```
> restart;
> TriSim := table():
> TriSim[init] := proc()
  global R_RG, R_KT, HDT, SOL, used;
  local read_rules;

            #-------Hàm c file lut-------------------------------
  --------
            read_rules := proc(filename)
            local rules, fd, line, s;
                      rules := [];
                      fd := fopen(cat(url, "/DATA/",filename),
  READ, TEXT);
                      line := readline(fd);
                      while line <> 0 and SearchText("begin_rules",
  line) = 0  do
          line := readline(fd);
                      od;
                      line := readline(fd);
                      while line <> 0 and SearchText("end_rules",
  line) = 0  do
                                  rules := [op(rules), parse(line)];
            line := readline(fd);
       od;
       fclose(fd);
                      return rules;
            end: # read_rules

            #-------Thân hàm init--------------------------------
  ----------
            SOL := [];
            R_RG := read_rules("luatrutgon.txt");
            R_KT := read_rules("luatkhaitrien.txt");
            HDT := read_rules("hangdangthuc.txt");

  end: # init
> TriSim[Expr_Length] := proc(expr)
  # Hàm tìm chiu dài ca mt biu thc hàm
  local listexpr, i;
            if type(expr,`function`) and member(op(0,expr),{sin,
  cos, tan, cot}) then
```

```
                    return 1;
          fi;
          if type(expr,`^`) and type(op(2, expr), posint)then
                    return op(2,expr)*Expr_Length(op(1,expr));
          fi;
          if type(expr,`^`) and op(2,expr) = -1 then
                    return Expr_Length(op(1,expr));
          fi;
          if type(expr,`^`) and type(op(2, expr), fraction) then
                    return Expr_Length(op(1,expr));
          fi;
          if type(expr,`+`) then
                    listexpr := [op(expr)];
                    return convert(map(s->Expr_Length(s),
listexpr),`+`);
          fi;
          if type(expr,`*`) then
                    # Không có mu
                    if denom(expr) = 1 then
                              listexpr := [op(expr)];
                              return convert(map(s->Expr_Length
(s), listexpr),`+`);
                    # Có mu
                    else
                              return Expr_Length(numer(expr)) +
Expr_Length(denom(expr));
                    fi;
          fi;
          if type(expr,`function`) and op(0,expr) = sqrt then
                    return Expr_Length(op(expr));
          fi;
          if type(expr,`=`) then
                    return Expr_Length(lhs(expr))  + Expr_Length
(rhs(expr));
          fi;

          return 0;
end: # Expr_Length
> TriSim[Is_Sub_Coeff] := proc(subexpr, s)
local setsub, sets, i, num, coms;
          setsub := {op(subexpr)};
          sets := {op(s)};
```

```
            for i in sets do
                        if type(i, `*`) and type(op(1, i), numeric)
  and denom(i) = 1 then
                                    num := op(1,i);
                                    break;
                        fi;
            od;
            sets := map(s->s/num, sets);
            sets := convert([op(sets)], `+`);
            if subexpr = sets then
                        return true, num;
            fi;
            return false, 0;
  end:
> TriSim[Is_SubExpr_Add] := proc(subexpr, expr)
  local setexpr, comexpr, s, temp, flag, num;
            # TH subexpr có dng A+B
            if type(subexpr, `+`) and nops({op(subexpr)}) = 2 then
                        if type(expr, `+`) then
                                    setexpr := {op(expr)};
                                    comexpr := combinat[permute]
  (setexpr,2);
                                    comexpr := map(s->convert(s,`+`),
  comexpr);
                                    for s in comexpr do
                                                if s = subexpr then
                                                            return true;
                                                else
                                                            flag:=
  Is_Sub_Coeff(subexpr, s);
                                                            if flag[1] =
  true then

  return true;
                                                            fi;
                                                fi;
                                    od;
                        fi;
            fi;
            return false;
  end:
```

```
> TriSim[Is_SubExpr] := proc(subexpr, expr)
  # Hàm kim tra subexpr có phi là biu thc con ca expr không?
  local temp;
            if has(expr, subexpr) then return true; fi;

            temp := applyrule(subexpr = _X, expr);
            if convert(temp,string) <> convert(expr, string) then
  return true; fi;

            if type(subexpr, `+`) and nops({op(subexpr)}) = 2 then
                      return Is_SubExpr_Add(subexpr, expr);
            fi;

            return false;
  end:# Is_SubExpr
> TriSim[Heu_HDT_Sim] := proc(Hdt, expr)
  # Hàm chn ra mt hdt t tp Hdt theo heuristic
  # Heuristic:
            # chn heu mà có th làm expr n gin hn
            # chon hdt có length(kt qu) là nh nht và length(gi
  thuyt) là dài nht
  local Heu_Hdt, hdt, maxlen, minlen;
            # Chn các hdt có length(gi thuyt) dài nht
            maxlen := max(map(t -> Expr_Length(rhs(t)), Hdt));
            Heu_Hdt := select(t -> is(Expr_Length(rhs(t)) =
  maxlen), Hdt);
            if nops(Heu_Hdt) = 1 then return op(Heu_Hdt);fi;

            # Chn tip hdt có length(kt lun) nh nht
            minlen := min(map(t->Expr_Length(lhs(t)),Heu_Hdt));
            Heu_Hdt := select(t -> is(Expr_Length(lhs(t)) =
  minlen), Heu_Hdt);

            return Heu_Hdt[1];
  end:# Heu_HDT_Sim
> TriSim[Find_HDT_Sim] := proc(expr)
  # Hàm tìm mmt hng ng thc có trong expr  rút gn
  local funcs, per, hdt, p, subhdt, Hdt, temp;
  global HDT;
            funcs := indets(expr, function);
            per := combinat[permute](funcs,2);
            Hdt := [];
```

```
            for hdt in HDT do
                                for p in per do
                                        subhdt := subs({_A = p
  [1], _B = p[2]}, hdt);
                                                if Is_SubExpr
  (rhs(subhdt),expr) then
                                                        temp
  := Apply(rhs(subhdt)=lhs(subhdt),expr);
                                                        if
  Expr_Length(temp) < Expr_Length(expr) then

  return rhs(subhdt)=lhs(subhdt);
                                                        fi;
                                                fi;
                                od;
            od;
            if Hdt = [] then  return FAIL;fi;
            return Heu_HDT_Sim(Hdt, expr);

  end: # Find_HDT_Sim
> TriSim[Find_Arc] := proc(expr)
  # Hàm tìm các i ca expr
  local lfun;
            lfun := indets(expr,function);
            lfun := map(t->op(t),lfun);
            return lfun;
  end:
> TriSim[Heu_Rule_Sim] := proc(Rrg, expr)
  # Hàm chn ra mt hdt t tp Hdt theo heuristic
  # Heuristic:
            # 1. u tiên chn rule có v phi là s
            # 2. # u tiên không sinh ra cung mi, tr trng hp
  length(expr) <= 2
            # 3. u tiên dùng các lut bin i thành tan, cot nu
  trong biu thc có sin, cos < tan, cot
            # 4, chon hdt có length(kt qu) là nh nht và length
  (gi thuyt) là dài nht

  local Heu_Rrg, rule, maxlen, minlen, temp, num, arce, Temp;

            # Th áp dng heuristic 1
            temp := select(t->is(type(rhs(t),`numeric`)), Rrg);
```

```
          if temp <> [] then return temp[1]; fi;

          # Th áp dng heuristic 2
          # Chn các lut không sinh cung mi
          temp := [];
          arce := Find_Arc(expr);
          for rule in Rrg do
                    if Find_Arc(rhs(rule)) subset arce then
                         temp := [op(temp), rule];
                    fi;
          od;

          # Nu là ln cui thì không cn chn
          if Expr_Length(expr) < 3  or temp = [] then
                    temp := Rrg;
          fi;

          Temp := temp;
          # Th áp dng heuristic 2
          num := Count_Func(expr, {sin, cos});
          if num <> 0  and num + 3 <  Count_Func(expr, {tan, cot}
) then
                    temp := [];
                    for rule in Temp do
                              if has(rhs(rule), tan) or has(rhs
(rule), cot) then
                                        temp := [op(temp), rule];
                              fi;
                    od;
          fi;
          if temp = [] then temp := Temp;fi;


          # Chn các lut có length(gi thuyt) dài nht
          maxlen := max(map(t -> Expr_Length(lhs(t)), temp));
          Heu_Rrg := select(t -> is(Expr_Length(lhs(t)) =
maxlen), temp);
          if nops(Heu_Rrg) = 1 then return op(Heu_Rrg);fi;
          # Chn tip lut có length(kt lun) nh nht
          minlen := min(map(t->Expr_Length(rhs(t)),Heu_Rrg));
          Heu_Rrg := select(t -> is(Expr_Length(rhs(t)) =
minlen), Heu_Rrg);
```

```
            return Heu_Rrg[1];
  end:# Heu_Rule_Sim
> TriSim[Get_Var_bien_so] := proc(expr)
  # Hàm ly các i dng x - s trong expr
  local vars, funcs, f, var;
            funcs := indets(expr, function);
            vars:={};
            for f in funcs do
                        if type(f, `function`) and member(op(0, f) ,
  {sin, cos, tan, cot}) then
                                    var := op(1,f);
                                    if type(var,`+`) and nops({op(var)}
  ) = 2 then
                                                vars := vars union {var};
                                    fi;
                        fi;
            od;
            return vars;
  end:
> TriSim[Find_Rule_Sim] := proc(expr)
  # Hàm tìm mt lut rút gn có th áp dng c lên expr
  # r gl áp dng c trên expr khi lhs(r) là biu thc con ca
  expr
  local vars, r, subr, varr, v, Rrg, per, p, varscong;
  global R_RG;
            vars := indets(expr, name) minus {Pi};
            # Có tham bin
            if nargs = 2 then vars := vars minus arg[2];fi;
            Rrg := [];
            for r in R_RG do
                        varr := indets(r, name);
                        # expr Có nhiu bin
                        if nops(vars) > 1 then
                                    # lut r có mt bin
                                    if nops(varr) = 1 then
                                                for v in vars do
                                                            subr := subs
  (varr[1] = v, r);
                                                            if Is_SubExpr
  (lhs(subr), expr) then
                                                                        Rrg
  := [op(Rrg), subr];
```

```
                                                fi;
                                        od;
                                # lut r có 2 bin
                                else
                                        per := combinat[permute]
(vars, 2);
                                        for p in per do
                                                subr := subs(
{varr[1] = p[1], varr[2] = p[2]}, r);
                                                if Is_SubExpr
(lhs(subr), expr) then
                                                        Rrg
:= [op(Rrg), subr];
                                                fi;
                                        od;
                                fi;

                        # expr Có mt bin
                        elif nops(vars) = 1 then
                                # lut r có mt bin
                                if nops(varr) = 1 then
                                        subr := subs(varr[1] =
vars[1], r);
                                        if Is_SubExpr(lhs(subr),
expr) then
                                                Rrg := [op
(Rrg), subr];
                                        fi;
                                #lut r có nhiu bin --> expr có
i dng (x - s). Ví d: (x - Pi/6)
                                else
                                        varscong :=
Get_Var_bien_so(expr);
                                        if varscong <> {} then
                                                for v in
varscong do
                                                        per
:= combinat[permute]({op(v)}, 2);
                                                        for p
in per do

subr := subs({varr[1] = p[1], varr[2] = p[2]}, r);
```

```
  if Is_SubExpr(lhs(subr), expr) then

            Rrg := [op(Rrg), subr];

  fi;
                                                                  od;
                                                          od;
                                                  fi;
                                          fi;
                                  fi;
            od;
            if Rrg = [] then return FAIL; fi;
            return Heu_Rule_Sim(Rrg, expr);
  end: # Find_Rule_Sim
> TriSim[Find_Common_2Poly] := proc(expr1, expr2, gc)
  # Hàm tìm phn t chung gia hai biu thc dng a thc expr1,
  expr2 t dng phân tích thành nhân t g
  local g, v1, v2, lg, temp;
            if type(gc, `^`) then lg := [gc];
            else lg := [op(gc)]; fi;
            temp :=[];
            for g in lg do
                      v1 := expr1/g; v2 := expr2/g;
                      if type(g,numeric) then
                                if g*v1 - expr1 = 0 and g*v2 -
  expr2 = 0 then
                                          temp := [op(temp), g];
                                fi;

                      elif Expr_Length(v1) < Expr_Length(expr1) and
  Expr_Length(v2) < Expr_Length(expr2) then
                                temp := [op(temp), g];
                      fi;
            od;
            return convert(temp,`*`);
  end:# Factor_2Expr

> TriSim[Find_Common_2Expr] := proc(expr1, expr2)
  # Hàm tìm phn t chung ca hai biu thc expr1, expr1 (c a
  thc và phân thc)
  local g, de, nu, temp ;
```

```
            g := factor(expr1 + expr2);
            if denom(g)  = 1 then
                        temp := Find_Common_2Poly(expr1, expr2, g);
                        if type(temp,numeric) then return FAIL;
                        else return temp; fi;
            else
                        nu := Find_Common_2Poly(expr1, expr2, numer(g));
                        de := Find_Common_2Poly(expr1, expr2, 1/denom
  (g));
                        temp :=  nu*de;
                        if type(temp,numeric) then return FAIL;
                        else return temp; fi;
            fi;
            return FAIL;
  end: #Find_Common_2Expr
> TriSim[Factor_2Expr] := proc(expr1, expr2)
  # Hàm t tha s chung ca biu thc expr1+expr2
  local cm, v1, v2;
            cm := Find_Common_2Expr(expr1, expr2);
            if cm = FAIL then return FAIL;fi;
            v1 := expr1 / cm;
            v2 := expr2/cm;
            return cm*(v1+v2);
  end: # Factor_2Expr
> TriSim[Has_Form] := proc(expr, form)
  # Hàm kim tra form có trong expr
  local vars, sf, v;
            vars := indets(expr, name) minus {Pi};
            if form = "sin^2+cos^2"  then
                        if nops(vars) = 1 then
                                    sf := subs(_x = vars[1],sin(_x)^2 +
  cos(_x)^2 );
                                    if Is_SubExpr(sf,expr) then return
  true;
                                    fi;
                        elif nops(vars) > 1 then
                                    for v in vars do
                                                sf := subs(_x = v,sin(_x)
  ^2 + cos(_x)^2 );
                                                if Is_SubExpr(sf,expr)
  then return true;
                                                fi;
```

```
                                od;
                        fi;
                fi;
                if form  = "tan=sin/cos" then
                        if nops(vars) = 1 then
                                sf := subs(_x = vars[1],tan(_x) =
sin(_x)/cos(_x) );
                                if Is_SubExpr(sf,expr) then return
true;fi;
                        elif nops(vars) > 1 then
                                for v in vars do
                                        sf := subs(_x = v,tan(_x)
= sin(_x)/cos(_x));
                                        if Is_SubExpr(sf,expr)
then return true;
                                        fi;
                                od;
                        fi;
                fi;
                if form  = "cot=cos/sin" then
                        if nops(vars) = 1 then
                                sf := subs(_x = vars[1],cot(_x) =
cos(_x)/sin(_x) );
                                if Is_SubExpr(sf,expr) then return
true;fi;
                        elif nops(vars) > 1 then
                                for v in vars do
                                        sf := subs(_x = v,cot(_x)
= cos(_x)/sin(_x));
                                        if Is_SubExpr(sf,expr)
then return true;
                                        fi;
                                od;
                        fi;
                fi;
                if form  = "tan^2=1/cos^2-1" then
                        if nops(vars) = 1 then
                                sf := subs(_x = vars[1],tan(_x)^2 =
1/cos(_x)^2 - 1 );
                                if Is_SubExpr(sf,expr) then return true;fi;
                        elif nops(vars) > 1 then
                                for v in vars do
```

```
                                        sf := subs(_x = v,tan(_x)
^2 = 1/cos(_x)^2 - 1 );
                                        if Is_SubExpr(sf,expr)
then return true;
                                        fi;
                                od;
                        fi;
                fi;
                if form  = "cot^2=1/sin^2-1" then
                        if nops(vars) = 1 then
                                sf := subs(_x = vars[1],cot(_x)^2 =
1/sin(_x)^2 - 1 );
                                if Is_SubExpr(sf,expr) then return
true;fi;
                        elif nops(vars) > 1 then
                                for v in vars do
                                        sf := subs(_x = v,cot(_x)
^2 = 1/sin(_x)^2 - 1 );
                                        if Is_SubExpr(sf,expr)
then return true;
                                        fi;
                                od;
                        fi;
                fi;
                return false;
end:# Has_Form
> TriSim[Count_Func] := proc(expr, func)
# Hàm m s lng hàm fun trong expr
# func là mt tp hp
local listexpr, i;
                if type(expr,`function`) and member(op(0,expr),func) then
                        return 1;
                fi;
                if type(expr,`^`) and type(op(2, expr), posint)then
                        return op(2,expr)*Count_Func(op(1,expr), func);
                fi;
                if type(expr,`+`) then
                        listexpr := [op(expr)];
                        return convert(map(s->Count_Func(s, func),
listexpr),`+`);
                fi;
                if type(expr,`*`) then
```

```
                    # Không có mu
                    if denom(expr) = 1 then
                              listexpr := [op(expr)];
                              return convert(map(s->Count_Func(s,
func), listexpr),`+`);
                    # Có mu
                    else
                              return Count_Func(numer(expr),
func) + Count_Func(denom(expr), func);
                    fi;
          fi;
          if type(expr,`function`) and op(0,expr) = sqrt then
                    return Count_Func(op(expr), func);
          fi;
          if type(expr,`=`) then
                    return Count_Func(lhs(expr), func)  +
Count_Func(rhs(expr), func);
          fi;

          return 0;
end: # Count_Func
> TriSim[Heu_Factor] := proc(lcme)
# Hàm chn ra mt cách t tha s chung theo heuristic
# Heuristic:
          # 1. có xut hin dng sin^2 + cos^2
          # 2. có phn t chung #--> Cha code tt cái này
          # 3. có hàm cot và tan ít nht
          # 4. có  dài nh nht

local l, hl, temp,  minlen;
          #  Ap dng heuristic 1
          hl := [];
          for l in lcme do
                    if Has_Form(l, "sin^2+cos^2") then
                              hl := [op(hl), l];
                    fi;
          od;
          if hl <> [] then return hl[1];fi;

          # Áp dng heuristic 3
          minlen := min(map(t->Count_Func(t, {tan,cot}),lcme));
          if minlen <> 0 then
```

```
                    return select(t->is(Count_Func(t, {tan,cot})
= minlen),lcme)[1];
          fi;

          # Ap dungj heuristic 4
          minlen := min(map(t->Expr_Length(t),lcme));
          return select(t->is(Expr_Length(t) = minlen),lcme)[1];

          # Ap dungj heuristic 2
          for l in lcme do
                    temp :=  Factor_Expr(l);
                    if temp <> FAIL then return l; fi;
          od;

end: # Heu_Factor
> TriSim[Factor_Expr] := proc(expr)
# Hàm t tha s chung trong expr
# Nu không c thì return expr
local lexpr, per1, per2, p1, p2, cmp1, cmp2, lcme, temp;
          lcme := [];
          if type(expr,`+`) then
                    lexpr := [op(expr)];
                    per1 := convert((map(t->convert(t,set),
combinat[permute](lexpr,2))),set);
                    for p1 in per1 do
                              per2 := convert(lexpr,set) minus p1;
                              per2 := convert((map(t->convert(t,
set),combinat[permute](per2,2))),set);
                              cmp1 := Factor_2Expr(op(p1));
                              if cmp1 <> FAIL then
                                        if per2 = {} then
                                                  temp := convert
(convert(convert(lexpr,set) minus p1,list),`+`);
                                                                  temp
:= cmp1 + temp;
                                                                  if
not member(temp, lcme) then

lcme := [op(lcme), temp];
                                                                  fi;
                                        fi;
                                        for p2 in per2 do
```

```
                                                        cmp2 :=
  Factor_2Expr(op(p2));
                                                        if cmp2 <> FAIL then
                                                                temp
  := convert(convert(convert(lexpr,set) minus (p1 union p2),list),
  `+`);
                                                                temp
  := cmp1 + cmp2 + temp;
                                                                if
  not member(temp, lcme) then

  lcme := [op(lcme), temp];
                                                                fi;
                                                        else
                                                                temp
  := convert(convert(convert(lexpr,set) minus p1,list),`+`);
                                                                temp
  := cmp1 + temp;
                                                                if
  not member(temp, lcme) then

  lcme := [op(lcme), temp];
                                                                fi;
                                                        fi;
                                                od;
                                        fi;
                        od;
                        if lcme = [] then return FAIL; fi;
                        if nops(lcme) = 1 then return lcme[1];
                        else
                                        return Heu_Factor(lcme);
                        fi;
                fi;
                return FAIL;
  end: #Find_Common
> TriSim[Apply] := proc(r, expr)
  # Hàm áp dng lut r lên expr
  local g, setexpr, comexpr, s, temp, flag, subexpr, reexpr;
                g := applyrule(r, expr);
                if g  - expr <> 0 then return g;fi;
                #----------------------------------------
                subexpr := lhs(r); reexpr := rhs(r);
```

```
            if type(lhs(r), `+`) and nops({op(lhs(r))}) = 2 then
                        setexpr := {op(expr)};
                        comexpr := combinat[permute](setexpr,2);
                        comexpr := map(s->convert(s,`+`), comexpr);
                        for s in comexpr do
                                    if s = subexpr then
                                                temp := setexpr minus({op(s)});
                                                temp := convert([op(temp)
  ], `+`) + reexpr;
                                                return temp;
                                    else
                                                flag:= Is_Sub_Coeff(subexpr, s);
                                                if flag[1] = true then
                                                            temp := setexpr
  minus({op(s)});
                                                            temp := convert
  ([op(temp)], `+`) + flag[2]*reexpr;
                                                            return temp;
                                                fi;
                                    fi;
                        od;
            fi;
            return expr;
  end: # Apply
> TriSim[Simplify_Simple] := proc(expr)
  # Hàm rút gn n gin
  local g, r, flag, temp;
  global SOL, used;
            used := [];
            g := expr;
            # Th dùng hng ng thc
            r := Find_HDT_Sim(g);
            while r <> FAIL and not member(r, used) do
                        temp := Apply(r,g);
                        SOL := [op(SOL), [g, r, temp]];
                        used := [op(used),r];
                        g := temp;
                        r := Find_HDT_Sim(g);

            od;
            # Th dùng lut rút gn
            r := Find_Rule_Sim(g);
```

```
            while r <> FAIL and not member(r, used) do
                      temp := Apply(r,g);
                      SOL := [op(SOL), [g, r, temp]];
                      used := [op(used),r];
                      g := temp;
                      r := Find_Rule_Sim(g);

            od;
            # Th t tha s chung
            r := Factor_Expr(g);
            if r = FAIL then return g;
            else
                      SOL := [op(SOL), [g, "t tha s chung", r]]
  ;
                      return Simplify_Simple(r);
            fi;
            return r;

  end:# Simplify_Simple
```

```
> TriSim[Simplify_Simple_Rad] := proc(expr,dk)
  # Hàm rút gn n gin cho cn thc
  local g, r, flag, temp, mu;
  global SOL, used;
            if not type(expr, `^`) or not type(op(2,expr),
  fraction) then return expr;fi;
            used := [];
            g := op(1,expr);
           mu := op(2,expr);
            # Th dùng hng ng thc
            r := Find_HDT_Sim(g);
            while r <> FAIL and not member(r, used) do
                      temp := Apply(r,g);
                      SOL := [op(SOL), [g^mu, r, temp^mu]];
                      used := [op(used),r];
                      g := temp;
                      r := Find_HDT_Sim(g);
            od;

            # Th dùng lut rút gn
            r := Find_Rule_Sim(g);
            while r <> FAIL and not member(r, used) do
```

```
                    temp := Apply(r,g);
                    SOL := [op(SOL), [g^mu, r, temp^mu]];
                    used := [op(used),r];
                    g := temp;
                    r := Find_HDT_Sim(g);

          od;

          # Th t tha s chung
          r := Factor_Expr(g);
          if r = FAIL then return g^mu;
          else
                    SOL := [op(SOL), [g^mu, "t tha s chung",
r^mu]];
                    return Simplify_Simple_Rad(r^mu, dk);
          fi;
          return r^mu;

end:# Simplify_Simple_Rad
> TriSim[Find_HDT_Exp] := proc(expr)
# Hàm tìm mt hng ng thc có trong expr  khai trin
local funcs, per, hdt, p, subhdt, Hdt;
global HDT;
          funcs := indets(expr, function);
          per := combinat[permute](funcs,2);
          Hdt := [];
          for hdt in HDT do
                    for p in per do
                              subhdt := subs({_A = p[1], _B = p
[2]}, hdt);
                              if Is_SubExpr(lhs(subhdt),expr) then
                                        Hdt := [op(Hdt), subhdt];
                              fi;
                    od;
          od;
          if Hdt = [] then  return FAIL;fi;
          return Hdt[1];

end: # Find_HDT_Exp
> TriSim[Heu_Rule_Exp] := proc(Rkt, expr)
# Hàm chn mt lut khai trin theo heuristic
# Heuristic:
```

```
          # 1. Nu biu thc có nhiu tan và cot hn thì uy tiên
các lut chuyn v tan, cot
          # 2. Nu có tan^2 = ... và cot^2 = ... thì uu tien s dng
          # 3. u tiên lut: tan = sin/cos, cot = cos/sin
local r, num, temp, rule, maxlen, minlen, Heu_Rkt;
          # Th áp dng heuristic 1
          #num := Count_Func(expr, {sin, cos});
          #temp := [];
          #if num <> 0  and num + 3 <  Count_Func(expr, {tan, cot}) then

          # for rule in Rkt do
          #  if has(rhs(rule), tan) or has(rhs(rule), cot) then
          #   temp := [op(temp), rule];
          #  fi;
          # od;
          #fi;
          #if temp <> [] then return temp[1];fi;


          # Th áp dng heuristic 3
          for r in Rkt do
                    if Has_Form(r,"tan=sin/cos") then return r; fi;
                    if Has_Form(r,"cot=cos/sin") then return r; fi;
          od;
          # Th áp dng heuristic 2
          #for r in Rkt do
          # if Has_Form(r,"tan^2=1/cos^2-1") then return r; fi;
          # if Has_Form(r,"cot^2=1/sin^2-1") then return r; fi;
          #od;

          varscong := Get_Var_bien_so(expr);
          if varscong <> {} then
                    # Chn các lut có length(gi thuyt) dài
nht
                    maxlen := max(map(t -> Expr_Length(lhs(t)), Rkt));
                    Heu_Rkt := select(t -> is(Expr_Length(lhs(t))
= maxlen), Rkt);
                    if nops(Heu_Rkt) = 1 then return op(Heu_Rkt);fi;

                    # Chn tip lut có length(kt lun) nh nht
                    minlen := min(map(t->Expr_Length(rhs(t)),
Heu_Rkt));
```

```
                    Heu_Rkt := select(t -> is(Expr_Length(rhs(t))
  = minlen), Heu_Rkt);
                    return Heu_Rkt[1];
          fi;

          return Rkt[1];
  end:# Heu_Rule_Exp
```

Warning, `varscong` is implicitly declared local to procedure `TriSim[Heu_Rule_Exp]`

```
> TriSim[Find_Rule_Exp] := proc(expr)
  # Hàm tìm mt lut khai trien có th áp dng c lên expr
  # r gl áp dng c trên expr khi lhs(r) là biu thc con ca
  expr
  local vars, r, subr, varr, v, Rkt, per, p, varscong;
  global R_KT;
          vars := indets(expr, name) minus {Pi};
          # Có tham bin
          if nargs = 2 then vars := vars minus arg[2];fi;
          Rkt := [];
          for r in R_KT do
                    varr := indets(r, name);
                    # expr Có nhiu bin
                    if nops(vars) > 1 then
                              # lut r có mt bin
                              if nops(varr) = 1 then
                                        for v in vars do
                                                  subr := subs
  (varr[1] = v, r);
                                                  if Is_SubExpr
  (lhs(subr), expr) then
                                                            Rkt
  := [op(Rkt), subr];
                                                  fi;
                                        od;
                              # lut r có 2 bin
                              else
                                        per := combinat[permute]
  (vars, 2);
                                        for p in per do
                                                  subr := subs(
  {varr[1] = p[1], varr[2] = p[2]}, r);
                                                  if Is_SubExpr
```

```
(lhs(subr), expr) then
                                                                Rkt
:= [op(Rkt), subr];
                                                        fi;
                                                od;
                                        fi;

                        # expr Có mt bin
                        elif nops(vars) = 1 then
                                # lut r có mt bin
                                if nops(varr) = 1 then
                                        subr := subs(varr[1] =
vars[1], r);

                                        if Is_SubExpr(lhs(subr),
expr) then
                                                Rkt := [op(Rkt), subr];
                                        fi;
                                #lut r có nhiu bin --> expr có
i dng (x - s). Ví d: (x - Pi/6)
                                else
                                        varscong :=
Get_Var_bien_so(expr);
                                        if nops(varscong) >= 2
then
                                                per := combinat
[permute](varscong, 2);
                                                for p in per do
                                                        subr
:= subs({varr[1] = p[1], varr[2] = p[2]}, r);
                                                        if
Is_SubExpr(lhs(subr), expr) then

Rkt := [op(Rkt), subr];
                                                        fi;
                                                od;
                                        fi;
                                        if varscong <> {} then
                                                for v in
varscong do
                                                        per
:= combinat[permute]({op(v)}, 2);
```

```
                                                                    for p
  in per do

  subr := subs({varr[1] = p[1], varr[2] = p[2]}, r);

  if Is_SubExpr(lhs(subr), expr) then

            Rkt := [op(Rkt), subr];

  fi;
                                                                    od;
                                                              od;
                                                        fi;
                                              fi;
                                    fi;
                                    #print("---   YOU ARE IN HERE ------------");
            od;

            if Rkt = [] then return FAIL; fi;
            return Heu_Rule_Exp(Rkt, expr);
  end: # Find_Rule_Exp
> TriSim[Mul_To_Sum] := proc(expr)
  # Hàm khai trin tích thành tng trong expr
  local le, temp, i,temp1, temp2;
            if type(expr,`^`) and op(2, expr) = -1 then
                      temp := Mul_To_Sum(op(1,expr));
                      if temp = FAIL then return FAIL;
                      else return 1/temp;fi;
            fi;
            if type(expr,`*`) then
                      le := [op(expr)];
                      temp1 :=[]; temp2 := [];
                      for i in le do
                                if type(i,`^`) and op(2,i) = -1 then
                                          temp2 := [op(temp2),i];
                                else
                                          temp1 := [op(temp1),i];
                                fi;
                      od;
                      #Có mu nhng t  = 1
                      if temp1 = [] then
                                temp2 := convert(temp2,`*`);
```

```
                                temp := Mul_To_Sum(denom(temp2));
                                if temp = FAIL then return FAIL;
                                else return 1/temp;fi;
                        fi;
                        # Có mu
                        if temp2  <> [] then
                                temp := [];
                                if convert(temp1,`*`) <> 1 then
temp := [convert(temp1,`*`)];fi;
                                temp := [op(temp), convert(temp2,`*
`)];
                                le := temp;
                                temp := map(t->Mul_To_Sum(t), temp);
                                if {op(temp)} = {FAIL} then return
FAIL;fi;
                                if temp[1] = FAIL then temp[1] :=
le[1];fi;
                                if temp[2] = FAIL then temp[2] :=
le[2];fi;
                                return temp[1]*temp[2];

                        # Không có mu
                        else
                                if expr <> expand(expr) then return
expand(expr);
                                else return FAIL;fi;
                        fi;
                fi;
                if type(expr,`+`) then
                        le := [op(expr)];
                        temp := map(t -> Mul_To_Sum(t), le);
                        if {op(temp)} = {FAIL} then
                                if expr <> expand(expr) then return
expand(expr);
                                else return FAIL;fi;
                        fi;
                        for i to nops(temp) do
                                if temp[i] = FAIL then
                                        temp[i] := le[i];
                                fi;
                        od;
                        temp := convert(temp,`+`);
```

```
                    return temp;
          fi;

          return FAIL;

  end: # Mul_To_Sum

> TriSim[Com_Denom] := proc(expr)
  # Hàm qui ng trong expr
  local le, des, lc, l, need, nu, de;

          if type(expr,`+`) then
                    le := [op(expr)];
                    des := map(t-> denom(t), le);
                    if {op(des)} = {1} then return FAIL;fi;
                    lc := lcm(op({op(des)}));
                    des :=[];
                    for l in le do
                              need := lc/ denom(l);
                              need := numer(l)*need;
                              des := [op(des), need];
                    od;
                    return convert(des,`+`)/lc;
          fi;
          if type(expr,`*`) and denom(expr) <> 1 then
                    if type(numer(expr),`+`) then nu := Com_Denom
  (numer(expr));
                    else nu := FAIL;fi;
                    if type(denom(expr),`+`) then de := Com_Denom
  (denom(expr));
                    else de := FAIL;fi;
                    if nu  = FAIL and de = FAIL then return FAIL;fi;
                    if nu = FAIL then nu := numer(expr);fi;
                    if de = FAIL then de := denom(expr); fi;
                    return nu/de;
          fi;
          if type(expr,`^`) then
                    if type(op(1,expr),`+`) then nu := Com_Denom
  (op(1,expr));
                    else nu := FAIL;fi;
                    if (type(op(2,expr),`+`)) then de :=
  Com_Denom(op(2,expr));
```

```
                    else de := FAIL;fi;
                    if nu  = FAIL and de = FAIL then return FAIL;fi;
                    if nu = FAIL then nu := op(1,expr);fi;
                    if de = FAIL then de := op(2,expr); fi;
                    return nu^de;
          fi;
          return FAIL;
end: # Com_Denom
> TriSim[Expand_Simple] := proc(expr)
# Hàm thc hin khai trin expr
local g, num, r, temp, varscong;
global SOL, used;
          used :=[];
          g := expr;
          # Th dùng hng ng thc
          r := Find_HDT_Exp(g);
          while r <> FAIL and not member(r, used) do
                    temp := Apply(r,g);
                    SOL := [op(SOL), [g, r, temp]];
                    used := [op(used),r];
                    g := temp;
                    r := Find_HDT_Exp(g);

          od;
          # Th dùng lut khai trin
          num := 0;
          r := Find_Rule_Exp(g);
          while r <> FAIL and not member(r, used) do
                    if num < 1 then
                              temp := Apply(r,g);
                              SOL := [op(SOL), [g, r, temp]];
                              used := [op(used),r];
                              g := temp;
                              num := num + 1;
                    else
                              if Has_Form(r, "tan=sin/cos") or
Has_Form(r, "cot=cos/sin") or
                                   Has_Form(r, "tan^2=1/cos^2-1")
or Has_Form(r, "cot^2=1/sin^2-1") then
                                        temp := Apply(r,g);
                                        SOL := [op(SOL), [g, r, temp]];
                                        used := [op(used),r];
```

```
                                        g := temp;
                                        num := num +1;
                                else
                                        varscong  :=
    Get_Var_bien_so(lhs(r));
                                        if varscong <> {} then
                                                temp := Apply(r,g);
                                                SOL := [op
    (SOL), [g, r, temp]];
                                                used := [op
    (used),r];
                                                g := temp;
                                                num := num +1;
                                        else
                                                break;
                                        fi;

                                fi;

                        fi;
                        r := Find_Rule_Exp(g);

                od;
                # Th bin i tích thành tng
                r := Mul_To_Sum(g);
                if r <> FAIL then
                        SOL := [op(SOL), [g, "Tích thành tng", r]];
                        return (r);
                fi;

                # Th qui ng mu thc
                r := Com_Denom(g);
                if r <> FAIL then
                        SOL := [op(SOL), [g, "Qui ng", r]];
                        return (r);
                fi;
                return g;


    end: # Expand_Simple
    >

    > TriSim[Expand_Simple_Rad] := proc(expr, dk)
```

```
# Hàm thc hin khai trin expr dng cn thc
local g, num, r, temp, mu;
global SOL, used;
          if not type(expr, `^`) or not type(op(2,expr),
fraction) then return expr;fi;
          used :=[];
          g := op(1,expr);
          mu := op(2,expr);
          # Th dùng hng ng thc
          r := Find_HDT_Exp(g);
          while r <> FAIL and not member(r, used) do
                    temp := Apply(r,g);
                    SOL := [op(SOL), [g^mu, r, temp^mu]];
                    used := [op(used),r];
                    g := temp;
                    r := Find_HDT_Exp(g);

          od;

          # Th dùng lut rút gn
          num := 0;
          r := Find_Rule_Exp(g);
          while r <> FAIL and not member(r, used) do
                    if num < 1 then
                              temp := Apply(r,g);
                              SOL := [op(SOL), [g^mu, r, temp^mu]];
                              used := [op(used),r];
                              g := temp;
                              num := num + 1;
                    else
                              if Has_Form(r, "tan=sin/cos") or
Has_Form(r, "cot=cos/sin") or
                                 Has_Form(r, "tan^2=1/cos^2-1")
or Has_Form(r, "cot^2=1/sin^2-1") then
                                        temp := Apply(r,g);
                                        SOL := [op(SOL), [g^mu,
r, temp^mu]];
                                        used := [op(used),r];
                                        g := temp;
                                        num := num +1;
                              else
                                        break;
```

```
                                fi;

                        fi;
                        r := Find_Rule_Exp(g);

                od;
                # Th bin i tích thành tng
                r := Mul_To_Sum(g);
                if r <> FAIL then
                        SOL := [op(SOL), [g^mu, "Tích thành tng",
  r^mu]];
                        return (r^mu);
                fi;

                # Th qui ng mu thc
                r := Com_Denom(g);
                if r <> FAIL then
                        SOL := [op(SOL), [g^mu, "Qui ng", r^mu]];
                        return (r^mu);
                fi;
                return g^mu;


  end: # Expand_Simple_Rad
> TriSim[Is_Common_Denom] := proc(expr)
  local mau, sete;
                if type(expr, `+`) then
                        sete := {op(expr)};
                        mau := map(s->denom(s), sete);
                        if nops(mau) = 1 then return true; fi;
                fi;
                return false;
  end:
> TriSim[Simplify_Tri_Step2] := proc(g)
  # Hàm thc hin bc 2 trong thut gii rút gn
  local h, l, temp;
  global count, gold, SOL;
                if Expr_Length(gold) <= 1 then return gold;fi;
                h := Expand_Simple(g);
                if h = g then return gold;
                else
                        l := Simplify_Simple(h);
```

```
                    if (Expr_Length(l) < Expr_Length(gold)) or
                              (Expr_Length(l) = Expr_Length(gold)
  and length(convert(l,string)) < length(convert(gold,string)))
  then
                              gold := l;
                              count := 0;
                              return Simplify_Tri_Step2(l);
                    else
                              if count < 3 then
                                        count := count +1;
                                        return Simplify_Tri_Step2
  (l);
                              else
                                        if Is_Common_Denom(gold) then
                                                  temp := numer
  (gold)/denom(gold);
                                                  SOL := [op
  (SOL), [gold, "Mu chung", temp]];
                                                  gold := temp;
                                        fi;
                                        return gold;
                              fi;
                    fi;
          fi;
  end: # Simplify_Tri_Step2
> TriSim[Omit_Sqrt] := proc(expr, dk)
  global SOL;
  local mu, f, g;
          if type(expr,`^`) then
                    mu := op(2,expr);
                    if type(mu, `fraction`) and mu = 1/2 then
                              f := op(1,expr);
                              g := sqrt(f) assuming op(dk);
                              if type(g, `^`) and op(2,g) = 1/2
  then return expr;fi;
                              SOL :=[op(SOL),[expr,dk,g]];
                              return g;
                    fi;
          fi;
          return expr;
  end:
> TriSim[Simplify_Tri_Step2_Rad] := proc(g, dk)
```

```
  # Hàm thc hin bc 2 trong thut gii rút gn dang can thuc
  local h, l, mu;
  global count, gold;
            if Expr_Length(gold) <= 1 then return gold;fi;
            h := Expand_Simple_Rad(g,dk);
            h := Omit_Sqrt(h,dk);
            if not type(h, `^`) or (type(h, `^`) and op(2,h) <>
  1/2) then return h;fi;
            if h = g then return gold;
            else
                        l := Simplify_Simple_Rad(h, dk); l :=
  Omit_Sqrt(l,dk);
                        if not type(l, `^`) or (type(l, `^`) and op
  (2,l) <> 1/2) then return l;fi;
                        if (Expr_Length(l) < Expr_Length(gold)) or
                                    (Expr_Length(l) = Expr_Length(gold)
  and length(convert(l,string)) < length(convert(gold,string)))
  then
                                    gold := l;
                                    count := 0;
                                    return Simplify_Tri_Step2_Rad(l,dk);
                        else
                                    if count < 3 then
                                                count := count +1;
                                                return
  Simplify_Tri_Step2_Rad(l,dk);
                                    else
                                                gold := Omit_Sqrt(gold, dk);
                                                return gold;
                                    fi;
                        fi;
            fi;
  end: # Simplify_Tri_Step2_Rad
> TriSim[Omit_Denom_Denom] := proc(expr)
  # Hàm kh mu  mu
  local setexpr, s, new, flag;
  global SOL;
            new := expr; flag := false;
            if type(expr, `+`) then
                        new := {};
                        setexpr := {op(expr)};
                        for s in setexpr do
```

```
                                if convert(numer(s)/denom(s),
  string) <> convert(s,string) then
                                        new := new union {numer
  (s)/denom(s)};
                                        flag := true;
                                else
                                        new := new union {s};
                                fi;
                        od;
                        new := convert([op(new)], `+`) ;
                        if flag = true then
                                SOL := [op(SOL), [expr, "Qui ng",
  new]];
                        fi;

                fi;
                return new;
  end:
> TriSim[Simplify_Tri_Poly] := proc(expr)
  # Hàm tng quát thc hin rút gn biu thc lng giác expr có
  dng a thc và phân thc
  local g, h, l, newexpr;
  global SOL, gold, count, goal;

                # Heuristic: Nu  mu có dng phân s thì thc hin
  qui ng.
                newexpr := Omit_Denom_Denom(expr);
                #  Bc 1
                g := Simplify_Simple(newexpr);
                gold := g;
                count := 0;

                #  Bc 2
                goal := Simplify_Tri_Step2(g);
                return goal;
  end:# Simplify_Tri_Poly
> TriSim[Simplify_Tri_Rad] := proc(expr, dk)
  # Hàm tng quát thc hin rút gn biu thc lng giác expr có
  dng a thc và phân thc
  local g, h, l;
  global SOL, gold, count, goal;
                #  Bc 1
```

```
          g := Simplify_Simple_Rad(expr, dk); g := Omit_Sqrt(g,
  dk);
          gold := g;
          count := 0;

          # Bc 2
          goal := Simplify_Tri_Step2_Rad(g, dk);
          return goal;
  end:# Simplify_Tri_Rad
> TriSim[Is_Radical] := proc(expr)
  # Hàm kim tra expr có phi dng cn thc
  local func, lfunc,f;
          func := indets(expr);
          lfunc := [];
          for f in func do
                    if type(f,`^`) and type(op(2,f), `fraction`) then
                              lfunc := [op(lfunc), f];
                    fi;
          od;
          if lfunc = [] then return false,[];
          else return true, lfunc;fi;
  end:# Is_Radical

> TriSim[Simplify_Tri] := proc(expr)
  # Hàm tng quát thc hin rút gn biu thc lng giác expr
  global SOL;
  local lfunc, temp, newexpr, dk, fl, f, solanlap, flag;
          SOL := [];
          solanlap := 0;
          lfunc := Is_Radical(expr);
          if not lfunc[1] then
                    return Simplify_Tri_Poly(expr);
          else
                    lfunc := lfunc[2];
                    if nargs = 2 then dk := args[2];
                    else dk := {};fi;

                    if nops(lfunc) = 1 and op(lfunc) = expr then

                              temp := Simplify_Tri_Rad(op(lfunc),dk);
                              return temp;
```

```
                    else

                                newexpr := expr;
                                flag := true;
                                while flag and solanlap <= 4 do
                                        solanlap := solanlap +1;
                                        for fl in lfunc do
                                                SOL := [op
(SOL), ["\t THC HIN RÚT GN: ",'g' =  fl]];
                                                temp :=
Simplify_Tri_Rad(fl, dk);
                                                newexpr := subs
(fl = temp, newexpr);
                                                SOL := [op
(SOL), ["\t SUY RA: ", 'f' = newexpr]];
                                        od;
                                        flag, lfunc := Is_Radical
(newexpr);
                                od;
                                SOL := [op(SOL), ["\t THC HIN RÚT
GN: ", 'f' = newexpr]];
                                return Simplify_Tri_Poly(newexpr);

                    fi;
          fi;

end: # Simplify_Tri
```

```
> TriSim[print_SOL] := proc()
  # Hàm in bài gii
  local s, num, temp;
  global SOL, goal;
          # Xóa bc gii tha
          temp :=[];
          for s in SOL do
                    if nops(s) = 3 then
                                if s[3] = goal then
                                        temp := [op(temp),  s];
                                        break;
                                else
                                        temp := [op(temp),  s];
                                fi;
```

```
                    else
                              temp := [op(temp), s];
                    fi;
          od;
          SOL := temp;
          num := 1;
          for s in SOL do
                    if nops(s) = 2 then
                              printf("%s\n", s[1]);
                              print(s[2]);
                    else
                              printf("\t Bc %d: \n", num);
                              if type(s[2], string) then printf
("\t\t %s\n",s[2]);
                              else printf("\t\t Áp dng: ");
print(s[2]); fi;
                              printf("\t\t Ta có: "); print('f' =
s[3]);
                              num := num +1;
                    fi;
          od;
          return;
end:

> url := currentdir();
save(TriSim,cat(url,"/TriSim.m"));
```

url := "D:\LUAN VAN THAC SI\CHUONG TRINH\2011-10-19" **(1)**

```
>
```