JK Lakshmipat University

Near Mahindra SEZ, P.O. 302 026, Ajmer Road, Mahapura,
Rajasthan 302026

Project Report

Design and Analysis of Algorithms

Course Code: CS1105

Submitted to:

Dr. S Taruna

Submitted by:

Pravesh Bisaria

Viral Natani

Problem:

Consider the following two statements

- All humans are mortal.
- Socrates is a human.

From these we can deduce that

- Socrates is mortal.

The logical statements above may be expressed as follows in the logical formalism called first-order predicate logic.

For all x: human(x) --> mortal(x).

human(Socrates)

mortal(Socrates)

The first statement has the following meaning: for every x, if x is a human, then x is mortal. A proof is found by applying one or more logical inference rules. For example, the proof above is found by the inference rule called modus ponens: given the following two statements

P → Q, P.

we may conclude

Q.

Problem Statement:

The goal of present project is an implementation of a program for automatic theorem proving. The program should be able to read a set of logical sentences, covert these to an internal form, called clause form, and prove that one of the statements logically follows from the other statements.

Principles:

Resolution principle:

The resolution procedure is a simple iterative process: at each step, two clauses, called the parent clauses, are compared(resolved), yielding a new clause that has been inferred from them. The new clause represents ways that the two parent clauses interact with each other. Suppose that there are two clauses in the system:

winter **V** *summer*

-winter **V** *cold*

Now we observe that precisely one of winter and -winter will be true at any point. If winter is true then cold must be true to guarantee the truth of the second clause. Id -winter is true, then summer must be true to guarantee the truth of first clause. Thus, we see that from these two clauses we can deduce:

summer **V** *cold*

Resolution operates by taking two clauses that each contain the same literal, in this example, winter. The literal must occur in positive form in on e clause and in negative form in other. The *resolvent* is obtained by combining all of the literals of the two parent clauses except the ones that cancel.

Clausal Form Rules:

there is a simple procedure for converting an arbitrary set of Propositional Logic sentences to an equivalent set of clauses. The conversion rules are summarized below and should be applied in order.

1. Implications (I):

$$\begin{array}{lcl} \varphi \Rightarrow \psi & \rightarrow & \neg\varphi \vee \psi \\ \varphi \Leftarrow \psi & \rightarrow & \varphi \vee \neg\psi \\ \varphi \Leftrightarrow \psi & \rightarrow & (\neg\varphi \vee \psi) \wedge (\varphi \vee \neg\psi) \end{array}$$

2. Negations (N):

$$\begin{array}{lcl} \neg\neg\varphi & \rightarrow & \varphi \\ \neg(\varphi \wedge \psi) & \rightarrow & \neg\varphi \vee \neg\psi \\ \neg(\varphi \vee \psi) & \rightarrow & \neg\varphi \wedge \neg\psi \end{array}$$

3. Distribution (D):

$$\begin{array}{lcl} \varphi \vee (\psi \wedge \chi) & \rightarrow & (\varphi \vee \psi) \wedge (\varphi \vee \chi) \\ (\varphi \wedge \psi) \vee \chi & \rightarrow & (\varphi \vee \chi) \wedge (\psi \vee \chi) \\ \varphi \vee (\varphi_1 \vee \ldots \vee \varphi_n) & \rightarrow & \varphi \vee \varphi_1 \vee \ldots \vee \varphi_n \\ (\varphi_1 \vee \ldots \vee \varphi_n) \vee \varphi & \rightarrow & \varphi_1 \vee \ldots \vee \varphi_n \vee \varphi \\ \varphi \wedge (\varphi_1 \wedge \ldots \wedge \varphi_n) & \rightarrow & \varphi \wedge \varphi_1 \wedge \ldots \wedge \varphi_n \\ (\varphi_1 \wedge \ldots \wedge \varphi_n) \wedge \varphi & \rightarrow & \varphi_1 \wedge \ldots \wedge \varphi_n \wedge \varphi \end{array}$$

4. Operators (O):

$$\begin{array}{lcl} \varphi_1 \vee \ldots \vee \varphi & \rightarrow & \{\varphi_1, \ldots, \varphi_n\} \\ \varphi_1 \wedge \ldots \wedge \varphi_n & \rightarrow & \{\varphi_1\}, \ldots, \{\varphi_n\} \end{array}$$

Unification Principle

In predicate logic, matching process of two literals is complicated, since bindings of variables must be considered.

In Unification algorithm each literal is represented as a list, where first element is the name of a predicate and the remaining elements are arguments. The argument may be a single element (atom) or may be another list.

i) Different constants, functions or predicates cannot match, whereas identical ones can.

ii) A variable can match another variable, any constant or a function or predicate expression, subject to the condition that the function or [predicate expression must not contain any instance of the variable being matched (otherwise it will lead to infinite recursion).

iii) The substitution must be consistent. Substituting y for x now and then z for x later is inconsistent. (a substitution y for x written as y/x)

Unification Algorithm:

Algorithm: Unify (L1,L2)

I. If LI or 12 are both variables or constants, then:
   a. If LI and L2 are identical, then return NIL.
   b. Else if LI is a variable, then if L1 occurs in L2 then return (FAIL), else return (L2/L1),
   c. Else if L2 h a variable then if L3 occurs m L\ than return (FAIL), else return (L1/L2)*
   d. Else return [FAIL],

II. If the initial predicate symbols in LI and L2 are not identical, then return (FAIL).

III. If LI and L2 have a different number of arguments, then return {FAIL J,

IV. Set SUBST to NIL, (At the end of this procedure, SUBST will contain all the substitutions used to unify LI and LI.)

V. For i = 1 to number of arguments in LI:

a. Call Unify with the ith argument of LI and the ith argument of L2, putting result in S.
b. If S contains FAIL then return {FAIL}
c. If S is not equal to NIL then:
    i. Apply S to the remainder of both LI and L2.
    ii. SUBST: - APPEND(S, SUBST).

VI. Return SUBST

Algorithm :

1, Convert all the statements of F to clause form.

2, Negate P and convert the result to clause form. Add it to the set of clauses obtained in step 1.

3, Repeat until either a contradiction is found, no progress can be made, or a pre-determined amount of effort has been expended.

(a) Select two clauses. Call these the parent clauses.,

(b) Resolve them together. The resolvent will be the disjunction of all the literals of both parent clauses with appropriate substitutions performed and with the following exception: If there is one pair of literals T1 and ~T2 such that one of the parent clauses contains T2 and the other contains TI and if T1 and T2 are unifiable, then neither T1 nor T2 should appear in the resolvent. We call T1 and T2 Complementary literals. Use the substitution produced by the unification to create the resolvent. If there is more than one pair of complementary literals, only one pair should be omitted from the resolvent.

(c) If the resolvent is the empty clause, then a contradiction has been found- If it is not, then add it to the set of clauses available to the procedure.

Pseudo Code:

LIST: List containing clausal form of statements.

CON: Clausal Form of conclusion.

KMPAlgo()--- Find the matching literal

StringSearch – To find string in list

Resolution () --- Unify the literals and return the unmatched literal

```
TheoremProver (CON,LIST)
    R = -CON
    Loop until LIST != NULL:
        A = StringSearch(R,LIST)
        R = Resolution(A,R)
            If R == NULL
                break

StringSearch(R,LIST):
            Loop until I < len(LIST):
                bool = KMPAlgo(R,LIST[I])
```

```
IF bool == TRUE :
    A = LIST(I)
    break
```

Complexity:

```
TheoremProver (CON,LIST)
    R = -CON
    Loop until LIST != NULL:
        A = StringSearch(R,LIST)    → n * m
        R = Resolution(A,R)
            If R == NULL
                break
```

m = length of each string

With each iteration length of list would decrease :

n*m + (n-1)*m + (n-2)*m + …… + 1*m

= **[n(n+1)*m] / 2**

: $\mathbf{O(n^3)}$

Example.

Consider following statements

(a) Marcus was a man.

(b) Marcus was a Roman.

(c) All men are people.

(d) Caesar was a ruler.

(e) All Romans were either loyal to Caesar or hated him (or both). (f) Everyone is loyal to someone.

(g) People only try to assassinate rulers they are not loyal to.

(h) Marcus tried to assassinate Caesar.

Conversion to First order Logic

A. Marcus was a man.
   man(marcus)

B. Marcus was a Roman.
   roman(marcus)

C. All men are people.
   $\forall$X. man(X) $\rightarrow$ person(X)

D. Caesar was a ruler.
   ruler(caesar)

E. All Romans were either loyal to Caesar or hated him (or both).

$\forall X.\ roman(x) \rightarrow loyal(X,caesar) \vee hate(X,caesar)$

F. Everyone is loyal to someone.

$\forall X \exists Y.\ loyal(X,Y)$

G. People only try to assassinate rulers they are not loyal to .

$\forall X \forall Y.\ person(X) \wedge ruler(Y)\ tryassasin(X,Y) \rightarrow \neg loyal(X,Y)$

H. Marcus tried to assassinate Caesar.

tryassasin(marcus,caesar)

Conversion to Clausal Form

1. man(marcus)
2. roman(marcus)
3. (¬man(X), person(X))
4. ruler(caesar)
5. (¬roman(X), loyal(X,caesar), hate(X,caesar))
6. (loyal(X,f(X))
7. (¬person(X), ¬ruler(Y), ¬tryassasin(X,Y), ¬loyal(X,Y))
8. tryassasin(marcus,caesar)

Diagramtic Representation:

Prove: *hate(marcus, caesar)*