ARTYOURTIME

function

1.(a) ~~predicate~~ S(a) means successor of a

function employ_n means employ_number of a

predicate prefer(X, Y) means X is prefer Y

(1) employ_n(kevin) = S(1)

(2) $\exists X$(prefer(X, VSCode) $\wedge$ (S(employ_n(X)) = employ_n(kevin)

$\vee$ S(S(employ_n(X))) = employ_n(kevin) $\vee$ S(S(S(employ_n(X))))

= employ_n(kevin))

(3) $\neg$(employ_n(kamal) = S(S(S(1)))) $\wedge$ $\neg$(prefer(kamal, Notepad)

$\wedge$ $\exists Y$($\neg$(employ_n(Y) = S(S(S(1)))) $\wedge$ prefer(Y, Notepad))

(4) $\exists Y$(prefer(Y, Notepad) $\wedge$ S(employ_n(Y)) = employ_n(karen)

(5) employ_n(kevin) $\vee$ ($\exists W$(employ_n(W) = S(S(1)) $\wedge$

prefer(W, Emacs)))

(b) knowledge base in CNF:

employ_n(kevin) = S(1)

prefer(x, VSCode)

S(employ_n(x)) = employ_n(kevin) $\vee$ S(S(employ_n(x))) = employ_n(kevin)

- $\vee$ S(S(S(employ_n(x)))) = employ_n(kevin)

$\neg$(employ_n(kamal) = S(S(S(1))))

$\neg$(prefer(kamal, Notepad)

prefer(y, Notepad)

$\neg$(employ_n(y) = S(S(S(1))))

S(employ_n(y)) = employ_n(karen)

employ_n(kevin) = S(S(1)) $\vee$ prefer(z, Emacs)

employ_n(kevin) = S(S(1)) $\vee$ employ_n(z) = S(S(1))

page 2 →

(b) applying resolution:
result: Kevin is the one who prefers Notepad

2.
```
and(atom(t), atom(t)).
and(X, Y) :- X, Y.

or(atom(t), atom(t)).
or(atom(t), atom(f)).
or(atom(f), atom(t)).
or(X, Y) :- X, Y.
or(X, Y) :- X, not(Y).
or(X, Y) :- not(X), Y.

implies(atom(t), atom(t)).
implies(atom(f), atom(t)).
implies(atom(f), atom(f)).
implies(X, Y) :- X, Y.
implies(X, Y) :- not(X), Y.
implies(X, Y) :- not(X), not(Y).

not(atom(f)).
not(not(X)) :- X.
not(and(X,Y)) :- ~~not(X),not(Y)~~ or(not(X), not(Y)).
not(or(X,Y)) :- and(not(X), not(Y)).
not(implies(X,Y)) :- and(X, not(Y)).
sat(X) :- X.
```

3.

Term is ***variables*** or ***functions***. Only expressions which can be obtained by finitely many applications of variables and functions are terms.

Predicate is a symbol which represents a ***property*** or a ***relation*** which can be either ***true*** or ***false***

Literal is a ***predicate*** or a ***negation of predicate***.

Conjunction is a ***logic operator, connectives*** that connects two literals or terms. It is close to "and" in English representation. The **and** of a set of operands is true if and only if all of its operands are true.

Disjunction is a ***logic operator, connectives*** that connects two literals or terms. It is close to "or" in English representation. The **or** of a set of operands is true if one of its operands are true.

Clause is a ***propositional formula*** formed from a finite collection of ***literals*** and ***logical connectives.***

Conjunctive normal form (CNF) is a ***conjunction*** of ***one or more clauses***, where a clause is a disjunction of literals. It is a product of SUMs or AND of ORs. A logical ***formula in CNF*** can be used to perform ***first-order resolution.***

Substitution is a total ***mapping*** $\sigma: V \rightarrow T$ **from variables to terms**

Unifier is the ***set of substitution*** that make ***unification successful***. A **unification** is a process of making ***two different logical atomic expression identical*** by finding a ***substitution.***

Resolution Rule is the ***resolving of two clauses*** if they contain ***complementary literals***, which are assumed to be standardized apart so that they share no variables.

First step of resolution is to convert facts into FOL. Turning terms into predicates, then literals. After that use logic connectives to form a FOL formula.

Second step is to turn that FOL formula into ***CNF*** by pushing all the negations (¬) down to the ***literals***, removing all of the ***existential quantifiers***, removing all of the ***universal quantifiers***, moving all of the ***conjunctions*** to the outside of the formula, and removing all of the true/false constants.

The final step is draw resolution graph which is a process of ***unification.***