TRƯỜNG ĐẠI HỌC GIAO THÔNG VẬN TẢI

KHOA CÔNG NGHỆ THÔNG TIN

BỘ MÔN MẠNG VÀ CÁC HỆ THỐNG THÔNG TIN

**BÁO CÁO XÂY DỰNG CHƯƠNG TRÌNH**

**HỌC PHẦN TRÍ TUỆ NHÂN TẠO**

1. Họ và tên: Ngô Trung Hiếu

2. Mã sinh viên: 191213136

3. Lớp: CNTT4-K60

4. Nội dung báo cáo

4.1. Thuật toán chứng minh logic mệnh đề

- Mô tả thuật toán:

+ Chứng minh logic mệnh đề bằng phương pháp bác bỏ.

+ Giả sử chúng ta có một tập công thức g = {G1,…, Gm} ta cần chứng minh công thức H là hệ quả logic của g.

+ Tức là ta chứng minh tập g’ = (G1, …, Gm, !H) là không thoả được. g’ sẽ không thoả được nếu từ g’ ta suy ra hai mệnh đề đối lập nhau.

**Bài toán chứng minh**

Cho giả thiết: Là các công thức được cho là đúng, gọi: Tiên đề

Yêu cầu chứng minh: Là công thức, chứng minh đúng, gọi: Định lý

**Phép chứng minh**

Xuất phát từ giả thiết, áp dụng các phép suy diễn để đi đến kết luận đúng

**VD:**

*Cho*

*Q* ∧ *S* ⇒ *G* ∧ *H* (1) đúng

*P* ⇒ *Q* (2) đúng

*R* ⇒ *S* (3) đúng

*P* (4) đúng

*R* (5) đúng

Yêu cầu chứng minh G đúng

+ Chứng minh bác bỏ:

Giả sử G sai, tức là !G

1) Chuẩn hóa công thức

*Q* ∧ *S* ⇒ *G* ∧ *H*

*!(Q^R) v (G^H)*

(!Qv!R) v (G^H)

((!Qv!R) v G) ^ ((!Qv!R) v H)

!Q v !R v G đúng (1)

!Q v !R v H đúng (2)

!P v Q đúng (3)

!R v S đúng (4)

P đúng (5)

R đúng (6)

CM: thêm !G (7) vào tập GT

// LQG: Luật phân giải

Áp dụng LPG cho 7 và 1 ta có !Q v !R (8)

Áp dụng LPG cho 8 và 3 ta có !P v !R (9)

Áp dụng LPG cho 9 và 5 ta có !R (10)

Áp dụng LPG cho 10 và 6 ta có câu rỗng (mâu thuẫn) => dừng suy ra điều phải chứng minh

Mã giả:

**Input:**

Tập G = {G1, G2, … Gn} các công thức (các tiên đề);

H – công thức cần chứng minh;

**Begin**

THOA = !H;

while (THOA != Ø)

begin

u = get(THOA); *// Lấy 1 công thức từ THOA*

for với mỗi công thức v thuộc G do

if res(u, v) = Ø then exit; *// thoát chương trình*

else THOA = THOA  res(u, v);

end;

if res(u, v) = Ø then thông báo “H là hệ quả logic của G”

else thông báo “H không là hệ quả logic của G”;

**end;**

- Kết quả cài đặt (mô tả chương trình hoặc mã chương trình):

+ Clause (class mô tả đối tượng là công thức):

**public class** Clause {  
 **private boolean state**;  
 **private** String **value**;  
  
 *// số thứ tự của công thức* **private int serial** = 0;  
  
 **public** Clause() {  
 }  
  
 **public** Clause(String value, **boolean** state, **int** serial) {  
 **this**.**state** = state;  
 **this**.**value** = value;  
 **this**.**serial** = serial;  
 }  
  
 **public** String getValue() {  
 **return value**;  
 }  
  
 **public void** setValue(String value) {  
 **this**.**value** = value;  
 }  
  
 **public boolean** getState() {  
 **return state**;  
 }  
  
 **public void** setState(**boolean** state) {  
 **this**.**state** = state;  
 }  
  
 **public int** getSerial() {  
 **return serial**;  
 }  
  
 **public void** setSerial(**int** serial) {  
 **this**.**serial** = serial;  
 }  
  
 @Override  
 **public** String toString() {  
 **return "Clause{"** +  
 **"state="** + **state** +  
 **", value='"** + **value** + **'\''** +  
 **", serial="** + **serial** +  
 **'}'**;  
 }  
}

+ Main:

**public class** Main {  
 **private static** Scanner *sc*;  
 **private static int** *size*;  
  
 *// đánh dấu số thứ tự công thức* **private static int** *flag*;  
  
 *// truy vết* **private static final** List<Trace> ***traces*** = **new** ArrayList<>();  
  
 *// các công thức* **private static final** List<Clause> ***G*** = **new** ArrayList<>();  
  
 **private static final** Queue<Set<Clause>> ***THOA*** = **new** ArrayDeque<>();  
  
 **private static final** List<Clause> ***visited*** = **new** ArrayList<>();  
  
 **private static boolean** *checkVisited* = **false**;  
  
 **public static void** main(String[] args) {  
 Main main = **new** Main();  
  
 main.input();  
 String clause = *sc*.nextLine();  
 System.***out***.println(main.solve(clause));  
 }  
  
 **public void** input() {  
 **try** {  
 System.*setIn*(**new** FileInputStream(**"src/com/company/queue/premises.txt"**));  
 } **catch** (FileNotFoundException e) {  
 *// TODO Auto-generated catch block* e.printStackTrace();  
 }  
  
 *sc* = **new** Scanner(System.***in***);  
  
 *size* = *sc*.nextInt();  
  
 *// số thứ tự* **int** serial = 0;  
  
 **for** (**int** i = 0; i < *size* + 1; i++) {  
 String input = *sc*.nextLine();  
 String[] parts = input.split(**"\\s+"**);  
 **for** (String part : parts) {  
 **if** (Objects.*equals*(part, **""**)) {  
 **break**;  
 }  
 Clause clause = **new** Clause();  
 **if** (part.indexOf(**'!'**) != -1) {  
 clause.setState(**false**);  
 } **else** {  
 clause.setState(**true**);  
 }  
  
 clause.setValue(part.replace(**"!"**, **""**));  
 clause.setSerial(serial);  
 ***G***.add(clause);  
 }  
 serial++;  
 }  
  
  
 }  
  
 *// trả về các số thứ tự của công thức cần xét* **public static** List<Integer> checkSerial(Set<Clause> u, Set<Clause> v) {  
 List<Integer> res = **new** ArrayList<>();  
 **for** (Clause clause : u) {  
 **for** (Clause value : v) {  
 **if** (clause.getValue().equals(value.getValue())  
 && clause.getState() != value.getState()) {  
 res.add(value.getSerial());  
 }  
 }  
 }  
 **return** res;  
 }  
  
 **public** String solve(String clause) {  
  
 Set<Clause> arrClause = **new** HashSet<>();  
 arrClause.add(**new** Clause(clause, **false**, *size* + 1));  
 ***THOA***.add(arrClause);  
 *flag* = *size* + 2;  
 **boolean** checkWhile = **true**;  
  
 *// lưu kết quả của giải thức* Set<Clause> resClause = **null**;  
 System.***out***.printf(**"%-20s%-20s%-20s%s\n"**, **"u"**, **"v"**, **"res(u,v)"**, **"THOA"**);  
 **while** (***THOA*** != **null** && checkWhile) {  
 Set<Clause> u = *get*();  
 **if** (u == **null**) **break**;  
 Set<Clause> v = **null**;  
 **for** (Clause g : ***G***) {  
 **for** (Clause uc : u) {  
 *checkVisited* = **false**;  
 v = **new** HashSet<>();  
 *// so sánh giá trị tiên đề trong THOA và các tiên đề G* **if** (Objects.*equals*(uc.getValue(), g.getValue())  
 && uc.getState() != g.getState()) {  
  
 *// tìm tất cả tiên đề có serial giống r push vào v* **for** (Clause value : ***G***) {  
 **if** (value.getSerial() == g.getSerial()) {  
 v.add(value);  
 }  
 }  
 *output*(u, 0);  
 *output*(v, 0);  
  
 resClause = *res*(u, v);  
 *output*(resClause, 0);  
  
 **if** (resClause.isEmpty()) {  
 **for** (Set<Clause> clauses : ***THOA***) {  
 *output*(clauses, 1);  
 }  
 trace(u.stream().findFirst().orElseThrow().getSerial());  
 checkWhile = **false**;  
 **break**;  
 } **else** {  
 **if** (!*checkVisited*) {  
 ***THOA***.add(resClause);  
 }  
  
 }  
 **for** (Set<Clause> clauses : ***THOA***) {  
 *output*(clauses, 1);  
 }  
 System.***out***.println();  
 }  
  
 }  
 }  
 System.***out***.println();  
 }  
 **if** (resClause != **null**) {  
 **return "\n"** + clause + **" là hệ quả logic của G"**;  
 }  
 **return "\n"** + clause + **" không là hệ quả logic của G"**;  
 }  
  
 **public static void** visit(Set<Clause> resClause) {  
 **int** count = 0;  
 **for** (**int** i = *size* + 1; i <= *flag* - 1; i++) {  
 **int** countS = 0;  
 count = 0;  
 List<Clause> temp = **new** ArrayList<>();  
 **for** (Clause s : ***visited***) {  
 **if** (s.getSerial() == i) {  
 countS++;  
 temp.add(s);  
 }  
 }  
 **int** serial = 0;  
 **if** (countS == resClause.size()) {  
 **for** (Clause t : resClause) {  
 **for** (Clause s : temp) {  
 **if** (t.getValue().equals(s.getValue()) && *flag* != 8) {  
 count++;  
 serial = s.getSerial();  
 }  
 }  
 }  
 }  
  
 **if** (count == resClause.size()) {  
 **for** (Clause s : resClause) {  
 s.setSerial(serial);  
 }  
 *flag*--;  
 *checkVisited* = **true**;  
 }  
 }  
  
 **if** (***visited***.isEmpty()) {  
 ***visited***.addAll(resClause);  
 } **else** {  
 List<Clause> visit = **new** ArrayList<>();  
 **for** (Clause clause : resClause) {  
 **for** (Clause resThoa : ***visited***) {  
 **if** (clause.getValue().equals(resThoa.getValue()) && clause.getSerial() == resThoa.getSerial()) {  
 visit.add(clause);  
 }  
 }  
 }  
  
 **if** (visit.isEmpty())  
 ***visited***.addAll(resClause);  
 }  
  
  
 }  
  
 **public static void** output(Set<Clause> res, **int** THOA) {  
 StringBuilder result = **new** StringBuilder();  
 **int** serial = 0;  
 **for** (Clause clause : res) {  
 **if** (!clause.getState()) {  
 result.append(**"!"**).append(clause.getValue()).append(**" "**);  
 } **else** result.append(clause.getValue()).append(**" "**);  
  
 serial = clause.getSerial();  
 }  
 result.append(**"("**).append(serial).append(**")"**);  
  
 **if** (THOA == 0) {  
 System.***out***.printf(**"%-20s"**, result);  
 } **else** {  
 System.***out***.printf(**"%s, "**, result);  
 }  
 }  
  
 *// lấy từ THOA* **public static** Set<Clause> get() {  
 **if** (***THOA***.isEmpty()) {  
 **return null**;  
 }  
  
 **return *THOA***.remove();  
 }  
  
 *// giải thức của 2 câu u, v* **public static** Set<Clause> res(Set<Clause> u, Set<Clause> v) {  
 **if** (u == **null** || v == **null**) {  
 **return null**;  
 }  
  
 List<Integer> serials = *checkSerial*(u, v);  
 Set<Clause> newClause = **new** HashSet<>();  
 Set<String> check = **new** HashSet<>();  
  
 *// lấy giá trị trùng của u và v* **for** (Integer serial : serials) {  
 **for** (Clause value : v) {  
 **if** (value.getSerial() == serial) {  
 **for** (Clause clause : u) {  
 **if** (Objects.*equals*(value.getValue(), clause.getValue())  
 && value.getState() != clause.getState()) {  
 check.add(value.getValue());  
 }  
 }  
 }  
 }  
 }  
  
 Trace trace = **new** Trace();  
 trace.setU(u.stream().findFirst().orElseThrow(**null**).getSerial());  
 trace.setV(v.stream().findFirst().orElseThrow(**null**).getSerial());  
  
 *addRes*(u, check, newClause);  
 *addRes*(v, check, newClause);  
  
 **if** (!newClause.isEmpty()) {  
 *visit*(newClause);  
 *flag*++;  
  
  
 trace.setRes(newClause.stream().findFirst().orElseThrow().getSerial());  
 }  
  
  
 ***traces***.add(trace);  
 **return** newClause;  
 }  
  
 *// tính giá trị của giải thức* **public static void** addRes(Set<Clause> clauses, Set<String> check, Set<Clause> newClause) {  
  
 **for** (String value : check) {  
 **for** (Clause clause : clauses) {  
 Clause clause1 = **new** Clause();  
  
 **if** (!Objects.*equals*(clause.getValue(), value)) {  
 clause1.setValue(clause.getValue());  
 clause1.setState(clause.getState());  
  
 **if** (*checkInThoa*(clause1) && *checkInNewClause*(newClause, clause1)) {  
 clause1.setSerial(*flag*);  
 newClause.add(clause1);  
 }  
 }  
 }  
 }  
  
 }  
  
 **public static boolean** checkInThoa(Clause clause) {  
 **if** (***THOA***.isEmpty()) **return true**;  
 **for** (Set<Clause> element : ***THOA***) {  
 **for** (Clause clauseSet : element) {  
 **if** (clause.equals(clauseSet)) {  
 **return false**;  
 }  
 **if** (clause.getValue().equals(clauseSet.getValue())) {  
 **return true**;  
 }  
 }  
 }  
 **return true**;  
 }  
  
 **public static boolean** checkInNewClause(Set<Clause> newClause, Clause clause) {  
 **if** (newClause.isEmpty()) **return true**;  
 **for** (Clause element : newClause) {  
 **if** (element.getValue().equals(clause.getValue())) {  
 **return false**;  
 }  
 }  
 **return true**;  
 }  
  
 **public void** trace(**int** u) {  
 System.***out***.printf(**"\nMâu thuẫn <- (%d, %d) <- "**, ***traces***.get(***traces***.size() - 1).getU(), ***traces***.get(***traces***.size() - 1).getV());  
 **for** (**int** i = ***traces***.size() - 1; i >= 0; i--) {  
 **if** (***traces***.get(i).getRes() == u) {  
 System.***out***.printf(**"(%d, %d) <- "**, ***traces***.get(i).getU(), ***traces***.get(i).getV());  
 u = ***traces***.get(i).getU();  
 }  
 }  
 }  
  
}

- Kết quả thử nghiệm:

+ Input:

6  
!Q !R G  
!Q !R H  
!P Q  
!R S  
P  
R  
G

+ Output:

