims to separate a program into its logic component and its control component:

- the logic component applies logic as a knowledge description language to describe what to compute;
- the control component applies inference to solve the problem.

Prolog is one of the most important logic programming languages:

- Developed in Marseille, France in 1972.
- First-order logic as its knowledge representation language.
- Widely used in theorem proving, expert systems, automated planning, and natural language processing.
- Download and install the client (Windows, Linux, MacOS (not recommended)):

  `https://www.swi-prolog.org/download/stable`

  Online version (recommended):

  `https://swish.swi-prolog.org/`

SWI Prolog

Example. *"The law says that it is a crime for a New Zealander to sell alcohol to minors. The girl Lucy is 15 years old and has some beers. All of the beers were sold to her by David, who is a New Zealander."* Prove that David is guilty.

- *"it is a crime for a New Zealander to sell alcohol to minors"*

$$(1)\ Crime(x) \leftarrow NZ(x) \wedge Alcohol(y) \wedge Sells(x, y, z) \wedge Minor(z)$$

- *"Lucy is 15 years old and has some beers"*

$$(2)\ Owns(Lucy, B), (3)\ Beers(B), (4)\ Under17(Lucy)$$

- *"All of the beers were sold to her by David"*

$$(5)\ Sells(David, x, Lucy) \leftarrow Beers(x) \wedge Owns(Lucy, x)$$

- Beers are alcohol

$$(6)\ Alcohol(x) \leftarrow Beers(x)$$

- Anyone younger than 17 years old is a minor.

$$(7)\ Minor(x) \leftarrow Under17(x)$$

- "David, who is a New Zealander"

$$(8)\ NZ(David)$$

- Query: $Ask(\mathrm{KB}, Crime(David))$

Prolog program:

```
crime(X) :- nz(X), alcohol(Y), sells(X,Y,Z), minor(Z).
owns(lucy,b).
beers(b).
under17(lucy).
sells(david,X,lucy) :- beers(X), owns(lucy,X).
alcohol(X) :- beers(X).
minor(X) :- under17(X).
nz(david).
```

Query:

```
?- crime(david).
true.
?- crime(lucy).
false.
```

Prolog Program

Basic Syntax. A Prolog program implements a first-order knowledge base:

- Facts: Atoms

  E.g. `rainy(dunedin).` `cold(dunedin).` `play.`

- Rules: Head :- $\text{Body}_1, \text{Body}_2, \ldots, \text{Body}_k$.

  E.g. `snowy(X) :- rainy(X), cold(X).`

Note.

- Constant, function, relation names all start with small case letters. **E.g.** `dunedin`, `david`, `sum`, `sells`, `''ET''`.
- Variable names start with capital letters or "_". **E.g.** `X, Lucy, _x`
- Left implication: "`:-`"
- Conjunction: "`,`"

Rules. A rule in Prolog

$$\text{Head :- Body}_1, \text{Body}_2, \ldots, \text{Body}_k.$$

represents a logical implication.

- Universal quantification: Variables in the head are universally quantified.

```
cold(wellington).
rainy(auckland).
snowy(X) :- rainy(X), cold(X).
```

  The last statement can be viewed as $\forall x\colon (rainy(x) \wedge cold(x)) \rightarrow snowy(x)$.

- Existential quantification: Variables that only appear in the body are existentially quantified.

```
in(auckland, northIs).
in(wellington, northIs).
sameIsland(X, Z) :- in(X, Y), in(Z, Y).
```

  The last statement can be viewed as

$$\forall x, z\colon\ [\exists y\colon in(x,y) \wedge in(z,y)] \rightarrow sameIsland(x,z).$$

Queries. Queries start with "?-"

```
rainy(dunedin).
cold(dunedin).
rainy(wellington).
cold(wellington).
rainy(auckland).
snowy(X) :- rainy(X), cold(X).
```

- True/false queries:

  E.g. `?- snowy(dunedin).`

  `true.`

- Unification in queries:

  E.g.

  ```
  ?- snowy(C).
  C=dunedin ;
  C=wellington.
  ```

Recursive rules. Prolog allows recursive definitions of predicates:

Example.

```
in(hamilton,waikato).
in(waikato,northIs).
belong(X,Y) :- in(X,Y).
belong(X,Y) :- in(X,Z), belong(Z,Y).

?- belong(hamilton,northIs).
true.
```

Negation as failure. \+ denotes the negation symbol in Prolog.
E.g.

```
man(jim).
man(fred).
woman(X) :- \+(man(X)).
?- woman(jim).
false.
?- woman(X).
false.
```

Note.

- `\+(`*literal*`(x))` is true whenever it is not possible to prove *literal*`(x)` to be true, i.e.,

  `\+(`*literal*`(x))` returns true $\Leftrightarrow$ *literal*`(x)` does not exist in KB.

- `\+(`*literal*`(X))` is true whenever it is not possible to unify `X` with a constant that makes *literal*`(X)` true, i.e.,

  `\+(`*literal*`(X))` returns true $\Leftrightarrow \neg\exists x$: *literal*(`x`).

Example. [a simple KB in Prolog]

```
parent(ann,bob).
parent(abe,bob).
parent(bob,dan).
parent(cat,dan).
parent(ann,ema).
parent(dan,fay).
male(abe).
male(bob).
male(dan).
female(X) :- \+(male(X)).

father(X,Y) :- parent(X,Y), male(X).
mother(X,Y) :- parent(X,Y), female(X).
son(X,Y) :- parent(Y,X), male(X).
sister(X,Z) :- parent(Y,X), parent(Y,Z), female(X).
aunt(X,Z) :- sister(X,Y), parent(Y,Z).
grandfather(X,Z) :- father(X,Y), parent(Y,Z).

ancestor(X,Y) :- parent(X,Y).
ancestor(X,Z) :- parent(X,Y), ancestor(Y,Z).
```

Q1 in Assignment 2

Implement a Prolog predicate `mapcolouring(N,A,WA,B,G,T,M,H,WE)` where the variable `X` should be the corresponding word as indicated in the following map. Submit your answer as `mapcolouring.pl` file.

```
% Knowledge bases
% -------------------------------
different(red, green).
different(red, blue).
different(green, blue).
different(green, red).
different(blue, red).
different(blue, green).

% Your implementation
% -------------------------------
mapcolouring(N,A,WA,B,G,T,M,H,WE):-
    % Your code goes here


```