COMENIUS UNIVERSITY IN BRATISLAVA
FACULTY OF MATHEMATICS, PHYSICS AND INFORMATICS
DEPARTMENT OF APPLIED INFORMATICS

Introduction to Artificial Intelligence

Exercise 7 - Backward-Chaining

November 2, 2022

7. BACKWARD-CHAINING

We will implement logic inference using Backward-Chaining (B-CH) algorithm, and afterwards we use it to find safe squares in the game.

Knowledge Base:

In order for inference to be possible using B-CH, our KB will contain only clauses. For easier implementations we will be processing those only in the form of implications with n premises and exactly one conclusion, for example:

$$A \wedge B \implies C$$
$$C \wedge \neg B \wedge D \implies F$$

Please notice, that we can rewrite an arbitrary implication $A \wedge B \implies C$ as $\top \wedge A \wedge B \implies C$, where $\top$ is a symbol for $True$. Thanks to this trick we can insert facts into KB, i.e. literals, which are true in a form of implication with zero premises:

$$\text{fact: } A \quad \equiv \quad \text{implication: } \top \implies A$$

Individual implications in KB can also be called as clauses, sentences or rules. The $conclusion$ is then the "head of the rule".

Backward-Chaining:

Input into B-CH algorithm is a literal q, which we want to prove, we want to show that $KB \models q$. The algorithm gradually tries to apply all the rules, and it proves premises of these rules. The algorithm can be summarized as:

1. input: literal q
2. find all rules $R_1, R_2, \ldots$ with q as their conclusion
3. prove **at least one** rule R_i by proving all of its premises (using B-CH).
4. if (3) succeeded, then $KB \models q$, otherwise $KB \not\models q$

When proving a sentence, we need to be careful because of the cycles, which can arise. For example if we want to prove A our KB is:

$$A \implies B$$
$$B \implies A$$

The resulting algorithm which takes into account cyclic rules is quite close to back-tracking or depth first search.

Program:

Program is divided into three files:
Do not modify *logic.py*, it contains prepared classes for literals and clauses.:

- $Literal$: simple literal, can be negated, but does not need to be ($\neg A$ or A)
- $Fact$ and $Implication$: classes for clauses

The file **kb.py** contains the class KB - representation of the Knowledge base. The most important parts are:

- $self.clauses$: list of clauses, which are in KB
- $self.tell(clause)$: function, which "tells" KB a new information. In fact, it just adds a clause (implication or a fact) into $self.clauses$
- $self.ask(goal\_literal)$: function, which asks KB for some knowledge. Here you will write the B-CH algorithm. Attention: the argument of the function is **Literal**, and not a **Fact** (clause)!

Examples of this are at the end of the file, check them out! If you do not understand some part of prepared code, ask us.

The file *wumpus.py* contains console implementation of Wumpus World (from lesson n.6). The possible actions are move [up/down/left/right], shoot [up/down/left/right], climb, the goal is to find gold, kill the Wumpus and climb out of the cave. There is an implementation of interactive player $choose\_action\_interactive$, which always writes safe moves and asks for the next action - for example "$moveup$", "$moveU$", "$shootdown$", "$climb$".
You do not need to modify this file. After a correct implementation of the B-CH you should be able to test, whether the player can find the gold while avoiding the pits and the wumpus using only the safe moves provided by the B-CH.

Task 1 (1p): Finish $KB.ask(goal\_literal)$, where you implement the B-CH algorithm for inference without cyclic rules.
Task 2 (1p): Finish $KB.ask(goal\_literal)$ in a way, that it works even if cyclic rules are used.