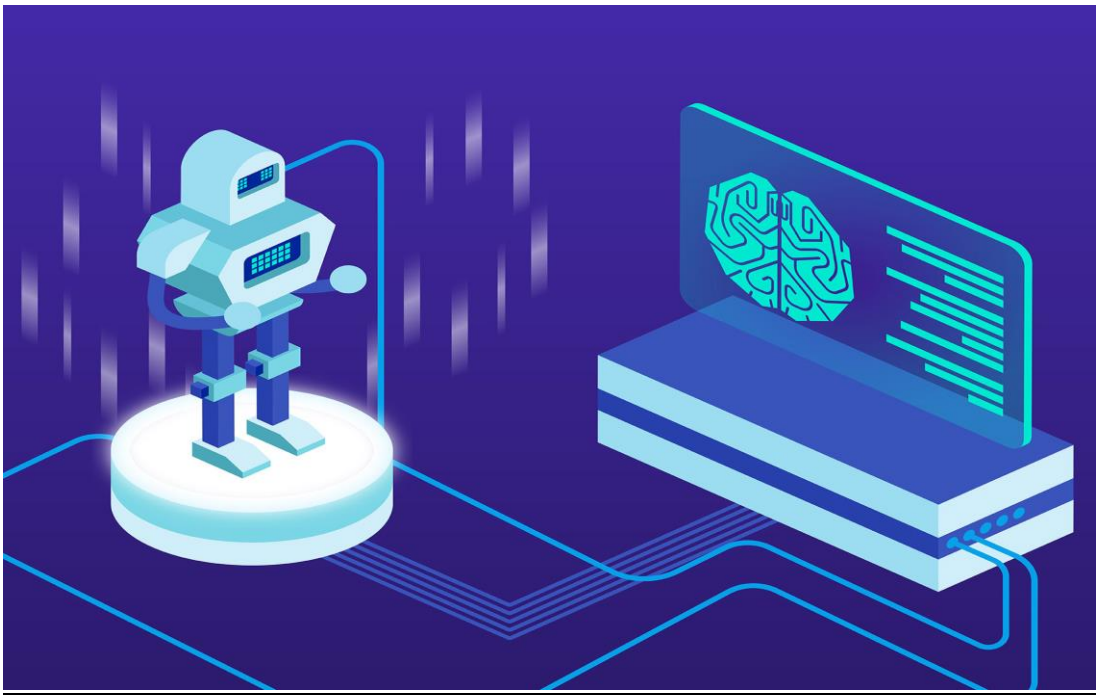
République Algérienne Démocratique et Populaire
Ministère de l'Enseignement Supérieur
et de la Recherche Scientifique
ECOLE SUPÉRIEURE EN INFORMATIQUE
8 Mai 1945 - Sidi-Bel-Abbès

الجمهورية الجزائرية الديمقراطية الشعبية
وزارة التعليم العالي والبحث العلمي
المدرسة العليا للإعلام الآلي
8 ماي 1945 - سيدي بلعباس

# Homework: Expert System

Knowledge Engineering



First Name: Malek Aya

Last Name: Sid El Mrabet

Group: G01 2nd Year Second Cycle SIW

College year: 2023/2024

ESI-SBA

# 1. Objective

The objective of this lab is to study and implement an expert system that operates using both forward chaining and backward chaining. The system will be designed to enrich a given fact base by applying rules from a rule base.

# 2. Code structure

## 2.1 Rule base

The rule base is represented by a set of rules loaded from a file. Each rule follows a specific structure: “R1: IF (List of premises) THEN (List of actions)”

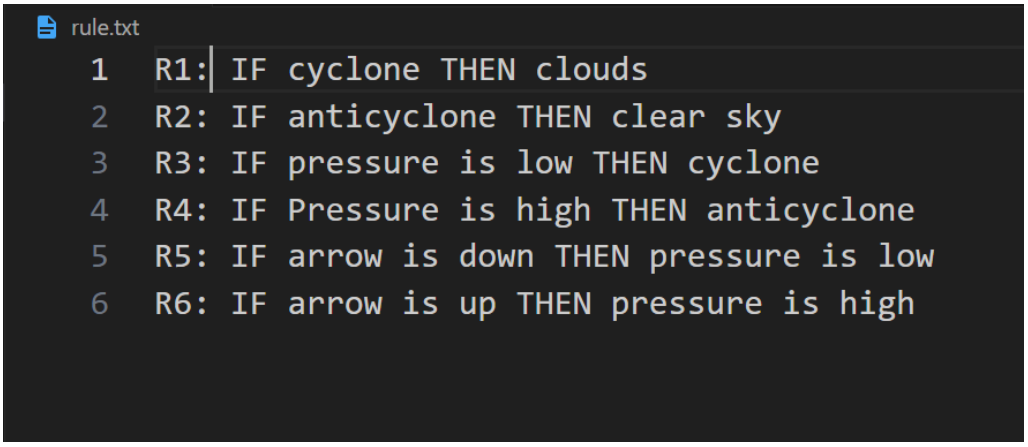



Figure 1: Rule base screenshot

Function which determines actions and premises:

```
def parse_rule(self, rule):
    rule_parts = rule.split(':')
    a =  rule_parts[1].split(' THEN ')
    b = a[0].strip().split('IF ')
    premises = b[1]
    actions =  a[1].strip()
    return premises, actions
```

Figure 2: Rule function screenshot

## 2.2 Expert System class

- Represents the main expert system.
- Initialized with a rule base file and an initial fact base.
- Load the rule base file to create a list of Rule objects.

```python
class ExpertSystem:

    def __init__(self, rule_file , initial_facts):
        self.rule_base = self.load_rules(rule_file)
        self.fact_base =initial_facts

    def load_rules(self, rule_file):
        with open(rule_file, 'r') as file:
            rule_lines = file.readlines()

        rule_base = []
        for line in rule_lines:
            rule_base.append(line.strip())

        return rule_base
```

Figure 3: Expert system class screenshot

## 2.3 Forward chaining

The forward chaining process involves:

- Detecting rules with verified premises (filtering).
- Selecting a rule to apply.
- Applying the rule and disabling it.
- Repeating until no more applicable rules.

```python
def apply_forward_chaining(self, rules_used,fact, base):

while True:
    rule_applied = False
    for rule in base:
            premise, action = self.parse_rule(rule)
            if premise in fact:
                rules_used.append(rule.split(':')[0])
                fact.append(action)
                base.remove(rule)
                rule_applied = True

    if not rule_applied:
        break

self.print_system_status(fact,rules_used)
```

Figure 4: forward chaining function screenshot

## 2.4 Backward chaining

The backward chaining process involves:

- Placing the initial goal at the top of a stack. => (init_goal)
- Detecting rules that conclude this goal. => (find_rules_for_goal)
- Resolving conflicts (if they exist). => (selected_rule)
- Applying the rule, where the elements of the premises become sub-goals to achieve.
- Stopping when the stack is empty or no applicable rules remain.

```
def apply_backward_chaining(self, goal , rules_used ,fact , base):
        rules = self.find_rules_for_goal(goal)
        if not rules:
            print(f"Goal cannot be achieved.")
            return

        selected_rule = rules[0]
        rules_used.append(selected_rule)
        self.apply_rule_backward(selected_rule,  rules_used,fact, base )

def find_rules_for_goal(self, goal):
    rules_for_goal = []
    for rule in self.rule_base:
        _, actions = self.parse_rule(rule)

        if goal in actions:
            rules_for_goal.append(rule)
    return rules_for_goal


def apply_rule_backward(self, rule ,rules_used ,fact, base):
    premises, actions = self.parse_rule(rule)
    fact.append(actions)
    if (premises in self.fact_base) :
        for_rules_used = []
        for i in rules_used :
            r=i.split(':')
            for_rules_used.append(r[0])
        self.print_system_status(fact,for_rules_used[::-1])
    else :
       base.remove(rule)
       self.apply_backward_chaining(premises ,rules_used ,fact, base)
```

Figure 5: backward chaining functions screenshot

# 3. Output

```
################ main code ##################
initial_facts= ['pressure is low']
rule_file = 'C:/Users/ayama/OneDrive/Bureau/IC/tp2/rule.txt'
system = ExpertSystem(rule_file , initial_facts)
goal = 'clouds'


rules_used = []
print("************************Backward chaining************************\n")
print(f"the goal is :"+ str(goal)+"\n")
system.apply_backward_chaining(goal ,rules_used ,system.fact_base.copy(), system.rule_base.copy())

rules_used = []
print("************************forward chaining************************\n")
system.apply_forward_chaining(rules_used, system.fact_base.copy(), system.rule_base.copy())
```

Figure 5: main code screenshot

```
************************Backward chaining************************

the goal is :clouds

New Fact Base: ['pressure is low', 'clouds', 'cyclone']

Rules order :['R3', 'R1']

************************forward chaining************************

New Fact Base: ['pressure is low', 'cyclone', 'clouds']

Rules order :['R3', 'R1']
```

Figure 6: output screenshot

# 4. Explanation

## 4.1 Forward chaining

- Facts base: [’pressure is low’, ’cyclone’, ’clouds’]
  – The expert system started with the initial fact base, which was ”pressure is low.”
  – By applying rules iteratively, it derived new facts, including ”cyclone” and ”clouds.”
  – The final fact base after forward chaining consists of ’pressure is low’, ’cyclone’, and ’clouds.’

- Rule order: [R3, R1]

– The system applied two rules during forward chaining.
– Rule 3 was applied first, resulting in the addition of ’cyclone’ to the fact base.
– Rule 1 was then applied, leading to the inclusion of ’clouds’ in the fact base.

## 4.2 Backword chaining

- New Fact base: [’pressure is low’, ’cyclone’, ’clouds’]
  – The backward chaining process started with a specific goal, in this case, ”clouds.”
  – By working backward from the goal, the system identified that ’cyclone’ was needed.
  – The process continued until the system reached the initial fact base.
  – The final fact base after backward chaining is the same as the one after forward chaining.

- Rule order: [R3, R1]
  – Similar to forward chaining, the backward chaining process applied Rule 3 first and then Rule 1.
  – The order of rule application during backward chaining matches that of forward chaining.

# 5. Conclusion

This lab focused on the implementation of an expert system capable of both forward and backward chaining. The rule base was read from a file and modeled appropriately. The system successfully demonstrated the ability to enrich a fact base by applying rules, and the output included the set of rules used and the updated fact base.