Technical University of Moldova

Faculty of Computers, Informatics and Microelectronics

Department of Software Engineering and Automation

Fundamentals of Artificial Intelligence

Laboratory work #1

Expert Systems

Author:
Daniel Pogorevici
std. gr. FAF-202

Supervisor:
Diana Marusic

Chișinău 2023

1 Task

1. Define 5 types of tourists. Draw the Goal Tree representing these types of tourists.
2. Implement the rules from the defined tree in task 1 in your code (use the IF, AND, OR, THEN etc rules which are already implemented in the code).
3. If you are using the provided code, check how the Forward chaining algorithm works and show an example. If you are implementing your own code, implement the Forward chaining algorithm yourself.
4. Implement the Backward chaining algorithm for the Goal tree.
5. Implement a system for generating questions from the goal tree. Have at least 2-3 types of questions (e.g yes/no, multiple choice, etc).
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.
7. Format the output and questions to human readable format.

2 Results

2.1 First Task

Define 5 types of tourists. Draw the Goal Tree representing these types of tourists.

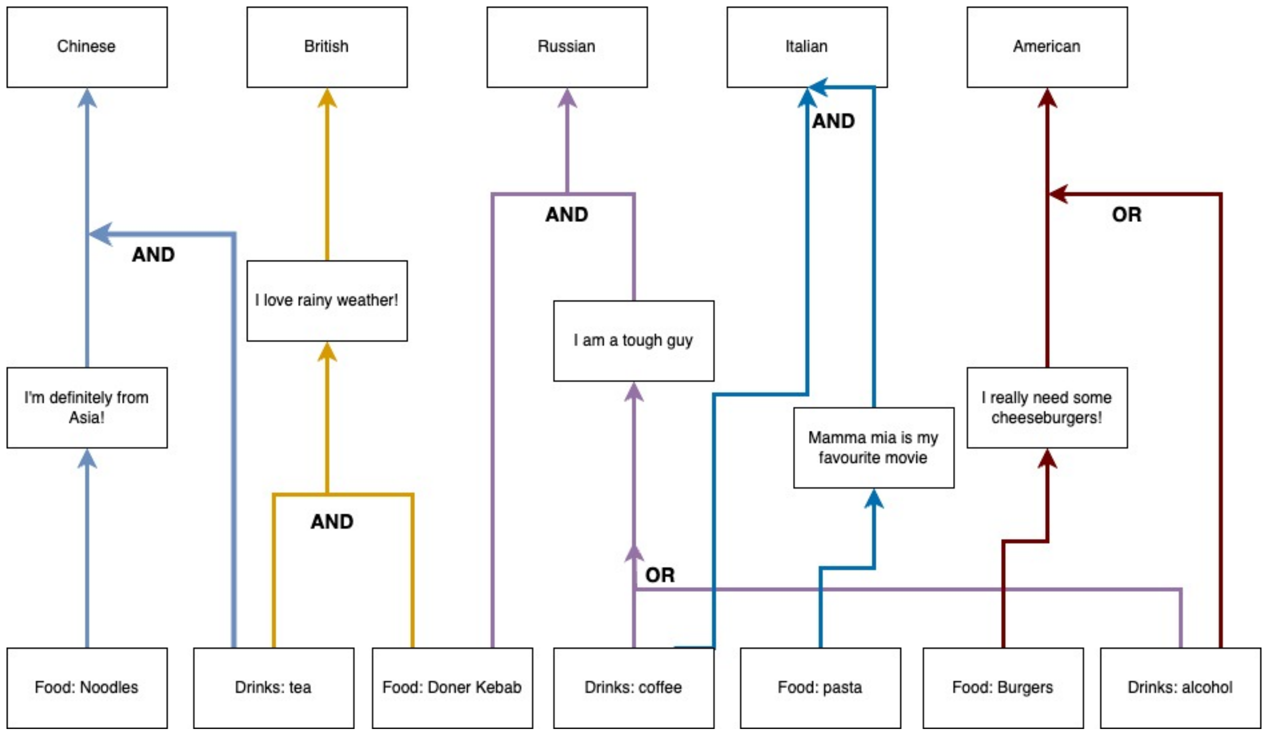

Figure 1: Tourist Tree

2.2 Second Task

Implement the rules from the defined tree in task 1 in your code (use the IF, AND, OR, THEN etc rules which are already implemented in the code).

```python
from production import IF, AND, THEN, OR

```

```python
# Define a class to represent different tourist types with their conditions and
    conclusions
class TouristType:
    def __init__(self, name, food_condition=None, drink_condition=None,
    other_condition=None, conclusion=None):
        self.name = name
        self.food_condition = food_condition
        self.drink_condition = drink_condition
        self.other_condition = other_condition
        self.conclusion = conclusion

# Define common conditions that are shared among multiple tourist types
class Common:
    loves_doner = "(?x) favorite food is doner"
    loves_alcohol = "(?x) favorite drink is alcohol"
    loves_coffee = "(?x) favorite drink is coffee"
    loves_tea = "(?x) favorite drink is tea"

# Define conditions and conclusions for the "Asian" tourist type
class Asian:
    loves_noodles = "(?x) favorite food is noodles"
    is_asian = "(?x) is from Asia"
    conclusion = "(?x) is an Asian tourist"

# Define conditions and conclusions for the "British" tourist type
class British:
    loves_tea = "(?x) favorite drink is tea"
    loves_doner = "(?x) favorite food is doner"
    is_english = "(?x) loves rainy weather"
    conclusion = "(?x) is a British tourist"

# Define conditions and conclusions for the "Russian" tourist type
class Russian:
    loves_doner = "(?x) favorite food is doner"
    loves_coffee = "(?x) favorite drink is coffee"
    is_tough = "(?x) is a tough guy"
    loves_alcohol = "(?x) favorite drink is alcohol"
    conclusion = "(?x) is a Russian tourist"

# Define conditions and conclusions for the "Italian" tourist type
class Italian:
    loves_coffee = "(?x) favorite drink is coffee"
    loves_pasta = "(?x) favorite food is pasta"
    mamma_mia = "(?x) loves to watch Mamma Mia"
    conclusion = "(?x) is an Italian tourist"

# Define conditions and conclusions for the "American" tourist type
class American:
    loves_burgers = "(?x) favorite food is burgers"
    loves_alcohol = "(?x) favorite drink is alcohol"
    needs_burgers = "(?x) really needs burgers"
    conclusion = "(?x) is an American tourist"

# Define instances of TouristType for each tourist type with their specific
    conditions and conclusions
asian = TouristType(
    name="Asian",
    food_condition=Asian.loves_noodles,
    drink_condition=Common.loves_tea,
    other_condition=Asian.is_asian,
    conclusion=Asian.conclusion
)

british = TouristType(
    name="British",
    food_condition=Common.loves_doner,
    drink_condition=Common.loves_tea,
    other_condition=British.is_english,
    conclusion=British.conclusion
)

russian = TouristType(
    name="Russian",
    food_condition=Common.loves_doner,
    drink_condition=Common.loves_coffee,
    other_condition=Russian.is_tough,
    conclusion=Russian.conclusion
)

italian = TouristType(
    name="Italian",
    food_condition=Italian.loves_pasta,
    drink_condition=Common.loves_coffee,
    other_condition=Italian.mamma_mia,
    conclusion=Italian.conclusion
)

american = TouristType(
    name="American",
    food_condition=American.loves_burgers,
    drink_condition=Common.loves_alcohol,
    other_condition=American.needs_burgers,
    conclusion=American.conclusion
)

# Define rules for each tourist type
asian_rules = (
    IF(asian.food_condition, THEN(asian.other_condition)),
    IF(AND(asian.other_condition, asian.drink_condition), THEN(asian.conclusion)),
)

british_rules = (
    IF(AND(Common.loves_tea, Common.loves_doner), THEN(british.other_condition)),
    IF(british.other_condition, THEN(british.conclusion)),
)

russian_rules = (
    IF(OR(Common.loves_coffee, Common.loves_alcohol), THEN(russian.other_condition)),
    IF(AND(russian.other_condition, Common.loves_doner), THEN(russian.conclusion)),
)

italian_rules = (
    IF(italian.food_condition, THEN(italian.other_condition)),
    IF(AND(italian.other_condition, Common.loves_coffee), THEN(italian.conclusion)),
)

american_rules = (
    IF(american.food_condition, THEN(american.other_condition)),
    IF(OR(american.other_condition, Common.loves_alcohol), THEN(american.conclusion))
    ,
)

# Combine all the rules into a single rule set
TOURIST_RULESET = asian_rules + british_rules + russian_rules + italian_rules +
    american_rules
```

Listing 1: Rules script.

2.3 Third Task

If you are using the provided code, check how the Forward chaining algorithm works and show an example. If you are implementing your own code, implement the Forward chaining algorithm yourself.

```
 def simulate_answers(self, name):
        print(f"Simulating answers for {name}...\n")
        print(f"For DEMO purposes to show forward chain.\n")
        drink_choice = [1, 2]
        food_choice = [2, 3]

        to_chain = [self.drink_list[int(index) - 1].replace(self.var, name) for index
    in drink_choice]
        to_chain += [self.food_list[int(index) - 1].replace(self.var, name) for index
    in food_choice]

        forward = forward_chain(TOURIST_RULESET, to_chain)
        print(f"Forward chain: {forward}")
```

Listing 2: Forward Chaining.

I didn't implement a new Forward chaining algorithm and instead used the provided one. In the upper code is the simulation of the answers for the user. So, I made the drink and food choices, I add them to the chain and perform the forward chain. In the main script can be seen that user can make the choices by himself or simulating them (as already shown)

```
Please write your name: Daniel
Do you want to perform backward or forward chaining? (backward/forward): forward
Daniel, do you want to choose your preferences for drinks and food? (yes/no)

Choose your answer: no
Simulating answers for Daniel...

For DEMO purposes to show forward chain.

Forward chain: ('Daniel favorite drink is coffee', 'Daniel favorite drink is tea', '
    Daniel favorite food is burgers', 'Daniel favorite food is pasta', 'Daniel is a
    tough guy', 'Daniel is an American tourist', 'Daniel is an Italian tourist', '
    Daniel loves to watch Mamma Mia', 'Daniel really needs burgers')

```

Listing 3: Forward Chaining User Flow.

2.4 Fourth Task

Implement the Backward chaining algorithm for the Goal tree

```
    goal_tree = []

    while goal_stack:
        current_goal = goal_stack.pop()

        found_rule = None
        for rule in rules:
            match_res = match(rule.consequent()[0], current_goal)
            if match_res:
                found_rule = rule
                break

        if found_rule:
            goal_tree.append(found_rule.antecedent())
            for antecedent in found_rule.antecedent():
                goal_stack.append(populate(antecedent, match_res))

    return goal_tree

```

Listing 4: Backward Chaining.

I implemented a backward chaining using an iterative approach along with a stack data structure. It starts with an initial hypothesis and a set of rules. The script iteratively pops a goal from the stack and searches for a rule whose conclusion matches the current goal. If a matching rule is found, the script adds the antecedents of that rule (the conditions) to the goal tree. The script continues this process until the goal stack is empty, which means all relevant goals have been explored. The end result is a goal tree that shows how the initial hypothesis can be proven based on the given rules and their conditions.

```
Please write your name: Daniel
Do you want to perform backward or forward chaining? (backward/forward): backward
You can choose from the following hypotheses:
Daniel is an Asian tourist
Daniel is a British tourist
Daniel is a Russian tourist
Daniel is an Italian tourist
Daniel is an American tourist
Please enter the hypothesis you want to test: Daniel is an Italian tourist
Backward chain: [AND('(?x) loves to watch Mamma Mia', '(?x) favorite drink is coffee'
    ), '(?x) favorite food is pasta']
```

Listing 5: Backward Chaining User Flow.

2.5 Fifth Task

Implement a system for generating questions from the goal tree. Have at least 2-3 types of questions (e.g yes/no, multiple choice, etc).

```
def selection_input(self, name):
        # Method to gather user preferences and perform forward chaining
        print(f"{name}, do you want to choose your preferences for drinks and food? (
    yes/no)\n")
        choice = input("Choose your answer: ").strip().lower()

        if choice == "no":
            self.simulate_answers(name)
            print(f"\nDo you wish to continue? \n")
        elif choice == "yes":
            # If user wants to choose preferences, prompt for drink and food choices
            print(f"{name}, pick from the list of your favorite drinks (ex: 1 2 3):\n
    ")
            for index, drink in enumerate(self.drink_list, start=1):
                print(f"{index}: {drink.replace(self.var, name)}")
```

```python

            drink_choice = input("\nChoose your favorite drink(s): ").strip().split()
            to_chain = [self.drink_list[int(index) - 1].replace(self.var, name) for index in drink_choice]

            print(f"\nNow, pick from the list of your favorite foods based on your drink choice (ex: 1 2 3):\n")
            for index, food in enumerate(self.food_list, start=1):
                print(f"{index}: {food.replace(self.var, name)}")

            food_choice = input("\nChoose your favorite food(s): ").strip().split()
            to_chain += [self.food_list[int(index) - 1].replace(self.var, name) for index in food_choice]

            # Perform forward chaining with user preferences
            forward = forward_chain(TOURIST_RULESET, to_chain)
            return self._suggestion(forward, name)

    def _suggestion(self, chained_data, name):
        # Method to suggest tourist group(s) based on forward chaining results
        suggestion_list = []
        for conclusion in self.conclusion_list:
            named_conclusion = conclusion.replace(self.var, name)
            if named_conclusion in chained_data:
                suggestion_list.append(named_conclusion)

        if len(suggestion_list) == 0:
            return f"{name} doesn't belong to any of the tourist groups. (write yes/no to proceed)"
        return f"Does {name} belong to the following tourist group(s): {', '.join(suggestion_list)}? (write yes/no to proceed)"

    def simulate_answers(self, name):
        # Method to simulate answers for demo purposes and perform forward chaining
        print(f"Simulating answers for {name}...\n")
        print(f"For DEMO purposes to show forward chain.\n")
        drink_choice = [1, 2]
        food_choice = [2, 3]

        to_chain = [self.drink_list[int(index) - 1].replace(self.var, name) for index in drink_choice]
        to_chain += [self.food_list[int(index) - 1].replace(self.var, name) for index in food_choice]

        forward = forward_chain(TOURIST_RULESET, to_chain)
        suggestion = self._suggestion(forward, name)
        # print(f"Forward chain: {forward}\n")
        print(f"Suggestion: {suggestion}")
```

Listing 6: System for Generating Questions.

In the following system, users are allowed to input their name, choose either they want to perform forward or backward chaining and pick their preferences for drinks and food and then use forward chaining to suggest tourist group(s) based on those preferences. It also includes a method for simulating answers for demonstration purposes.

2.6 Sixth and Seventh Tasks

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.

Format the output and questions to human readable format.

```
Welcome to the Expert System!

Please write your name: Daniel
Do you want to perform backward or forward chaining? (backward/forward): backward
You can choose from the following hypotheses:
Daniel is an Asian tourist
Daniel is a British tourist
Daniel is a Russian tourist
Daniel is an Italian tourist
Daniel is an American tourist
Please enter the hypothesis you want to test: Daniel is an Italian tourist
Backward chain: [AND('(?x) loves to watch Mamma Mia', '(?x) favorite drink is coffee'
    ), '(?x) favorite food is pasta']
Write 'yes' or 'no': yes
We are glad to help you!
Please write your name: Daniel
Do you want to perform backward or forward chaining? (backward/forward): forward
Daniel, do you want to choose your preferences for drinks and food? (yes/no)

Choose your answer: yes
Daniel, pick from the list of your favorite drinks (ex: 1 2 3):

1: Daniel favorite drink is tea
2: Daniel favorite drink is coffee
3: Daniel favorite drink is alcohol

Choose your favorite drink(s): 1 3

Now, pick from the list of your favorite foods based on your drink choice (ex: 1 2 3)
    :

1: Daniel favorite food is doner
2: Daniel favorite food is pasta
3: Daniel favorite food is burgers
4: Daniel favorite food is noodles

Choose your favorite food(s): 4 2
Does Daniel belong to the following tourist group(s): Daniel is an Asian tourist,
    Daniel is an American tourist? (write yes/no to proceed)

```

Listing 7: System for Generating Questions.

3 Conclusion

In this laboratory work, I've gained valuable insights into the world of AND/OR-trees, forward and backward chaining techniques, and the fundamental principles. To sum it up, while my system may not be as complex as it could be, it is using the rule-based logic and the principles of forward chaining. This experience has laid a strong foundation for upcoming, more substantial projects, making it a significant and instructive learning journey.

In summary, this laboratory experience has provided me with a solid stepping stone for future, more substantial projects. It has been an enlightening and worthwhile learning journey, poised to serve as a launchpad for more advanced ventures down the road.