Technical University of Moldova
Faculty of Computers, Informatics and Microelectronics
Department of Software and Automation Engineering

Laboratory work No. 1

Expert Systems

Elaborated: **Dan Hariton**
st. FAF-211

Checked: **Elena Graur**
un. assistant.

Chişinău, 2024

1 THE TASK OF THE LABORATORY WORK

Finally, your dream came true and you have landed a job at “HeinleinAI” – the biggest AI company in the whole Luna-City! You’re on a testing period so you will want to make sure that you do your best. While you are daydreaming about free lunch, the mentor comes and hands you your first task – you’ll need to build an expert system. They say that while going through the central library database they stumbled upon this ancient approach that looked like it could solve a particular lunar problem – detecting tourists. The tourists in Luna-City are a big source of income and while almost every salesman and hotel administrator can easily tell one from a Loonie, our mentor researches a more systematic way of detecting them. While this would prove a trivial task to the city’s main computer, resources are scarce these days and so you will be researching alternative, more special approaches. For starters, research the types of tourists that visit Luna-City and collect a knowledge base of at least 5 tourist types and the criteria by which they can be distinguished from Loonies and between themselves (e.g. clothes, accent, gait, height or opinion on politics). After your knowledge base is done, create a system that would allow the user to answer some questions about a possible tourist. If the set of given answers matches a type of tourist from the knowledge base, this should be the system’s answer. If on the other hand, the system determines that the person in question is a Loonie, the answer should be returned accordingly. Make sure to consider the case when the set of answers does not find a match in the knowledge base (highly improbable if you’ve done your research well). Watch out! The Company has a very strict grading policy and guidelines which you should follow! Both of them are described on the next page. If you want to be sure that the company appreciates your work, follow them rigorously.

2 THE PROGRESS OF THE WORK

2.1 Task 1 Define at least 5 types of tourists that visit Luna-City. Draw the Goal Tree representing these types of tourists

I defined five types of tourists visiting Luna-City based on traits and attributes [1]:

- **Lonie**: A local resident who lacks special skills but has local knowledge and city map.
- **Genin**: A tourist with good balance and basic training.
- **Chunin**: A tourist who is a calm person, has team leadership, and possesses mission skills.
- **Jonin**: A tourist with great wisdom, nature link, and mission skills.
- **Sage**: A tourist with multiple jutsu, strong mental strength, and skilled ninja abilities.
- **Kage**: A tourist with a strong mental state, a written book, and wears a cloak.

2.2 Task 2 Implement the rules from the defined tree in Task 1 in your code (use the IF, AND, OR and THEN rules which are already implemented).

For Task 2, I implemented the rules from the goal tree defined in Task 1 using the logical structure of IF, AND, OR, and THEN. These rules follow the relationships between traits and tourist types as described in the goal tree. For example, a tourist is identified as a Jonin if they have mission skills and are an upper level, which is determined by having a nature link and great wisdom. Similarly, a tourist can be classified as a skilled **ninja** if they have either multiple jutsu or ultimate jutsu, using OR logic. For writing this rule I also used the ZOOKEPEER rules example

```
IF( AND( '(?x) has nature link',
            '(?x) has great wisdom' ),
       THEN( '(?x) is a upper level' )),

   IF( AND( '(?x) has mission skills',
            '(?x) is a upper level'),
       THEN( '(?x) a Jonin' )),

   IF( OR( '(?x) has multiple jutsu' ,
            '(?x) has ultimate jutsu' ),
       THEN( '(?x) is a skilled ninja' )),
```

2.3 Task 3 If you are using the provided code, check how the Forward Chaining algorithm works and illustrate an example. If you are implementing your own code, implement the Forward Chaining algorithm yourself.

For Task 3**,** I implemented the Forward Chaining algorithm[2] to dynamically gather facts about tourists. The algorithm starts by asking a multiple-choice question to gather the first fact. After confirming this fact, the system checks for rules that contain the fact in their antecedents. Once a rule is found, the system asks an open-ended question to gather more details and confirm the next fact.

For example, if a tourist has the trait "has no skill," the system might find a rule that leads to the conclusion that the tourist is a city guide. The algorithm proceeds by asking further questions, using open-ended formats, until a tourist type is confirmed. In the final step, the algorithm asks a yes/no question to confirm the tourist type and print the result, such as "Tourist confirmed as a Lonie."

```
fact = generate_multiple_choice_question(subject)
    data.add(fact)

    # Step 2: Find the rule that contains the confirmed first fact
    for rule in rules:
        antecedents = rule.antecedent()
        if fact in [antecedent.replace('(?x)', subject, 1) for antecedent in antecedents]:
            # We found a rule containing the confirmed fact, now ask about the second antecedent
            other_fact = [antecedent for antecedent in antecedents if antecedent.replace('(?x)',
subject, 1) != fact][0]
            open_ended_question = generate_open_ended_question(subject, rule.consequent()[0])
            print(open_ended_question)

            # Step 3: Ask an open-ended question to get more details
            answer = input(f"Provide more details about {subject}:").strip().lower()
            if other_fact.replace('(?x)', subject, 1) in answer:
                data.add(other_fact.replace('(?x)', subject, 1))

                # Step 4: Find the rule where the consequent becomes the antecedent for the next
rule
                consequent = rule.consequent()[0].replace('(?x)', subject, 1)
                for next_rule in rules:
                    if consequent in [antecedent.replace('(?x)', subject, 1) for antecedent in
next_rule.antecedent()]:
                        # Step 5: Confirm the tourist type using a yes/no question
                        next_antecedent = next_rule.antecedent()[0].replace('(?x)', subject, 1)
                        yes_no_question = generate_yes_no_question(subject, next_antecedent)
                        confirmation = input(yes_no_question).strip().lower()
                        if confirmation == 'yes':
                            tourist_type = next_rule.consequent()[0].replace('(?x)', subject, 1)
                            print(f"Tourist confirmed as {tourist_type}.")
```

2.4 Task 4 Implement the Backward Chaining algorithm for the Goal Tree.

For Task 4, I implemented the Backward Chaining algorithm[2] to confirm hypotheses by asking dynamic questions. The algorithm starts with a hypothesis, such as "Naruto is a Jonin" and works backward to verify the conditions that would lead to this conclusion.

First, it checks if the hypothesis is already known, and if not, it searches for a rule where the hypothesis is the consequent. The system then asks a yes/no question to confirm the first antecedent. If the antecedent is confirmed, the fact is added to the known facts.

```
for rule in rules:
        consequent = rule.consequent()[0].replace('(?x)', subject)

        if consequent == hypothesis:
            if verbose:
                print(f"Rule found that can prove the hypothesis: {rule}")
            antecedents = rule.antecedent()

            if isinstance(antecedents, str):
                antecedents = [antecedents]  # Ensure antecedents is a list

            # Step 2: Ask yes/no question for the first antecedent
            first_antecedent = antecedents[0].replace('(?x)', subject)
            if first_antecedent not in known_facts:
                yes_no_question = generate_yes_no_question(subject, first_antecedent)
                confirmation = input(yes_no_question).strip().lower()
                if confirmation == 'yes':
                    known_facts.add(first_antecedent)
```

Next, it finds another rule where the second antecedent is a consequent. The system asks a multiple-choice question to verify one of the antecedents from this rule.

```
second_antecedent = antecedents[1].replace('(?x)', subject)
            for next_rule in rules:
                next_consequent = next_rule.consequent()[0].replace('(?x)', subject)
                if second_antecedent == next_consequent:
                    # Step 4: Ask specialized multiple-choice question for one correct antecedent
                    is_correct = generate_backward_multiple_choice_question(subject,
next_rule.antecedent()[0], next_rule.antecedent())
                    if is_correct:
                        known_facts.add(next_rule.antecedent()[0].replace('(?x)', subject))
```

After the multiple-choice step, the system follows up with an open-ended question to confirm the remaining antecedent. If both antecedents are confirmed, the hypothesis is proven, and the system prints a confirmation, such as "Naruto is a Jonin confirmed."

```
 remaining_antecedent = [ant.replace('(?x)', subject) for ant in next_rule.antecedent() if ant !=
next_rule.antecedent()[0]][0]
                        final_answer = input(f"Provide more details about {subject}:
").strip().lower()
                        if remaining_antecedent in final_answer:
                            known_facts.add(remaining_antecedent)
                            # Add the second antecedent to known facts
                            known_facts.add(second_antecedent)
                            # If both antecedents are proven, the hypothesis is true
                            known_facts.add(hypothesis)
                            print(f"{subject} is a {hypothesis.split()[2]} confirmed.")
```

2.5 Task 5 Implement a system for generating questions from the Goal Tree. Have at least 3 types of questions (e.g. yes/no, multiple choice, etc).

For Task 5, we created a system to generate three types of questions: multiple-choice, open-ended, and yes/no**.**

The system begins by asking a multiple-choice question to gather initial facts about the tourist, such as whether they have a specific trait (e.g., "has no skill" or "has multiple jutsu"). After confirming a fact, the system asks an open-ended question to collect more details and refine the search.

```
def generate_multiple_choice_question(subject):
    print(f"Choose one of the following traits about {subject}:")
    choices = […# the choices we selected]
    for i, choice in enumerate(choices, 1):
        print(f"{i}. {choice}")

    choice_index = int(input("Enter the number of your choice: ")) - 1
    return choices[choice_index]

def generate_open_ended_question(subject, consequent):
    return f"Tell us more about how {consequent.replace('(?x)', subject, 1)}?"

def generate_yes_no_question(subject, antecedent):
    return f"Does {antecedent.replace('(?x)', subject, 1)}? (yes/no)"
```

Finally, a yes/no question is used to verify key facts and confirm the tourist type.

For backward chaining the system works differently since we follow a different path to find a solution. It uses both **yes/no** and **open-ended** functions but has a specific **multiple-choice** function implement jut for backward chaining.

```
    distractions = […]
    distractions = [d.replace('(?x)', subject) for d in distractions if d != correct_option]
    distractions = random.sample(distractions, min(3, len(distractions)))
    options = distractions + [correct_option]
    random.shuffle(options)
    print(f"Choose one of the following traits about {subject}:")
    for i, option in enumerate(options, 1):
        print(f"{i}. {option}")
    choice_index = int(input("Enter the number of your choice: ")) - 1
    return options[choice_index] == correct_option  # Return True if the correct option is chosen
```

2.6 Task 6 Wrap up everything in an interactive Expert System that will dynamically ask questions based on the input from the user. Both Forward Chaining and Backward Chaining should be working.

For Task 6, I wrapped everything into an interactive expert system that dynamically asks questions based on user input. The system allows the user to choose between forward chaining and backward chaining, then proceeds to ask questions to either gather facts or confirm a hypothesis.

When the user selects forward chaining, the system uses dynamic questions to collect information step by step and apply rules to identify the tourist type. In the case of backward chaining, the system works backward from the user's hypothesis, confirming antecedents through yes/no, multiple-choice, and open-ended questions until it either proves or disproves the hypothesis.

This interaction ensures that both forward and backward chaining processes work seamlessly, guiding the user through dynamic question-based interactions until the tourist type is confirmed.

2.7 Task 7 Format the output and questions in a human-readable format (they should respect the grammatical rules of the English language).

This task was implemented along the way, while I implemented our questions. It was easier for me to test the system and how a user would interact with it.

CONCLUSIONS

In this laboratory work, I successfully developed an Expert System capable of identifying different types of tourists in Luna-City using forward and backward chaining algorithms. By defining a good ninja-based knowledge base with multiple types of tourists and their unique traits, the system efficiently detects whether an individual is a tourist/ninja or a Lonie. The question-generation system adds interactivity, guiding the user through fact-gathering or hypothesis verification processes.

Throughout the project, I integrated logical structures like IF, AND, OR, and THEN into the rule-based engine, ensuring accurate identification of tourist types. Additionally, I designed three types of questions: multiple-choice, yes/no, and open-ended, creating a seamless interaction between the system and the user.

I leaned about Expert Systems and how they used to work and how forward and backward chaining is working, which is valuable knowledge.

This project demonstrated how an Expert System can use goal trees and rule-based logic to systematically solve real-world problems. The final result is an efficient, user-friendly expert system that mimics how humans might deduce conclusions in a structured, logical manner.

BIBLIOGRAPHY

1. Lecture 3: Reasoning: Goal Trees and Rule-Based Expert Systems | Artificial Intelligence | Electrical Engineering and Computer Science | MIT OpenCourseWare." *MIT OpenCourseWare*, 2024, Accessed 9 September 2024. https://ocw.mit.edu/courses/6-034-artificial-intelligence-fall-2010/resources/lecture-3-reasoning-goal-trees-and-rule-based-expert-systems/

2. "Forward Chaining and Backward Chaining in AI - Javatpoint." *Javatpoint.*, Accessed 13 September 2024 www.javatpoint.com/forward-chaining-and-backward-chaining-in-ai