Middle East Technical University | Department of Computer Engineering

CENG 462

Artificial Intelligence

Fall '2024-2025

Homework 2

Due date:12 December 2024, Thursday, 23:55

1 Objectives

This assignment aims to deepen your understanding of First Order Predicate Logic, focusing on theorem proving techniques using Forward Chaining and Backward Chaining. You will gain experience reasoning with logic systems constrained to Horn clauses.

2 Problem Definition

In this assignment, you are required to implement two Python functions, `forward_chaining` and `backward_chaining`, to serve as theorem provers for First Order Predicate Logic using Forward Chaining and Backward Chaining techniques. The theorem to be proven will always be in Horn clause form. The functions will determine whether a given theorem is derivable and will print specific outputs depending on the chaining method used:

1. `forward_chaining(kb, q)`:
   - **Inputs:** A knowledge base (`kb`) containing facts and rules, and a query (`q`).
   - **Output:** If the theorem is provable, print the list of inferred facts leading to the conclusion. If the theorem is not provable, print `"Query cannot be proven."`.
2. `backward_chaining(kb, q)`:
   - **Inputs:** A knowledge base (`kb`) containing facts and rules, and a query (`q`).
   - **Output:** If the theorem is provable, print the sequence of triggered Horn clauses that lead to the conclusion. If the theorem is not provable, print `"Query cannot be proven."`.

- By convention in FOPL:
  - **Variables** must start with a **lowercase letter**.
  - **Constants**, **predicate names** must start with an **uppercase letter**.
- In the given clauses:
  - `<-` is used to denote "if" in rules (e.g., `r(x) <- s(x), t(x)`).
  - `,` is used to separate multiple conditions in a rule (e.g., `s(x), t(x)`).

3 Sample Input/Output

```
>>> forward_chaining(
    ["Partner(x,y) <- Loves(x,y), Loves(y,x)", "Happy(y) <- Gift(x,z), Partner(x,y)",
    "Loves(Feyza,Can)",
    "Loves(Can,Feyza)",
    "Gift(Can,z)"],
    "Happy(Feyza)"
)
['Partner(Can,Feyza)', 'Partner(Feyza,Can)', 'Happy(Feyza)']

>>> forward_chaining(
    ["Q(x) <- P(x)", "P(x) <- L(x), M(x)", "M(x) <- B(x), L(x)",
     "L(x) <- A(x), P(x)", "L(x) <- A(x), B(x)", "A(John)", "B(John)"],
    "Q(John)"
)
['P(John)', 'M(John)', 'L(John)', 'Q(John)']

>>> backward_chaining(
    ["Q(x) <- P(x)", "P(x) <- L(x), M(x)", "M(x) <- B(x), L(x)",
     "L(x) <- A(x), P(x)", "L(x) <- A(x), B(x)", "A(John)", "B(John)"],
    "Q(John)"
)
[
'L(John), M(John) -> P(John)',
'B(John), L(John) -> M(John)',
'A(John), B(John) -> L(John)',
'P(John) -> Q(John)']

>>> forward_chaining(
    ["Q(x) <- P(x)", "P(x) <- L(x), M(x)", "M(x) <- B(x), L(x)",
     "L(x) <- A(x), P(x)", "L(x) <- A(x), B(x)", "A(John)"],
    "Q(John)"
)
Query cannot be proven.

>>> backward_chaining(
    ["Q(x) <- P(x)", "P(x) <- L(x), M(x)", "M(x) <- B(x), L(x)",
     "L(x) <- A(x), P(x)", "L(x) <- A(x), B(x)", "A(John)"],
    "Q(John)"
)
Query cannot be proven.

>>> backward_chaining(
    ["Criminal(x) <- American(x), Weapon(y), Sells(x,y,z), Hostile(z)",
     "Weapon(x) <- Missile(x)",
     "Missile(M)",
     "Owns(Nono,M)",
     "Sells(West,x,Nono) <- Owns(Nono,x), Missile(x)",
     "Hostile(x) <- Enemy(x,America)",
     "American(West)",
```

```
     "Enemy(Nono,America)"],
    "Criminal(West)"
)

[
'Criminal(West) <- American(West), Weapon(y), Sells(West,y,z), Hostile(z)',
'Missile(M) <- Weapon(M)',
'Owns(Nono,M), Missile(M) <- Sells(West,M,Nono)',
'Enemy(Nono,America) <- Hostile(Nono)']
```

4 Regulations

1. **Programming Language:** You must code your program in Python 3.
2. **Allowed Libraries:** You may use the `re` package and the `count` function from the `itertools` package. You can ask for additional packages.
3. **Implementation:** Adhere strictly to the function signatures provided in the problem definition.
4. **Late Submission:** No late submissions are allowed.
5. **Cheating: Zero tolerance policy for cheating.** Sharing code or using external sources without proper acknowledgment will result in penalties as per university regulations.
6. **Evaluation:** Your program will be evaluated automatically using a "white-box" technique. Your outputs may be slightly different than sample input/output (ordering or intermediate steps, variable or constants in horn clauses. For example you can print `Missile(M) <- Weapon(M)` as `Missile(y) <- Weapon(y)`). It's okay as long as you are implementing the algoritms correctly.
7. **Submission:** Submit your assignment via OdtuClass. Upload a **single** Python file named <**your-student-id**>_**hw2.py** (e.g., `e1234567_hw2.py`).