Rule Based Knowledge Assignment

Abdul Khader, Syed

30 October 2022

```python
with open("./data/file.txt") as f:
    observations = set(f.readline().strip().split(","))
    # Empty Line
    f.readline()
    rules = {}
    # Loop over the remaining lines
    for line in f.readlines():
        # For rules
        if "=>" in line:
            # Split with "=>"
            rule, goal = line.strip().split("=>")
            # If the goal is already present in dictionary, append the
            # new rule_line as a set to the list
            # If goal not present, create an empty list and than
            # append the rule set
            rules.setdefault(goal, []).append(set(rule.split("+")))
        # For Final Goal
        elif line.isalpha():
            finalgoal = line

print(observations)
print(finalgoal)
print(rules)
```

```
{'c', 'k', 'd', 'h', 'e', 'a', 'g'}
q
{'i': [{'k', 'l', 'm'}], 'q': [{'j', 'l', 'i'}, {'b', 'a'}, {'o', 'l', 'n', 'p'}], 'b': [{'c', 'e', 'd'}, {'h',
```

Backward Chainning

Given a `Final Goal` and a set of `observations`, determine if that set of observations leads to the final goal or not.

```python
def backwardChainning(rules_set: dict, observation_set: set, final_goal: str) -> bool:
    """
    Predict the given Final Goal is reachable or not from the given observations.
    """
    # Base Cases
    # If final_goal in observation, return True
    if final_goal in observation_set:
        return True
    # If Goal/Final Goal not in observation
    # Not even in knowledge base, we can't reach it.
    # Return False
    elif final_goal not in rules_set:
        return False

    for rule_set in rules_set[final_goal]:
        for rule in rule_set:
            # Recursion with Depth First Search
            temp_result = backwardChainning(rules_set, observation_set, rule)
            if temp_result == False:
                break # Go to next rule_set
        # If there was no break, which means all rule in the set
        # was present, return True.
        if temp_result: return temp_result

    return temp_result

backwardChainning(rules, observations, finalgoal)
```

True

Forward Chainning

Given a set of **observations** and **knowledge base**, identify the most deep goal possible

```python
def forwardChainning(rules_set: dict, observation_set: set) -> str|None:
    reached_goals = []
    addition = True

    while addition:
        # Loop until there is no addition to observation
        addition = False
        # Go through each goal in the rules
        for goal, rule_list in rules_set.items():
            # Loop through all the set of rules(paths) to reach
            # the goal
            if goal in observation_set:
                continue
            # Check for each observation in the rule list(each line in txt file)
            for rule_set in rule_list:
                for rule in rule_set:
                    # If observation is not present, skip the list
                    if rule not in observation_set:
                        break
                else:
                    # All rule of rule_set is present in observation
                    # So goal acheived, add to observation set
                    observation_set.add(goal)

                    # Add the goal to listr of goals reached.
                    reached_goals.append(goal)

                    # Since there was an addition, we have to do one more run
                    addition = True

                    # Since goal is reached no need to check for further in that list.
                    break

    return reached_goals


obs_copy = observations.copy()
forwardChainning(rules, obs_copy)
```

['b', 'r', 'f', 'q']