29/11/24

LAB-7

Forward Reasoning Algorithm:

function FOL-FC-Ask(KB, α) returns a substitution or false
inputs: KB, the knowledge base, a set of first-order definite clauses
α, the query, an atomic sentence
local variables: new, the new sentences inferred on each iteration

repeat until new is empty:
new ← {}
for each rule in KB do
(p1 ∧ ... ∧ pn ⇒ q) ← STANDARDIZE-VARIABLES(rule)
for each θ such that SUBST(θ, p1 ∧ ... ∧ pn) = SUBST(θ, p'1 ∧ ... ∧ p'n) for some p'1, ..., p'n in KB:
q' ← SUBST(θ, q)
if q' does not unify with some sentence already in KB or new then:
add q' to new
φ ← UNIFY(q', α)
if φ is not fail then return φ
add new to KB
return false

OUTPUT:
Robert is a criminal

Knowledge base (KB):

KB : [

"American (Robert)",

"Enemy (A, America)",

"missile (T1)",

"owns (A, T1)",

"missile (x) => Weapon (x)",

"Weapon (x) ^ sells (Robert, x, A) ^ Hostile (A) => criminal (Robert)",

"Enemy (x, America) => Hostile (x)",

"sells (Robert, T1, A)",

]

query = "criminal (Robert)"

This query can be proved

- missile (x) => weapon (x)
  Substitute x = T1, Inference : Weapon (T1)

- Enemy (x, America) => Hostile (x)
  Fact in KB : Enemy (A, America)
  Substitute x : A, Inference : Hostile (A)

- Weapon (x) ^ sells (p, x, r) ^ Hostile (r) => criminal (p)
  Substitute x = T1, p = Robert, r = A
  Weapon (T1), sells (Robert, T1, A), Hostile (A)
  Inference : criminal (Robert)

g)

Consider a vocabulary with the following symbols:

Occupation (P, O) : Predicate person P has occupation O
Customer (P1, P2) : Predicate person P1 is customer of P2
Boss (P1, P2) : Predicate person P1 is boss of P2

Doctor, Surgeon, Lawyer, Actor : constants denote occupations
Emily, Joe : constants denoting ppl

Using these symbols to write assertions in FLO:

a) Emily is surgeon or lawyer
Occupation (Emily, Surgeon) ∨ Occupation (Emily, Lawyer)

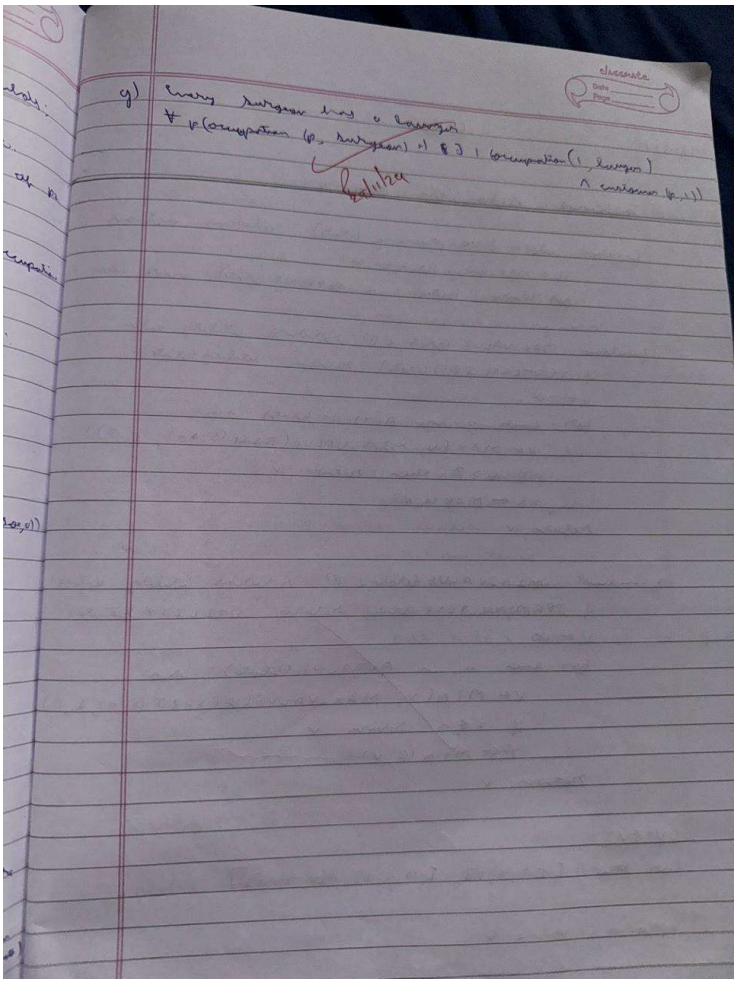
b) Joe is an actor, but has other job
Occupation (Joe, Actor) ∧ ∃O (O ≠ actor ∧ Occupation (Joe, O))

c) All surgeons are doctors
∀ P Occupation (P, Surgeon) ⇒ Occupation (P, Doctor)

d) Joe doesn't have a lawyer
¬ ∃ (P, Lawyer) ∧ Customer (Joe, P)

e) Emily has lost who is lawyer
∃ P Boss (P, Emily) ∧ Occupation (P, Lawyer)

f) There exists a lawyer all of whose customers are doctors
∃ P (Occupation (P, Lawyer) ∧ ∀ c (Customer (c, P) ⇒ Occupation (c,



Code

```
knowledge_base = {
  "facts": {
    "American(Robert)",
    "Enemy(A, America)",
    "Owns(A, T1)",
    "Missile(T1)",
  },
  "rules": [
    {"if": ["Missile(x)"], "then": ["Weapon(x)"]},
    {"if": ["Enemy(x, America)"], "then": ["Hostile(x)"]},
```

```
        {"if": ["Missile(x)", "Owns(A, x)"], "then": ["Sells(Robert, x, A)"]},
        {
            "if": ["American(p)", "Weapon(q)", "Sells(p, q, r)", "Hostile(r)"],
            "then": ["Criminal(p)"],
        },
    ],
}

def forward_chaining(kb):
    facts = kb["facts"].copy()
    rules = kb["rules"]
    inferred = set()

    while True:
        new_inferences = set()

        for rule in rules:
            if_conditions = rule["if"]
            then_conditions = rule["then"]
            substitutions = {}
            all_conditions_met = True

            for condition in if_conditions:
                predicate, args = condition.split("(")
                args = args[:-1].split(",")
                matched = False

                for fact in facts:
```

```
            fact_predicate, fact_args = fact.split("(")
            fact_args = fact_args[:-1].split(",")

            if predicate == fact_predicate and len(args) == len(fact_args):
              temp_subs = {}
              for var, val in zip(args, fact_args):
                if var.islower():
                  if var in temp_subs and temp_subs[var] != val:
                    break
                  temp_subs[var] = val
                elif var != val:
                  break
              else:
                matched = True
                substitutions.update(temp_subs)
                break

          if not matched:
            all_conditions_met = False
            break

        if all_conditions_met:
          for condition in then_conditions:
            predicate, args = condition.split("(")
            args = args[:-1].split(",")
            new_fact = predicate + "(" + ",".join(substitutions.get(arg, arg) for arg in args) +
")"
            new_inferences.add(new_fact)
```

```
        if new_inferences - inferred:
            inferred.update(new_inferences)
            facts.update(new_inferences)
        else:
            break

    return inferred

result = forward_chaining(knowledge_base)

if "Criminal(Robert)" in result:
    print("Proved: Robert is a criminal.")
else:
    print("Could not prove that Robert is a criminal.")
```

OUTPUT: