/\* regole base \*/

member(X,[X|Xs],1).

member(X,[Y|Ys],Pos) :- member(X,Ys,Pos\_sofar), Pos is Pos\_sofar+1.

nonmember(X,[Y|Ys]) :- X \== Y, nonmember(X,Ys).

nonmember(X,[]).

subterm(Term,Term).

subterm(Sub,Term) :- not(atomic(Term)), functor(Term,F,N), subterm(N,Sub,Term).

subterm(N,Sub,Term) :- N > 1, N1 is N-1, subterm(N1,Sub,Term).

subterm(N,Sub,Term) :- arg(N,Term,Arg), subterm(Sub,Arg).

subterm\_from\_sec(Term,Term).

subterm\_from\_sec(Sub,Term) :- not(atomic(Term)), functor(Term,F,N), subterm\_from\_sec(N,Sub,Term).

subterm\_from\_sec(N,Sub,Term) :- N > 2, N1 is N-1, subterm\_from\_sec(N1,Sub,Term).

subterm\_from\_sec(N,Sub,Term) :- arg(N,Term,Arg), subterm\_from\_sec(Sub,Arg).

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* regole base sugli operatori unari e binari \*/

same\_exp(X,Y) :- X==Y.

comm\_add\_exp(Stp,Op1Stp,Op2Stp,Class,Method) :- plus(Stp,Op1Stp,Op2Stp,Class,Method) ;

plus(Stp,Op2Stp,Op1Stp,Class,Method).

comm\_mul\_exp(Stp,Op1Stp,Op2Stp,Class,Method) :- times(Stp,Op1Stp,Op2Stp,Class,Method) ;

times(Stp,Op2Stp,Op1Stp,Class,Method).

comm\_equal\_exp(Stp,Op1Stp,Op2Stp,Class,Method) :- equal(Stp,Op1Stp,Op2Stp,Class,Method) ;

equal(Stp,Op2Stp,Op1Stp,Class,Method).

comm\_and\_exp(Stp,Op1Stp,Op2Stp,Class,Method) :- and(Stp,Op1Stp,Op2Stp,Class,Method) ;

and(Stp,Op2Stp,Op1Stp,Class,Method).

comm\_and\_and\_exp(Stp,Op1Stp,Op2Stp,Class,Method) :- and\_and(Stp,Op1Stp,Op2Stp,Class,Method) ;

and\_and(Stp,Op2Stp,Op1Stp,Class,Method).

comm\_or\_exp(Stp,Op1Stp,Op2Stp,Class,Method) :- or(Stp,Op1Stp,Op2Stp,Class,Method);

or(Stp,Op2Stp,Op1Stp,Class,Method).

comm\_or\_or\_exp(Stp,Op1Stp,Op2Stp,Class,Method) :- or\_or(Stp,Op1Stp,Op2Stp,Class,Method);

or\_or(Stp,Op2Stp,Op1Stp,Class,Method).

comm\_xor\_exp(Stp,Op1Stp,Op2Stp,Class,Method) :- xor(Stp,Op1Stp,Op2Stp,Class,Method);

xor(Stp,Op2Stp,Op1Stp,Class,Method).

comm\_not\_equal\_exp(Stp,Op1Stp,Op2Stp,Class,Method) :- not\_equal(Stp,Op1Stp,Op2Stp,Class,Method) ;

not\_equal(Stp,Op2Stp,Op1Stp,Class,Method).

comm\_exp(Stp,Op1Stp,Op2Stp,Class,Method) :- comm\_add\_exp(Stp,Op1Stp,Op2Stp,Class,Method) ;

comm\_mul\_exp(Stp,Op1Stp,Op2Stp,Class,Method) ;

comm\_and\_exp(Stp,Op1Stp,Op2Stp,Class,Method) ;

comm\_or\_exp(Stp,Op1Stp,Op2Stp,Class,Method) ;

comm\_and\_and\_exp(Stp,Op1Stp,Op2Stp,Class,Method) ;

comm\_or\_or\_exp(Stp,Op1Stp,Op2Stp,Class,Method) ;

comm\_xor\_exp(Stp,Op1Stp,Op2Stp,Class,Method) ;

comm\_equal\_exp(Stp,Op1Stp,Op2Stp,Class,Method) ;

comm\_not\_equal\_exp(Stp,Op1Stp,Op2Stp,Class,Method) .

binary\_exp(Stp,Op1Stp,Op2Stp,Class,Method) :- plus(Stp,Op1Stp,Op2Stp,Class,Method) ;

minus(Stp,Op1Stp,Op2Stp,Class,Method) ;

times(Stp,Op1Stp,Op2Stp,Class,Method) ;

divide(Stp,Op1Stp,Op2Stp,Class,Method) ;

equal(Stp,Op1Stp,Op2Stp,Class,Method) ;

not\_equal(Stp,Op1Stp,Op2Stp,Class,Method) ;

and(Stp,Op1Stp,Op2Stp,Class,Method) ;

or(Stp,Op1Stp,Op2Stp,Class,Method) ;

and\_and(Stp,Op1Stp,Op2Stp,Class,Method) ;

or\_or(Stp,Op1Stp,Op2Stp,Class,Method) ;

modulo(Stp,Op1Stp,Op2Stp,Class,Method) ;

xor(Stp,Op1Stp,Op2Stp,Class,Method) ;

less(Stp,Op1Stp,Op2Stp,Class,Method) ;

less\_equal(Stp,Op1Stp,Op2Stp,Class,Method) ;

greater(Stp,Op1Stp,Op2Stp,Class,Method) ;

greater\_equal(Stp,Op1Stp,Op2Stp,Class,Method) ;

left\_shift(Stp,Op1Stp,Op2Stp,Class,Method) ;

right\_shift(Stp,Op1Stp,Op2Stp,Class,Method) ;

unsigned\_right\_shift(Stp,Op1Stp,Op2Stp,Class,Method) .

unary\_exp(Stp,OpStp,Class,Method) :- complement(Stp,OpStp,Class,Method) ;

sign\_negate(Stp,OpStp,Class,Method) ;

post\_decr(Stp,OpStp,Class,Method);

post\_incr(Stp,OpStp,Class,Method);

pre\_decr(Stp,OpStp,Class,Method);

pre\_incr(Stp,OpStp,Class,Method);

not\_exp(Stp,OpStp,Class,Method).

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* espressioni di equivalenza \*/

equiv\_exp(X,Y,Class,Method) :- val\_inst(X,SameType,SameVal,Class,Method),

val\_inst(Y,SameType,SameVal,Class,Method).

equiv\_exp(X,Y,Class,Method) :- scalar\_var\_inst(X,SameIdent,Class,Method),

scalar\_var\_inst(Y,SameIdent,Class,Method).

equiv\_exp(X,Y,Class,Method) :- array\_elem\_inst(X,SameArrIdent,SubScrListX,Class,Method),

array\_elem\_inst(Y,SameArrIdent,SubScrListY,Class,Method),

equiv\_subscr\_list(SubScrListX,SubScrListY,Class,Method).

equiv\_exp(X,Y,Class,Method) :- plus(X,Op1X,Op2X,Class,Method),

comm\_add\_exp(Y,Op1Y,Op2Y,Class,Method),

equiv\_exp(Op1X,Op1Y,Class,Method),

equiv\_exp(Op2X,Op2Y,Class,Method).

equiv\_exp(X,Y,Class,Method) :- minus(X,Op1X,Op2X,Class,Method),

minus(Y,Op1Y,Op2Y,Class,Method),

equiv\_exp(Op1X,Op1Y,Class,Method),

equiv\_exp(Op2X,Op2Y,Class,Method).

equiv\_exp(X,Y,Class,Method) :- times(X,Op1X,Op2X,Class,Method),

comm\_mul\_exp(Y,Op1Y,Op2Y,Class,Method),

equiv\_exp(Op1X,Op1Y,Class,Method),

equiv\_exp(Op2X,Op2Y,Class,Method).

equiv\_exp(X,Y,Class,Method) :- divide(X,Op1X,Op2X,Class,Method),

divide(Y,Op1Y,Op2Y,Class,Method),

equiv\_exp(Op1X,Op1Y,Class,Method),

equiv\_exp(Op2X,Op2Y,Class,Method).

equiv\_exp(X,Y,Class,Method) :- sign\_negate(X,OpX,Class,Method),

sign\_negate(Y,OpY,Class,Method),

equiv\_exp(OpX,OpY,Class,Method).

equiv\_exp(X,Y,Class,Method) :- equal(X,Op1X,Op2X,Class,Method),

comm\_equal\_exp(Y,Op1Y,Op2Y,Class,Method),

equiv\_exp(Op1X,Op1Y,Class,Method),

equiv\_exp(Op2X,Op2Y,Class,Method).

equiv\_exp(X,Y,Class,Method) :- not\_equal(X,Op1X,Op2X,Class,Method),

not\_equal(Y,Op1Y,Op2Y,Class,Method),

equiv\_exp(Op1X,Op1Y,Class,Method),

equiv\_exp(Op2X,Op2Y,Class,Method).

equiv\_exp(X,Y,Class,Method) :- and(X,Op1X,Op2X,Class,Method),

comm\_and\_exp(Y,Op1Y,Op2Y,Class,Method),

equiv\_exp(Op1X,Op1Y,Class,Method),

equiv\_exp(Op2X,Op2Y,Class,Method).

equiv\_exp(X,Y,Class,Method) :- xor(X,Op1X,Op2X,Class,Method),

comm\_xor\_exp(Y,Op1Y,Op2Y,Class,Method),

equiv\_exp(Op1X,Op1Y,Class,Method),

equiv\_exp(Op2X,Op2Y,Class,Method).

equiv\_exp(X,Y,Class,Method) :- or(X,Op1X,Op2X,Class,Method),

comm\_or\_exp(Y,Op1Y,Op2Y,Class,Method),

equiv\_exp(Op1X,Op1Y,Class,Method),

equiv\_exp(Op2X,Op2Y,Class,Method).

equiv\_exp(X,Y,Class,Method) :- and\_and(X,Op1X,Op2X,Class,Method),

comm\_and\_and\_exp(Y,Op1Y,Op2Y,Class,Method),

equiv\_exp(Op1X,Op1Y,Class,Method),

equiv\_exp(Op2X,Op2Y,Class,Method).

equiv\_exp(X,Y,Class,Method) :- or\_or(X,Op1X,Op2X,Class,Method),

comm\_or\_or\_exp(Y,Op1Y,Op2Y,Class,Method),

equiv\_exp(Op1X,Op1Y,Class,Method),

equiv\_exp(Op2X,Op2Y,Class,Method).

equiv\_exp(X,Y,Class,Method) :- modulo(X,Op1X,Op2X,Class,Method),

modulo(Y,Op1Y,Op2Y,Class,Method),

equiv\_exp(Op1X,Op1Y,Class,Method),

equiv\_exp(Op2X,Op2Y,Class,Method).

equiv\_exp(X,Y,Class,Method) :- less(X,Op1X,Op2X,Class,Method),

less(Y,Op1Y,Op2Y,Class,Method),

equiv\_exp(Op1X,Op1Y,Class,Method),

equiv\_exp(Op2X,Op2Y,Class,Method).

equiv\_exp(X,Y,Class,Method) :- less\_equal(X,Op1X,Op2X,Class,Method),

less\_equal(Y,Op1Y,Op2Y,Class,Method),

equiv\_exp(Op1X,Op1Y,Class,Method),

equiv\_exp(Op2X,Op2Y,Class,Method).

equiv\_exp(X,Y,Class,Method) :- greater(X,Op1X,Op2X,Class,Method),

greater(Y,Op1Y,Op2Y,Class,Method),

equiv\_exp(Op1X,Op1Y,Class,Method),

equiv\_exp(Op2X,Op2Y,Class,Method).

equiv\_exp(X,Y,Class,Method) :- greater\_equal(X,Op1X,Op2X,Class,Method),

greater\_equal(Y,Op1Y,Op2Y,Class,Method),

equiv\_exp(Op1X,Op1Y,Class,Method),

equiv\_exp(Op2X,Op2Y,Class,Method).

equiv\_exp(X,Y,Class,Method) :- left\_shift(X,Op1X,Op2X,Class,Method),

left\_shift(Y,Op1Y,Op2Y,Class,Method),

equiv\_exp(Op1X,Op1Y,Class,Method),

equiv\_exp(Op2X,Op2Y,Class,Method).

equiv\_exp(X,Y,Class,Method) :- right\_shift(X,Op1X,Op2X,Class,Method),

right\_shift(Y,Op1Y,Op2Y,Class,Method),

equiv\_exp(Op1X,Op1Y,Class,Method),

equiv\_exp(Op2X,Op2Y,Class,Method).

equiv\_exp(X,Y,Class,Method) :- unsigned\_right\_shift(X,Op1X,Op2X,Class,Method),

unsigned\_right\_shift(Y,Op1Y,Op2Y,Class,Method),

equiv\_exp(Op1X,Op1Y,Class,Method),

equiv\_exp(Op2X,Op2Y,Class,Method).

equiv\_exp(X,Y,Class,Method) :- complement(X,OpX,Class,Method),

complement(Y,OpY,Class,Method),

equiv\_exp(OpX,OpY,Class,Method).

equiv\_exp(X,Y,Class,Method) :- post\_decr(X,OpX,Class,Method),

post\_decr(Y,OpY,Class,Method),

equiv\_exp(OpX,OpY,Class,Method).

equiv\_exp(X,Y,Class,Method) :- post\_incr(X,OpX,Class,Method),

post\_incr(Y,OpY,Class,Method),

equiv\_exp(OpX,OpY,Class,Method).

equiv\_exp(X,Y,Class,Method) :- pre\_decr(X,OpX,Class,Method),

pre\_decr(Y,OpY,Class,Method),

equiv\_exp(OpX,OpY,Class,Method).

equiv\_exp(X,Y,Class,Method) :- pre\_incr(X,OpX,Class,Method),

pre\_incr(Y,OpY,Class,Method),

equiv\_exp(OpX,OpY,Class,Method).

equiv\_exp(X,Y,Class,Method) :- not\_exp(X,OpX,Class,Method),

not\_exp(Y,OpY,Class,Method),

equiv\_exp(OpX,OpY,Class,Method).

equiv\_subscr\_list([X|Xs],[Y|Ys],Class,Method) :- equiv\_exp(X,Y,Class,Method),

equiv\_subscr\_list(Xs,Ys,Class,Method).

equiv\_subscr\_list([],[],Class,Method).

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

var\_inst(Stp,Ident,Class,Method) :- scalar\_var\_inst(Stp,Ident,Class,Method).

var\_inst(Stp,Ident,Class,Method) :- array\_elem\_inst(Stp,Ident,SubScrList,Class,Method).

inst(Stp,ValOrIdent,Class,Method) :- var\_inst(Stp,ValOrIdent,Class,Method).

inst\_in\_exp(ValOrIdent,ExpStp,Class,Method) :- inst(ExpStp,ValOrIdent,Class,Method).

inst\_in\_exp(ValOrIdent,ExpStp,Class,Method) :- binary\_exp(ExpStp,Op1Stp,Op2Stp,Class,Method),

(

inst\_in\_exp(ValOrIdent,Op1Stp,Class,Method) ;

inst\_in\_exp(ValOrIdent,Op2Stp,Class,Method)

).

inst\_in\_exp(ValOrIdent,ExpStp,Class,Method) :- unary\_exp(ExpStp,OpStp,Class,Method),

inst\_in\_exp(ValOrIdent,OpStp,Class,Method).

inst\_in\_exp(ValOrIdent,ExpStp,Class,Method) :- array\_elem\_inst(ExpStp,Dummy,SubScrList,Class,Method),

member(SubScrExp,SubScrList,Class,Method),

inst\_in\_exp(ValOrIdent,SubScrExp,Class,Method).

inst\_in\_exp(ValOrIdent,ExpStp,ExpStp,Class,Method) :- inst(ExpStp,ValOrIdent,Class,Method).

inst\_in\_exp(ValOrIdent,InstStp,ExpStp,Class,Method) :- binary\_exp(ExpStp,Op1Stp,Op2Stp,Class,Method),

(

inst\_in\_exp(ValOrIdent,InstStp,Op1Stp,Class,Method) ;

inst\_in\_exp(ValOrIdent,InstStp,Op2Stp,Class,Method)

).

inst\_in\_exp(ValOrIdent,InstStp,ExpStp,Class,Method) :- unary\_exp(ExpStp,OpStp,Class,Method),

inst\_in\_exp(ValOrIdent,InstStp,OpStp,Class,Method).

inst\_in\_exp(ValOrIdent,InstStp,ExpStp,Class,Method) :- array\_elem\_inst(ExpStp,Dummy,SubScrList,Class,Method),

member(SubScrExp,SubScrList,Class,Method),

inst\_in\_exp(ValOrIdent,InstStp,SubScrExp,Class,Method).

inst\_in\_exp\_notcheck\_subexps(ValOrIdent,ExpStp,SubExpStpList,Class,Method) :- not( member(ExpStp,SubExpStpList) ),

inst(ExpStp,ValOrIdent,Class,Method).

inst\_in\_exp\_notcheck\_subexps(ValOrIdent,ExpStp,SubExpStpList,Class,Method) :- not( member(ExpStp,SubExpStpList) ),

binary\_exp(ExpStp,Op1Stp,Op2Stp,Class,Method),

(

inst\_in\_exp\_notcheck\_subexps(ValOrIdent,Op1Stp,SubExpStpList,Class,Method) ;

inst\_in\_exp\_notcheck\_subexps(ValOrIdent,Op2Stp,SubExpStpList,Class,Method)

).

inst\_in\_exp\_notcheck\_subexps(ValOrIdent,ExpStp,SubExpStpList,Class,Method) :- not( member(ExpStp,SubExpStpList) ),

unary\_exp(ExpStp,OpStp,Class,Method),

inst\_in\_exp\_notcheck\_subexps(ValOrIdent,OpStp,SubExpStpList,Class,Method).

inst\_in\_exp\_notcheck\_subexps(ValOrIdent,ExpStp,SubExpStpList,Class,Method) :- not( member(ExpStp,SubExpStpList) ),

array\_elem\_inst(ExpStp,Dummy,SubScrList,Class,Method),

member(SubScrExp,SubScrList),

inst\_in\_exp\_notcheck\_subexps(ValOrIdent,SubScrExp,SubExpStpList,Class,Method).

sub\_exp\_in\_exp(SubExpStp,ExpStp,Class,Method) :- SubExpStp = ExpStp.

sub\_exp\_in\_exp(SubExpStp,ExpStp,Class,Method) :- binary\_exp(ExpStp,Op1Stp,Op2Stp,Class,Method),

(

sub\_exp\_in\_exp(SubExpStp,Op1Stp,Class,Method) ;

sub\_exp\_in\_exp(SubExpStp,Op2Stp,Class,Method)

).

sub\_exp\_in\_exp(SubExpStp,ExpStp,Class,Method) :- unary\_exp(ExpStp,OpStp,Class,Method),

sub\_exp\_in\_exp(SubExpStp,OpStp,Class,Method).

sub\_exp\_in\_exp(SubExpStp,ExpStp,Class,Method) :- array\_elem\_inst(ExpStp,Dummy,SubScrList,Class,Method),

member(SubScrExp,SubScrList),

sub\_exp\_in\_exp(SubExpStp,SubScrExp,Class,Method).

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

exp\_in\_stm(ExpStp,IDass,Assign) :- assign\_r(IDass,Assign,ExpStp,Class,Method).

exp\_in\_stm(ExpStp,IDdo,For) :- for\_r(IDfor,For,InitExpStp,ExpStp,StepExpStp,Class,Method),

(

sub\_exp\_in\_exp(ExpStp,InitExpStp,Class,Method) ;

sub\_exp\_in\_exp(ExpStp,StepExpStp,Class,Method)

).

exp\_in\_stm(ExpStp,IDforeach,Foreach) :- foreach\_r(IDforeach,Foreach,ExpStp,VarDeclStp,Class,Method),

sub\_exp\_in\_exp(ExpStp,VarDeclStp,Class,Method).

exp\_in\_stm(ExpStp,IDwhile,While) :- while\_r(IDwhile,While,ExpStp,Class,Method).

exp\_in\_stm(ExpStp,IDdo\_while,Do\_while) :- do\_while\_r(IDdo\_while,Do\_while,ExpStp,Class,Method).

exp\_in\_stm(ExpStp,IDlabeled,Labeled) :- labeled\_r(IDlabeled,Labeled,Ident\_label,LoopStp),

sub\_exp\_in\_exp(ExpStp,LoopStp).

/\*\*\* Necessita di essere rivista insieme al professore a causa di informazioni mancanti nella grammatica Java sull'espressione condizionale\*\*\*/

exp\_in\_stm(ExpStp,IDif,If) :- if\_r(IDif,If,ExpStp,Class,Method).

exp\_in\_stm(ExpStp,IDswitch,Switch) :- switch\_r(IDswitch,Switch,Ident\_label,Class,Method).

exp\_in\_stm(ExpStp,IDassert,Assert) :- assert\_r(IDassert,Assert,Class,Method).

exp\_in\_stm(ExpStp,IDbreak,Break) :- break\_r(IDbreak,Break,Ident\_label,Class,Method).

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

inherit\_dep(Concetto,ListaSottoConc) :-

findall( AltroConc,

(

between(1,Concetto,AltroConc),

findall( DepBranch,

( member(DepBranch,[true,false]),

( ( forall( member(SottoConc1,ListaSottoConc),

control\_dep(SottoConc1,AltroConc,DepBranch,Class,Method) ),

not( member(AltroConc,ListaSottoConc) )

) ->

assert\_ifnotalready(control\_dep(Concetto,AltroConc,DepBranch,Class,Method)) ;

true

),

( ( member(SottoConc2,ListaSottoConc),

control\_dep(AltroConc,SottoConc2,DepBranch,Class,Method),

not( member(AltroConc,ListaSottoConc) )

) ->

assert\_ifnotalready(control\_dep(AltroConc,Concetto,DepBranch,Class,Method)) ;

true

)

),

DepBranchList ),

forall( member(SottoConc3,ListaSottoConc),

(

forall( ( data\_dep(Type,SottoConc3,AltroConc,DepVar,DepLevel,Class,Method),

not( member(AltroConc,ListaSottoConc) )

),

assert\_ifnotalready(data\_dep(Type,Concetto,AltroConc,DepVar,DepLevel,Class,Method))

),

forall( ( data\_dep(Type,AltroConc,SottoConc3,DepVar,DepLevel,Class,Method),

not( member(AltroConc,ListaSottoConc) )

),

assert\_ifnotalready(data\_dep(Type,AltroConc,Concetto,DepVar,DepLevel,Class,Method))

),

forall( data\_dep(Type,SottoConc3,SottoConc3,DepVar,DepLevel,Class,Method),

assert\_ifnotalready(data\_dep(Type,Concetto,Concetto,DepVar,DepLevel,Class,Method))

)

)

)

),

AltroConcList ).

assert\_ifnotalready(Term) :-

( clause(Term,true) ->

true ;

assert(Term)

).

asserisci(Term) :-

( clause(Term,true) ->

true ;

( flag(global\_ID,GIDold,GIDold+1),

ID is GIDold + 1,

arg(1,Term,ID),

asserta( Term )

)

).

dep\_chain(HierChain,SrcStm,SinkStm,ListStm,ListVar) :-

chained(SrcStm,SinkStm,ListStm,ListVar,ListHierStm,[]),

HierChain =.. [dep\_chain,IN|ListHierStm],

flag(global\_ID,GIDold,GIDold+1),

ID is GIDold + 1,

asserisci( dep\_chain\_r(IN,HierChain,SrcStm,SinkStm,ListStm,ListVar) ),

inherit\_dep(IN,ListStm).

chained(SrcStm,SinkStm,[SrcStm|ListStm],[VarIdent|ListVar],[HierSrcStm|ListHierStm],ListAlreadyInChain) :-

SrcStm \== SinkStm,

assign\_r(SrcStm,HierSrcStm,\_,Class,Method),

data\_dep(true,MiddleSinkStm,SrcStm,VarIdent,0,Class,Method),

SrcStm \== MiddleSinkStm,

not( member(MiddleSinkStm,ListAlreadyInChain) ),

chained(MiddleSinkStm,SinkStm,ListStm,ListVar,ListHierStm,[SrcStm|ListAlreadyInChain]).

chained(SinkStm,SinkStm,[SinkStm],[],[HierSinkStm],ListAlreadyInChain) :-

assign\_r(SinkStm,HierSinkStm,\_,Class,Method).

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* problema con assign: informazione mancante \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

elem\_update\_r(ID,HierElUp,ElemUpdateStruct) :-

assign\_r(Def,HierDef,ElemDefStp,RhsDefStp,Class,Method),

var\_inst(ElemDefStp,ElemIdent,Class,Method),

assign\_r(Use,HierUse,LhsUseStp,RhsUseStp,Class,Method),

inst\_in\_exp(ElemIdent,ElemUseStp,RhsUseStp,Class,Method),

equiv\_exp(ElemDefStp,ElemUseStp,Class,Method),

( Def == Use ->

( HierElUp = elem\_update(ID,HierDef),

CompList = [Def],

ChainVarsList = [ElemIdent]

) ;

( data\_dep(anti,Def,Use,ElemIdent,0,Class,Method),

dep\_chain(HierChain,Use,Def,CompList,ChainVarsListButFirst),

append([ElemIdent],ChainVarsListButFirst,ChainVarsList),

HierElUp = elem\_update(IN,HierChain)

)

),

ElemUpdateStruct = elem\_update\_s(ElemIdent,[ElemDefStp,ElemUseStp],CompList,ChainVarsList),

asserisci( elem\_update\_r(IN,HierElUp,ElemUpdateStruct) ),

inherit\_dep(ID,CompList).

elem\_shift\_r(ID,elem\_shift(ID,HierElUp),ElemShiftStruct) :-

elem\_update\_r(ElUp,HierElUp,ElemShiftStruct),

ElemShiftStruct = elem\_update\_s(ElemIdent,[ElemDefStp,ElemUseStp],CompList,ChainVarsList),

forall( member(ChainStm,CompList,Pos),

( member(ChainVar,ChainVarsList,Pos),

assign\_r(ChainStm,\_,\_,ChainStmRhs,Class,Method),

inst\_descent\_of\_add\_or\_leftminus\_unique(ChainVar,ChainStmRhs,\_,Class,Method)),

forall( (member(Var,ChainVarsList), Var \== ChainVar),

not( inst\_in\_exp(Var,ChainStmRhs) )

)

)

),

asserisci( elem\_shift\_r(ID,elem\_shift(ID,HierElUp),ElemShiftStruct) ),

inherit\_dep(ID,[ElUp]).

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* ok\*/

inst\_descent\_of\_add\_or\_leftminus\_unique(Ident,Exp,Exp,Class,Method) :-

var\_inst(Exp,\_,Ident,Class,Method).

inst\_descent\_of\_add\_or\_leftminus\_unique(Ident,Exp,IdentStp,Class,Method) :-

( comm\_add\_exp(Exp,Op1,Op2,Class,Method) ;

minus(Exp,Op1,Op2,Class,Method)

),

inst\_descent\_of\_add\_or\_leftminus\_unique(Ident,Op1,IdentStp,Class,Method),

not( inst\_in\_exp(Ident,Op2,Class,Method) ).

/\*\*\*\*\*\*\*\*\*\*\* spiegazione al problema sollevato dal professore da trovare nei fatti prodotti da cocktail, SubScribe è parametro di uscita \*\*\*\*\*\*\*\*\*/

/\* forse l' ultima non va bene (solo Ident) per gli arrays \*/

inst\_descent\_of\_mul\_or\_leftdiv\_unique(Ident,Exp,Exp,Class,Method) :-

var\_inst(Exp,Ident,Class,Method).

inst\_descent\_of\_mul\_or\_leftdiv\_unique(Ident,Exp,IdentStp,Class,Method) :-

( comm\_mul\_exp(Exp,Op1,Op2,Class,Method) ;

divide(Exp,Op1,Op2,Class,Method)

),

inst\_descent\_of\_mul\_or\_leftdiv\_unique(Ident,Op1,IdentStp,Class,Method),

not( inst\_in\_exp(Ident,Op2,Class,Method) ).

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* spiegazione da trovare nei fatti prodotti da cocktail, SubScribe è parametro di uscita \*\*\*\*\*\*\*\*\*/

/\* forse l' ultima non va bene (solo Ident) per gli arrays \*/

inst\_descent\_of\_add\_or\_leftminus\_or\_mul\_or\_leftdiv\_unique(Ident,Exp,Exp,Class,Method) :-

var\_inst(Exp,Ident,Class,Method).

inst\_descent\_of\_add\_or\_leftminus\_or\_mul\_or\_leftdiv\_unique(Ident,Exp,IdentStp,Class,Method) :-

( comm\_add\_exp(Exp,Op1,Op2,Class,Method) ;

minus(Exp,Op1,Op2,Class,Method)

),

inst\_descent\_of\_add\_or\_leftminus\_or\_mul\_or\_leftdiv\_unique(Ident,Op1,IdentStp,Class,Method),

not( inst\_in\_exp(Ident,Op2,Class,Method) ).

inst\_descent\_of\_add\_or\_leftminus\_or\_mul\_or\_leftdiv\_unique(Ident,Exp,IdentStp,Class,Method) :-

( comm\_mul\_exp(Exp,Op1,Op2,Class,Method) ;

divide(Exp,Op1,Op2,Class,Method)

),

inst\_descent\_of\_add\_or\_leftminus\_or\_mul\_or\_leftdiv\_unique(Ident,Op1,IdentStp,Class,Method),

not( inst\_in\_exp(Ident,Op2,Class,Method) ).

inst\_descent\_of\_and\_or\_or\_unique(Ident,Exp,Exp,Class,Method) :-

var\_inst(Exp,Ident,Class,Method).

inst\_descent\_of\_and\_or\_or\_unique(Ident,Exp,IdentStp,Class,Method) :-

( comm\_and\_exp(Exp,Op1,Op2,Class,Method) ;

comm\_or\_exp(Exp,Op1,Op2,Class,Method)

),

inst\_descent\_of\_and\_or\_or\_unique(Ident,Op1,IdentStp,Class,Method),

not( inst\_in\_exp(Ident,Op2,Class,Method) ).

inst\_descent\_of\_xor\_unique(Ident,Exp,Exp,Class,Method) :-

var\_inst(Exp,Ident,Class,Method).

inst\_descent\_of\_xor\_unique(Ident,Exp,IdentStp,Class,Method) :-

comm\_xor\_exp(Exp,Op1,Op2,Class,Method),

inst\_descent\_of\_xor\_unique(Ident,Op1,IdentStp,Class,Method),

not( inst\_in\_exp(Ident,Op2,Class,Method) ).

inst\_descent\_of\_modulo\_unique(Ident,Exp,Exp,Class,Method) :-

var\_inst(Exp,Ident,Class,Method).

inst\_descent\_of\_modulo\_unique(Ident,Exp,IdentStp,Class,Method) :-

modulo(Exp,Op1,Op2,Class,Method),

inst\_descent\_of\_modulo\_unique(Ident,Op1,IdentStp,Class,Method),

not( inst\_in\_exp(Ident,Op2,Class,Method) ).

exp\_composed\_of\_add(Exp,Class,Method) :-

plus(Exp,Op1,Op2,Class,Method),

( inst(Op1,\_,Class,Method) ;

exp\_composed\_of\_add(Op1,Class,Method)

),

( inst(Op2,\_,Class,Method) ;

exp\_composed\_of\_add(Op2,Class,Method)

).

exp\_composed\_of\_divide(Exp,Class,Method) :-

divide(Exp,Op1,Op2,Class,Method),

( inst(Op1,\_,Class,Method) ;

exp\_composed\_of\_divide(Op1,Class,Method)

),

( inst(Op2,\_,Class,Method) ;

exp\_composed\_of\_divide(Op2,Class,Method)

).

exp\_composed\_of\_minus(Exp,Class,Method) :-

minus(Exp,Op1,Op2,Class,Method),

( inst(Op1,\_,Class,Method) ;

exp\_composed\_of\_minus(Op1,Class,Method)

),

( inst(Op2,\_,Class,Method) ;

exp\_composed\_of\_minus(Op2,Class,Method)

).

exp\_composed\_of\_and(Exp,Class,Method) :-

and(Exp,Op1,Op2,Class,Method),

( inst(Op1,\_,Class,Method) ;

exp\_composed\_of\_and(Op1,Class,Method)

),

( inst(Op2,\_,Class,Method) ;

exp\_composed\_of\_and(Op2,Class,Method)

).

exp\_composed\_of\_or(Exp,Class,Method) :-

or(Exp,Op1,Op2,Class,Method),

( inst(Op1,\_,Class,Method) ;

exp\_composed\_of\_or(Op1,Class,Method)

),

( inst(Op2,\_,Class,Method) ;

exp\_composed\_of\_or(Op2,Class,Method)

).

exp\_composed\_of\_xor(Exp,Class,Method) :-

or(Exp,Op1,Op2,Class,Method),

( inst(Op1,\_,Class,Method) ;

exp\_composed\_of\_xor(Op1,Class,Method)

),

( inst(Op2,\_,Class,Method) ;

exp\_composed\_of\_xor(Op2,Class,Method)

).

exp\_composed\_of\_mul(Exp,Class,Method) :-

times(Exp,Op1,Op2,Class,Method),

( inst(Op1,\_,Class,Method) ;

exp\_composed\_of\_mul(Op1,Class,Method)

),

( inst(Op2,\_,Class,Method) ;

exp\_composed\_of\_mul(Op2,Class,Method)

).

exp\_descent\_of\_divide(SubExp,Exp,Class,Method):- SubExp = Exp.

exp\_descent\_of\_divide(SubExp,Exp,Class,Method):-

divide(Exp,Op1,Op2,Class,Method),

exp\_descent\_of\_divide(SubExp,Op1,Class,Method).

exp\_descent\_of\_and(SubExp,Exp,Class,Method):- SubExp = Exp.

exp\_descent\_of\_and(SubExp,Exp,Class,Method):-

and(Exp,Op1,Op2,Class,Method),

exp\_descent\_of\_and(SubExp,Op1,Class,Method).

exp\_descent\_of\_or(SubExp,Exp,Class,Method):- SubExp = Exp.

exp\_descent\_of\_or(SubExp,Exp,Class,Method):-

or(Exp,Op1,Op2,Class,Method),

exp\_descent\_of\_or(SubExp,Op1,Class,Method).

exp\_descent\_of\_xor(SubExp,Exp,Class,Method):- SubExp = Exp.

exp\_descent\_of\_xor(SubExp,Exp,Class,Method):-

xor(Exp,Op1,Op2,Class,Method),

exp\_descent\_of\_xor(SubExp,Op1,Class,Method).

exp\_descent\_of\_modulo(SubExp,Exp,Class,Method):- SubExp = Exp.

exp\_descent\_of\_modulo(SubExp,Exp,Class,Method):-

modulo(Exp,Op1,Op2,Class,Method),

exp\_descent\_of\_modulo(SubExp,Op1,Class,Method).

exp\_descent\_of\_add\_or\_leftminus(SubExp,Exp,Class,Method):- SubExp = Exp.

exp\_descent\_of\_add\_or\_leftminus(SubExp,Exp,Class,Method):-

( comm\_add\_exp(Exp,Op1,Op2,Class,Method) ;

minus(Exp,Op1,Op2,Class,Method)

),

exp\_descent\_of\_add\_or\_leftminus(SubExp,Op1,Class,Method).

exp\_descent\_of\_mul(SubExp,Exp,Class,Method):- SubExp = Exp.

exp\_descent\_of\_mul(SubExp,Exp,Class,Method):-

comm\_mul\_exp(Exp,Op1,Op2,Class,Method),

exp\_descent\_of\_mul(SubExp,Op1,Class,Method).

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* ok \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\* concetti del professore da rivedere \*\*\*\*\*\*\*\*\*\*\*/

scan\_r(IN,HierScan,Kind,Range,Stride,IndexIdent,ArrayInstStp,SubScrExp,SubScrPos) :-

simple\_scan(Loop,HierLoop,ScanStm,HierScanStm,Range,Stride,IndexIdent,ArrayInstStp,SubScrExp,SubScrPos),

HierScan =.. [scan,IN,HierLoop,HierScanStm],

Kind = k\_simple\_scan,

asserisci( scan\_r(IN,HierScan,Kind,Range,Stride,IndexIdent,ArrayInstStp,SubScrExp,SubScrPos) ),

inherit\_dep(IN,[Loop,ScanStm]).

scan\_r(IN,HierScan,Kind,Range,Stride,IndexIdent,ArrayInstStp,SubScrExp,SubScrPos) :-

strip\_mined\_scan(OutLoop,HierOutLoop,InLoop,HierInLoop,ScanStm,HierScanStm,Range,Stride,IndexIdent,ArrayInstStp,SubScrExp,SubScrPos,StripSize,NumStrips),

HierScan =.. [strip\_mined\_scan,IN,HierOutLoop,HierInLoop,HierScanStm],

Kind = k\_strip\_mined\_scan\_constant(StripSize,NumStrips),

asserisci( scan\_r(IN,HierScan,Kind,Range,Stride,IndexIdent,ArrayInstStp,SubScrExp,SubScrPos) ),

inherit\_dep\_strip\_mined\_scan(IN,[OutLoop,InLoop,ScanStm]).

inherit\_dep\_strip\_mined\_scan(Concetto,ListaSottoConc) :-

findall( AltroConc,

(

between(1,Concetto,AltroConc),

findall( DepBranch,

( member(DepBranch,[true,false]),

( ( forall( member(SottoConc1,ListaSottoConc),

control\_dep(SottoConc1,AltroConc,DepBranch) ),

not( member(AltroConc,ListaSottoConc) )

) ->

assert\_ifnotalready(control\_dep(Concetto,AltroConc,DepBranch)) ;

true

),

/\* diff: AltroConc deve dipendere da entrambi OutLoop e InLoop \*/

( ( member(OutLoop,ListaSottoConc,1),

member(InLoop,ListaSottoConc,2),

control\_dep(AltroConc,OutLoop,DepBranch),

control\_dep(AltroConc,InLoop,DepBranch),

not( member(AltroConc,ListaSottoConc) )

) ->

assert\_ifnotalready(control\_dep(AltroConc,Concetto,DepBranch)) ;

true

)

),

DepBranchList ),

forall( member(SottoConc3,ListaSottoConc),

(

forall( ( data\_dep(Type,SottoConc3,AltroConc,DepVar,DepLevel),

not( member(AltroConc,ListaSottoConc) )

),

assert\_ifnotalready(data\_dep(Type,Concetto,AltroConc,DepVar,DepLevel))

),

forall( ( data\_dep(Type,AltroConc,SottoConc3,DepVar,DepLevel),

not( member(AltroConc,ListaSottoConc) )

),

assert\_ifnotalready(data\_dep(Type,AltroConc,Concetto,DepVar,DepLevel))

),

forall( data\_dep(Type,SottoConc3,SottoConc3,DepVar,DepLevel),

assert\_ifnotalready(data\_dep(Type,Concetto,Concetto,DepVar,DepLevel))

)

)

)

),

AltroConcList ).

simple\_scan(Loop,HierLoop,ScanStm,HierScanStm,Range,Stride,IndexIdent,ArrayInstStp,SubScrExp,SubScrPos) :-

count\_loop\_r(Loop,HierLoop,IndexIdent,InitExpStp,LastExpStp,StepExpStp),

control\_dep(ScanStm,Loop,true,Class,Method),

exp\_in\_stm(ArrayInstStp,ScanStm,HierScanStm),

arr\_el\_inst(ArrayInstStp,ArrayIdent,SubScrList),

array\_var(ArrayIdent,ArrayType,NDim,ArrayRangesList),

member(SubScrExp,SubScrList,SubScrPos),

member(ArrayDimRange,ArrayRangesList,SubScrPos),

inst\_in\_exp(IndexIdent,SubScrExp),

/\* migliorare: implementare an. simb. con i loop-bounds \*/

val\_inst(InitExpStp,integer,InitValueLB), /\* assume esp. simpl. e n. \*/

val\_inst(LastExpStp,integer,LastValueLB),

StepExpStp = lnull,

Stride = unit\_stride, /\* modificare \*/

nth1(1,ArrayDimRange,InitValueRange),

nth1(2,ArrayDimRange,LastValueRange),

ArrayDimRangeValue is (LastValueRange - InitValueRange + 1),

LoopRangeValue is (LastValueLB - InitValueLB + 1),

ArrayDimRangeValue == LoopRangeValue,

Range = whole\_array\_scan(LoopRangeValue).

strip\_mined\_scan(OutLoop,HierOutLoop,InLoop,HierInLoop,ScanStm,HierScanStm,

Range,Stride,InIndexIdent,ArrayInstStp,SubScrExp,SubScrPos,

StripSize,NumStrips) :-

count\_loop\_r(OutLoop,HierOutLoop,OutIndexIdent,OutInitExpStp,OutLastExpStp,

OutStepExpStp),

count\_loop\_r(InLoop,HierInLoop,InIndexIdent,InInitExpStp,InLastExpStp,

InStepExpStp),

control\_dep(InLoop,OutLoop,true),

control\_dep(ScanStm,InLoop,true),

exp\_in\_stm(ArrayInstStp,ScanStm,HierScanStm),

arr\_el\_inst(ArrayInstStp,ArrayIdent,SubScrList),

array\_var(ArrayIdent,ArrayType,NDim,ArrayRangesList),

member(SubScrExp,SubScrList,SubScrPos),

member(ArrayDimRange,ArrayRangesList,SubScrPos),

val\_inst(OutInitExpStp,integer,OutInitValue), /\* assume esp. simpl. e n. \*/

val\_inst(OutLastExpStp,integer,OutLastValue),

OutStepExpStp = lnull,

NumStrips is OutLastValue - OutInitValue + 1,

( inst\_in\_exp(OutIndexIdent,SubScrExp) ->

( val\_inst(InInitExpStp,integer,InInitValue),

val\_inst(InLastExpStp,integer,InLastValue),

InStepExpStp = lnull,

Stride = unit\_stride, /\* modificare \*/

StripSize is InLastValue - InInitValue + 1,

comm\_add\_exp(SubScrExp,SubScrAddOp1,SubScrAddOp2),

scal\_var\_inst(SubScrAddOp2,InIndexIdent),

( ( comm\_mul\_exp(SubScrAddOp1,SubScrMulOp1,SubScrMulOp2),

scal\_var\_inst(SubScrMulOp2,OutIndexIdent),

val\_inst(SubScrMulOp1,integer,StripSize),

0 is 1 - InInitValue - (OutInitValue \* StripSize)

) ;

( comm\_add\_exp(SubScrAddOp1,AddAddOp1,AddAddOp2),

comm\_mul\_exp(AddAddOp2,SubScrMulOp1,SubScrMulOp2),

scal\_var\_inst(SubScrMulOp2,OutIndexIdent),

val\_inst(SubScrMulOp1,integer,StripSize),

val\_inst(AddAddOp1,integer,TempValue1),

TempValue1 is 1 - InInitValue - (OutInitValue \* StripSize)

)

)

) ;

(

( ( comm\_mul\_exp(InInitExpStp,InInitMulOp1,InInitMulOp2),

scal\_var\_inst(InInitMulOp2,OutIndexIdent),

val\_inst(InInitMulOp1,integer,StripSize),

InInitValue is OutInitValue \* StripSize

) ;

( comm\_add\_exp(InInitExpStp,InInitAddOp1,InInitAddOp2),

comm\_mul\_exp(InInitAddOp2,InInitMulOp1,InInitMulOp2),

scal\_var\_inst(InInitMulOp2,OutIndexIdent),

val\_inst(InInitMulOp1,integer,StripSize),

val\_inst(InInitAddOp1,integer,TempValue2),

InInitValue is TempValue2 + (OutInitValue \* StripSize)

)

),

( ( comm\_mul\_exp(InLastExpStp,InLastMulOp1,InLastMulOp2),

scal\_var\_inst(InLastMulOp2,OutIndexIdent),

val\_inst(InLastMulOp1,integer,StripSize),

0 is InInitValue - 1 - (OutInitValue \* StripSize) + StripSize

) ;

( comm\_add\_exp(InLastExpStp,InLastAddOp1,InLastAddOp2),

comm\_mul\_exp(InLastAddOp2,InLastMulOp1,InLastMulOp2),

scal\_var\_inst(InLastMulOp2,OutIndexIdent),

val\_inst(InLastMulOp1,integer,StripSize),

val\_inst(InLastAddOp1,integer,TempValue3),

TempValue3 is InInitValue - 1 - OutInitValue \* StripSize + StripSize

)

),

InStepExpStp = lnull,

( ( scal\_var\_inst(SubScrExp,InIndexIdent),

InInitValue is 1

) ;

( comm\_add\_exp(SubScrExp,SubScrAddOp1,SubScrAddOp2),

scal\_var\_inst(SubScrAddOp2,InIndexIdent),

val\_inst(SubScrAddOp1,integer,TempValue4),

TempValue4 is 1 - InInitValue

)

)

)

),

nth1(1,ArrayDimRange,InitValueRange),

nth1(2,ArrayDimRange,LastValueRange),

ArrayDimRangeValue is (LastValueRange - InitValueRange + 1),

/\* implementare: StripSize\*NumStrips == ArrayDimRangeValue \*/

Range = whole\_array\_scan(ArrayDimRangeValue).

array\_update\_values\_r(IN,

array\_update\_values(IN,HierScanDef,HierScanUse,HierElUp),

ElemStruct,SubScrPos,IndexIdent) :-

elem\_update\_r(ElUp,HierElUp,ElemStruct),

ElemStruct = elem\_update\_s(ArrayIdent,[ArrayDefStp,ArrayUseStp],\_,\_),

array\_var(ArrayIdent,\_,\_,\_),

scan\_r(ScanDef,HierScanDef,KindScan,whole\_array\_scan(RangeValue),

unit\_stride,IndexIdent,ArrayDefStp,DefSubScrExp,SubScrPos),

control\_dep(ElUp,ScanDef,true),

scan\_r(ScanUse,HierScanUse,KindScan,whole\_array\_scan(RangeValue),

unit\_stride,IndexIdent,ArrayUseStp,UseSubScrExp,SubScrPos),

control\_dep(ElUp,ScanUse,true),

HierScanDef =.. [ConceptName,ScanDef|ScanDefHierList],

append(ScanDefHierListButLast,[HierScanDefStm],ScanDefHierList),

HierScanUse =.. [ConceptName,ScanUse|ScanUseHierList],

append(ScanUseHierListButLast,[HierScanUseStm],ScanUseHierList),

ScanDefHierListButLast == ScanUseHierListButLast,

asserisci( array\_update\_values\_r(IN,

array\_update\_values(IN,HierScanDef,HierScanUse,HierElUp),

ElemStruct,SubScrPos,IndexIdent) ),

inherit\_dep(IN,[ScanDef,ScanUse,ElUp]).

array\_shift\_values\_r(IN,

array\_shift\_values(IN,HierScanDef,HierScanUse,HierElSh),

ElemStruct,SubScrPos,IndexIdent) :-

elem\_shift\_r(ElSh,HierElSh,ElemStruct),

ElemStruct = elem\_update\_s(ArrayIdent,[ArrayDefStp,ArrayUseStp],\_,\_),

array\_var(ArrayIdent,\_,\_,\_),

scan\_r(ScanDef,HierScanDef,KindScan,whole\_array\_scan(RangeValue),

unit\_stride,IndexIdent,ArrayDefStp,DefSubScrExp,SubScrPos),

control\_dep(ElSh,ScanDef,true),

scan\_r(ScanUse,HierScanUse,KindScan,whole\_array\_scan(RangeValue),

unit\_stride,IndexIdent,ArrayUseStp,UseSubScrExp,SubScrPos),

control\_dep(ElSh,ScanUse,true),

HierScanDef =.. [ConceptName,ScanDef|ScanDefHierList],

append(ScanDefHierListButLast,[HierScanDefStm],ScanDefHierList),

HierScanUse =.. [ConceptName,ScanUse|ScanUseHierList],

append(ScanUseHierListButLast,[HierScanUseStm],ScanUseHierList),

ScanDefHierListButLast == ScanUseHierListButLast,

asserisci( array\_shift\_values\_r(IN,

array\_shift\_values(IN,HierScanDef,HierScanUse,HierElSh),

ElemStruct,SubScrPos,IndexIdent) ),

inherit\_dep(IN,[ScanDef,ScanUse,ElSh]).

shift\_reduction\_r(IN,HierRed,ElemAccumStruct,ArrayReducedStruct) :-

/\* puo' essere anche: elem\_scale; controllare e modificare \*/

elem\_shift\_r(ElSh,HierElSh,ElemAccumStruct),

ElemAccumStruct =

elem\_update\_s(ElemIdent,[ElemDefStp,ElemUseStp],ShiftCompList,\_),

last(ElemDefStm,ShiftCompList),

nth1(1,ShiftCompList,ElemUseStm),

scan\_r(Scan,HierScan,KindScan,whole\_array\_scan(RangeValue),

unit\_stride,IndexIdent,ArrayInstStp,SubScrExp,SubScrPos),

control\_dep(ElSh,Scan,true),

/\* dip. loop carried che chiude il ciclo con dep\_chain; controllare ev. probl.\*/

data\_dep(true,ElemUseStm,ElemDefStm,ElemIdent,DepLevel),

DepLevel \== 0,

/\* studiare perche' non basta: DepLevel = 1 \*/

HierScan =.. HierScanList,

last(HierScanStm,HierScanList),

assign\_r(ScanStm,HierScanStm,LhsExpStp,RhsExpStp),

inst\_in\_exp(ArrayIdent,ArrayInstStp,RhsExpStp), /\*per ctrl. che sia un uso\*/

dep\_chain(HierChainScanDef,ScanStm,ElemDefStm,ChainScanDefListStm,

ChainScanDefListVarButFirst),

append([ArrayIdent],ChainScanDefListVarButFirst,ChainScanDefListVar),

/\* definire per bene le prop. della riduzione, e appl. a ChainScanDefListStm \*/

not( inst\_in\_exp(IndexIdent,ElemDefStp) ),

not(( inst\_in\_exp(VarIdent,VarStp,ElemDefStp),

var\_inst(VarStp,VarIdent),

data\_dep(true,Stm,Scan,IndexIdent,AnyLevel1), /\* check il livello \*/

/\* att: non check i sbcn di Scan\*/

assign\_r(Stm,\_,\_,\_),

dep\_chain(\_,Stm,ElemDefStm,\_,ListVar),

last(VarIdent,ListVar)

)),

not(( data\_dep(true,ElemUseStm,Stm2,ElemIdent,AnyLevel2), /\* check il liv.\*/

Stm2 \== ElemUseStm,

control\_dep(Stm2,Scan,true),

Stm2 \== ElemDefStm

)),

( ScanStm == ElemDefStm ->

HierRed = shift\_reduction(IN,HierScan,HierElSh) ;

HierRed = shift\_reduction(IN,HierScan,HierElSh,HierChainScanDef)

),

ArrayReducedStruct =

array\_reduced\_s([ArrayIdent,ArrayInstStp,SubScrExp,SubScrPos],

IndexIdent,

ChainScanDefListStm,ChainScanDefListVar),

asserisci(shift\_reduction\_r(IN,HierRed,ElemAccumStruct,ArrayReducedStruct)),

( ScanStm == ElemDefStm ->

inherit\_dep(IN,[Scan,ElSh]) ;

inherit\_dep(IN,[Scan,ElSh|ChainScanDefListStm])

).

nonsimul\_relaxation\_step\_r(IN,HierNSRStep,NSRStepArrayStruct) :-

array\_update\_values\_r(AUVal,HierAUVal,AUValElUpStruct,AUValSubScrPos,

AUValIndexIdent),

shift\_reduction\_r(Red,HierRed,RedAccumStruct,RedArrStruct),

control\_dep(Red,AUVal,true),

RedArrStruct = array\_reduced\_s([ArrIdent,\_,\_,ArrSubScrPos],\_,\_,\_),

AUValSubScrPos == ArrSubScrPos,

AUValElUpStruct = elem\_update\_s(ArrIdent,\_,AUValElUpStmList,\_),

HierRed =.. [\_,Red,\_,HierRedElSh|\_],

HierRedElSh =..[\_,\_,HierRedElUp], HierRedElUp =.. [\_,RedElUp|\_],

HierAUVal =.. [\_,AUVal,\_,\_,HierAUValElUp],

HierAUValElUp =.. [\_,AUValElUp|\_],

last(AUValElUpDefStm,AUValElUpStmList),

RedAccumStruct = elem\_update\_s(RedAccumIdent,\_,RedAccumStmList,\_),

last(RedAccumDefStm,RedAccumStmList),

( RedElUp == AUValElUp ->

( HierNSRStep = nonsimul\_relaxation\_step(IN,HierAUVal,HierRed),

CompList = [AUVal,Red]

) ;

( dep\_chain(HierChain,RedAccumDefStm,AUValElUpDefStm,ChainStmList,\_),

/\*controllare che venga asserito questo fatto; altrimenti, ristrutturare tutto

data\_dep(true,AUVal,Red,RedAccumIdent,0), \*/

HierNSRStep = nonsimul\_relaxation\_step(IN,HierAUVal,HierRed,HierChain),

CompList = [AUVal,Red|ChainStmList]

)

),

NSRStepArrayStruct = nsr\_step\_array\_s(ArrIdent,ArrSubScrPos),

asserisci( nonsimul\_relaxation\_step\_r(IN,HierNSRStep,NSRStepArrayStruct) ),

inherit\_dep(IN,CompList).

dot\_product\_r(IN,HierDotProd,AccumStruct,Arr1Struct,Arr2Struct) :-

shift\_reduction\_r(Red1,HierRed1,AccumStruct,Arr1Struct),

shift\_reduction\_r(Red2,HierRed2,AccumStruct,Arr2Struct),

/\* restr: = AccumStruct per le due rid \*/

Red1 \== Red2,

HierRed1 =.. [shift\_reduction,Red1,HierScan1,HierElSh1|Dummy1],

HierRed2 =.. [shift\_reduction,Red2,HierScan2,HierElSh2|Dummy2],

HierElSh1 == HierElSh2, /\* restr: = elem\_shift per le due rid (ridond.) \*/

HierScan1 =.. [ScanName,Scan1|Scan1HierList],

append(Scan1HierListButLast,[HierScan1Stm],Scan1HierList),

HierScan2 =.. [ScanName,Scan2|Scan2HierList],

append(Scan2HierListButLast,[HierScan2Stm],Scan2HierList),

Scan1HierListButLast == Scan2HierListButLast,

/\* i due scan devono essere comp. dagli stessi count loop(s) \*/

Arr1Struct =

array\_reduced\_s([Arr1Ident,Arr1InstStp,Arr1SubScrExp,Arr1SubScrPos],

IndexIdent,

Arr1ChainListStm,Arr1ChainListVar),

Arr2Struct =

array\_reduced\_s([Arr2Ident,Arr2InstStp,Arr2SubScrExp,Arr2SubScrPos],

IndexIdent, /\* restr: = IndexIdent per le due strutt. \*/

Arr2ChainListStm,Arr2ChainListVar),

not( same\_exp(Arr1InstStp,Arr2InstStp) ), /\* forse ridond. \*/

equiv\_exp(Arr1SubScrExp,Arr2SubScrExp),

/\* migliorare con simb. an. e/o consid. le caratt. dello scan \*/

append(DistArr1ChainListStm,CommonArr12ChainListStm,Arr1ChainListStm),

append(DistArr2ChainListStm,CommonArr12ChainListStm,Arr2ChainListStm),

forall( member(DistArr1ChainStm,DistArr1ChainListStm,Pos1),

( assign\_r(DistArr1ChainStm,\_,\_,DistArr1ChainStmRhs),

member(DistArr1ChainVar,Arr1ChainListVar,Pos1),

inst\_descent\_of\_mul\_or\_leftdiv\_unique(DistArr1ChainVar,

DistArr1ChainStmRhs,\_),

not( inst\_in\_exp\_notcheck\_subexps(IndexIdent,DistArr1ChainStmRhs,

[Arr1SubScrExp,Arr2SubScrExp]) )

/\* ridondante con il check di sotto; ma serve per contr. Scan1Stm, che non viene

contr. dal check di sotto perche' e' un subconc. di Scan1 (data\_dep n.f.) \*/

)

),

forall( member(DistArr2ChainStm,DistArr2ChainListStm,Pos2),

( assign\_r(DistArr2ChainStm,\_,\_,DistArr2ChainStmRhs),

member(DistArr2ChainVar,Arr2ChainListVar,Pos2),

inst\_descent\_of\_mul\_or\_leftdiv\_unique(DistArr2ChainVar,

DistArr2ChainStmRhs,\_),

not( inst\_in\_exp\_notcheck\_subexps(IndexIdent,DistArr2ChainStmRhs,

[Arr1SubScrExp,Arr2SubScrExp]) )

)

),

length(DistArr1ChainListStm,Length1),

length(Temp1ListLength,Length1),

append(Temp1ListLength,TailArr1ChainListVar,Arr1ChainListVar),

length(DistArr2ChainListStm,Length2),

length(Temp2ListLength,Length2),

append(Temp2ListLength,TailArr2ChainListVar,Arr2ChainListVar),

TailArr1ChainListVar = [LastDistArr1ChainVar|CommonSubList],

TailArr2ChainListVar = [LastDistArr2ChainVar|CommonSubList],

/\* restr: controllo sulla parte comune: = CommonSubList \*/

nth1(1,CommonArr12ChainListStm,FirstCommonArr12ChainStm),

assign\_r(FirstCommonArr12ChainStm,\_,\_,FirstCommonArr12ChainStmRhs),

sub\_exp\_in\_exp(RootCommonExp,FirstCommonArr12ChainStmRhs),

exp\_composed\_of\_mul(RootCommonExp),

inst\_in\_exp(LastDistArr1ChainVar,RootCommonExp),

inst\_in\_exp(LastDistArr2ChainVar,RootCommonExp),

inst\_descent\_of\_add\_or\_leftminus\_or\_mul\_or\_leftdiv\_unique(

LastDistArr2ChainVar,FirstCommonArr12ChainStmRhs,\_),

forall( member(CommonArr12ChainStm,CommonArr12ChainListStm,Pos3),

( assign\_r(CommonArr12ChainStm,\_,\_,CommonArr12ChainStmRhs),

member(Arr12ChainVar,TailArr1ChainListVar,Pos3),

inst\_descent\_of\_add\_or\_leftminus\_or\_mul\_or\_leftdiv\_unique(

Arr12ChainVar,CommonArr12ChainStmRhs,\_)

)

),

AccumStruct = elem\_update\_s(\_,\_,AccumChainListStm,\_),

last(AccumDefStm,AccumChainListStm),

not(( data\_dep(true,TmpStm,Scan1,IndexIdent,AnyLevel),

/\* non check i sbcn di Scan; check il livello \*/

assign\_r(TmpStm,\_,\_,TmpStmRhs),

inst\_in\_exp\_notcheck\_subexps(IndexIdent,TmpStmRhs,

[Arr1SubScrExp,Arr2SubScrExp]),

dep\_chain(\_,TmpStm,AccumDefStm,\_,\_)

)),

HierDotProd = dot\_product(IN,HierRed1,HierRed2),

asserisci(dot\_product\_r(IN,HierDotProd,AccumStruct,Arr1Struct,Arr2Struct) ),

inherit\_dep(IN,[HierRed1,HierRed2]).

/\*

matrix\_vector\_multiply\_r(IN,HierMVMul,

\*/

array\_updated\_from\_temp\_var\_in\_count\_loop\_r(IN,using\_temp\_var(IN,HierTempUsing),

Loop,IndexIdent,TempInstStp,TempUpdate,ArrayIdent,ArrayInstStp) :-

scal\_var\_inst(TempInstStp,TempIdent),

assign\_r(TempUsing,HierTempUsing,ArrayInstStp,TempUsingRhs),

inst\_in\_exp(TempIdent,TempUsingRhs),

arr\_el\_inst(ArrayInstStp,ArrayIdent,Dummy1),

inst\_in\_exp(IndexIdent,ArrayInstStp),

control\_dep(TempUsing,Loop,true),

data\_dep(true,TempUsing,TempUpdate,TempIdent),

asserisci( array\_updated\_from\_temp\_var\_in\_count\_loop\_r(IN,

using\_temp\_var(IN,HierTempUsing),Loop,IndexIdent,TempInstStp,

TempUpdate,ArrayIdent,ArrayInstStp) ).

matrix\_vector\_multiply\_r(IN,MVStruct,

Array1Ident,ScPrPosInArray1,MVPosInArray1,

Array2Ident,ScPrPosInArray2,

ResArrayIdent,MVPosInResArray,

SubScrListArrEl2,SubScrListResArr) :-

scalar\_product\_r(ScalarProd,HierScalarProd,AccumScPrVar,

Array1Ident,SubScrListArrEl1,ScPrPosInArray1,

Array2Ident,SubScrListArrEl2,ScPrPosInArray2),

arg(2,HierScalarProd,HierScPrLoop),

arg(1,HierScPrLoop,ScPrLoop),

arg(3,HierScalarProd,HierScPrUpdateAccum),

arg(1,HierScPrUpdateAccum,ScPrUpdateAccum),

( strip\_mined\_scan\_r(Loop,HierLoop,IndexIdent,Array1InstStp,SubScrExpArr1,

StripSize,NumStrips) ;

count\_loop\_r(Loop,HierLoop,IndexIdent,InitExpStp,LastExpStp,StepExpStp)

),

/\*

count\_loop\_r(Loop,HierLoop,IndexIdent,InitExpStp,LastExpStp,StepExpStp),

\*/

control\_dep(ScPrUpdateAccum,Loop,true),

member(SubScrExpArr1,SubScrListArrEl1,MVPosInArray1),

MVPosInArray1 \== ScPrPosInArray1,

( functor(HierLoop,strip\_mined\_scan,4) ->

true ;

( scal\_var\_inst(SubScrExpArr1,IndexIdent),

array\_var(Array1Ident,ArrayType,NDim,Array1RangesList),

member(Range,Array1RangesList,MVPosInArray1)

/\* implementare: LastExpStp - InitExpStp + 1 = Range \*/

)

),

/\* forall(

(member(SubScrExpArr2,SubScrListArrEl2,PosInArray2),

PosInArray2 \== ScPrPosInArray2,

inst\_in\_exp(VarIdent,SubScrExpArr2),

var\_inst(Dummy10,VarIdent)

),

not(( data\_dep(true,ScPrUpdateAccum,Stm,VarIdent),

control\_dep(Stm,Loop,true)

))

), \*/

arg(2,AccumScPrVar,ScPrAccumInstList),

member(ScPrAccumStp,ScPrAccumInstList,2),

( ( arr\_el\_inst(ScPrAccumStp,ResArrayIdent,SubScrListResArr),

Temp1List = []

) ;

( array\_updated\_from\_temp\_var\_in\_count\_loop\_r(AccumUsing,HierAccumUsing,

Loop,IndexIdent,ScPrAccumStp,ScPrUpdateAccum,

ResArrayIdent,ResArrayInstStp),

/\*

( control\_dep(AccumUsing,ScPrLoop,true) ->

( count\_loop\_r(ScPrloop,HierScPrLoop,ScPrLoopIndexIdent,Dummy1,Dummy2,Dummy3),

not( inst\_in\_exp(ScPrLoopIndexIdent,ResArrStp) )

) ;

true

),

\*/

arr\_el\_inst(ResArrayInstStp,ResArrayIdent,SubScrListResArr),

Temp1List = [HierAccumUsing]

)

),

member(SubScrExpResArr,SubScrListResArr,MVPosInResArray),

equiv\_exp(SubScrExpArr1,SubScrExpResArr),

append([matrix\_vector\_multiply,IN,HierLoop,HierScalarProd],Temp1List,Temp2List),

MVStruct =.. Temp2List,

asserisci( matrix\_vector\_multiply\_r(IN,MVStruct,Array1Ident,ScPrPosInArray1,

MVPosInArray1,Array2Ident,ScPrPosInArray2,ResArrayIdent,

MVPosInResArray,SubScrListArrEl2,SubScrListResArr) ),

inherit\_dep(IN,[ScalarProd,Loop]).

matrix\_matrix\_multiply\_r(IN,matrix\_matrix\_multiply(IN,HierLoop,HierMatrVecMul),

Array1Ident,ScPrPosInArray1,MmPosInArray1,

Array2Ident,ScPrPosInArray2,MmPosInArray2,

ResArrayIdent,MmPos1InResArray,MmPos2InResArray) :-

matrix\_vector\_multiply\_r(MatrVecMul,HierMatrVecMul,

Array1Ident,ScPrPosInArray1,MmPosInArray1,

Array2Ident,ScPrPosInArray2,

ResArrayIdent,MmPos1InResArray,

SubScrListArr2,SubScrListResArr),

arg(2,HierMatrVecMul,HierMVLoop),

arg(1,HierMVLoop,MVLoop),

arg(3,HierMatrVecMul,HierScalarProd),

arg(1,HierScalarProd,ScalarProd),

arg(2,HierScalarProd,HierScPrLoop),

arg(1,HierScPrLoop,ScPrLoop),

arg(3,HierScalarProd,HierScPrUpdateAccum),

arg(1,HierScPrUpdateAccum,ScPrUpdateAccum),

( strip\_mined\_scan\_r(Loop,HierLoop,IndexIdent,Array2InstStp,SubScrExpArr2,

StripSize,NumStrips) ;

count\_loop\_r(Loop,HierLoop,IndexIdent,InitExpStp,LastExpStp,StepExpStp)

),

/\*

count\_loop\_r(Loop,HierLoop,IndexIdent,InitExpStp,LastExpStp,StepExpStp),

\*/

control\_dep(ScPrUpdateAccum,Loop,true),

/\* in effetti serve, ma rallenta meledettamente

abs\_control\_dep(MVLoop,Loop,true),

\*/

member(SubScrExpArr2,SubScrListArr2,MmPosInArray2),

MmPosInArray2 \== ScPrPosInArray2,

( functor(HierLoop,strip\_mined\_scan,4) ->

true ;

( scal\_var\_inst(SubScrExpArr2,IndexIdent),

array\_var(Array2Ident,ArrayType,NDim,Array2RangesList),

member(Range,Array2RangesList,MmPosInArray2)

/\* implementare: LastExpStp - InitExpStp + 1 = Range \*/

)

),

member(SubScrExpResArr,SubScrListResArr,MmPos2InResArray),

equiv\_exp(SubScrExpArr2,SubScrExpResArr),

MmPos1InResArray \== MmPos2InResArray,

asserisci( matrix\_matrix\_multiply\_r(IN,

matrix\_matrix\_multiply(IN,HierLoop,HierMatrVecMul),Array1Ident,

ScPrPosInArray1,MmPosInArray1,Array2Ident,ScPrPosInArray2,

MmPosInArray2,ResArrayIdent,MmPos1InResArray,MmPos2InResArray) ),

inherit\_dep(IN,[MatrVecMul,Loop]).

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* fine regole professore \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* nuovi concetti \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

swap\_r(IdentVar1,IdentVar2,SwapSourceStp,SwapSinkStp,Var2AssignStp,Class,Method) :-

(

(

(

data\_dep('true',SwapSinkStp,SwapSourceStp,IdentSwap,\_,Class,Method),

assign\_r(IDswap,assign(IDswap,SwapSourceStp),SwapSourceStp,Class,Method)

),

(

data\_dep('anti',SwapSinkStp,Var2AssignStp,IdentVar2,\_,Class,Method),

assign\_r(IDvar2,assign(IDvar2,Var2AssignStp),Var2AssignStp,Class,Method)

),

(

data\_dep('anti',Var2AssignStp,SwapSourceStp,IdentVar1,\_,Class,Method),

assign\_r(IDvar1,assign(IDvar1,SwapSourceStp),SwapSourceStp,Class,Method)

)

),

(

control\_dep(SwapSourceStp,Stp,\_,Class,Method),

control\_dep(SwapSinkStp,Stp,\_,Class,Method),

control\_dep(Var2AssignStp,Stp,\_,Class,Method)

),

not( data\_dep('output',\_,Var2AssignStp,\_,\_,Class,Method) )

);

(

(

(

data\_dep('true',SwapSinkStp,SwapSourceStp,IdentSwap,\_,Class,Method),

assign\_r(IDswap,assign(IDswap,SwapSourceStp),SwapSourceStp,Class,Method)

),

(

data\_dep('anti',DummyStp,SwapSourceStp,IdentVar2,\_,Class,Method),

assign\_r(IDdummy,assign(IDdummy,DummyStp),DummyStp,Class,Method)

),

(

data\_dep('output',Var2AssignStp,DummyStp,IdentVar2,\_,Class,Method),

assign\_r(IDvar2,assign(IDvar2,Var2AssignStp),Var2AssignStp,Class,Method)

),

(

data\_dep('anti',SwapSinkStp,Var2AssignStp,IdentVar1,\_,Class,Method),

assign\_r(IDvar1,assign(IDvar1,SwapSinkStp),SwapSinkStp,Class,Method)

)

),

(

control\_dep(SwapSourceStp,Stp,\_,Class,Method),

control\_dep(SwapSinkStp,Stp,\_,Class,Method),

control\_dep(Var2AssignStp,Stp,\_,Class,Method)

)

),

asserta( swap(IdentVar1,IdentVar2) ).

array\_inversion\_r(IdentArray,Class,Method) :- (

swap\_r(IdentVar2,IdentVar1,SwapSourceStp,SwapSinkStp,Var2AssignStp,Class,Method),

IdentVar1 == IdentVar2,

IdentArray = IdentVar1

),

(

control\_dep(SwapSourceStp,Stp,\_,Class,Method),

control\_dep(SwapSinkStp,Stp,\_,Class,Method),

control\_dep(Var2AssignStp,Stp,\_,Class,Method)

),

for\_r(IDfor,for(IDfor,Stp),InitStp,Stp,IncrStp,Class,Method),

asserta( array\_inversion(IdentArray) ).

array\_sort\_r(IdentArray,Class,Method) :- (

swap\_r(IdentVar2,IdentVar1,SwapSourceStp,SwapSinkStp,Var2AssignStp,Class,Method),

IdentVar1 == IdentVar2,

IdentArray = IdentVar1

),

(

control\_dep(SwapSourceStp,IfStp,\_,Class,Method),

control\_dep(SwapSinkStp,IfStp,\_,Class,Method),

control\_dep(Var2AssignStp,IfStp,\_,Class,Method)

),

(

if\_r(IDif,if(IDif,IfStp),IfStp,Class,Method),

control\_dep(IfStp,InnerForStp,\_,Class,Method),

nested\_for\_r(Nfor,InnerForStp)

),

asserta( array\_sort(IdentArray) ).

nested\_for\_r(Nfor,InnerForStp) :- (

for\_r(IDfor,for(IDfor,InnerForStp),InitInnerStp,InnerForStp,IncrInnerStp,Class,Method),

control\_dep(InnerForStp,OuterForStp,\_,Class,Method),

for\_r(IDfor2,for(IDfor2,OuterForStp),InitOuterStp,OuterForStp,IncrOuterStp,Class,Method),

Nfor is 2,

asserta( nested\_for(Nfor) ),!

);

(

for\_r(IDfor,for(IDfor,InnerForStp),InitInnerStp,InnerForStp,IncrInnerStp,Class,Method),

control\_dep(InnerForStp,MiddleForStp,\_,Class,Method),

for\_r(IDfor2,for(IDfor2,MiddleForStp),InitMiddleStp,MiddleForStp,IncrMiddleStp,Class,Method),

control\_dep(MiddleForStp,OuterForStp,\_,Class,Method),

for\_r(IDfor3,for(IDfor3,OuterForStp),InitOuterStp,OuterForStp,IncrOuterStp,Class,Method),

Nfor is 3,

asserta( nested\_for(Nfor) ),!

).

count\_loop\_r(ID,count\_loop(ID,X),CntVarIdent,InitExpStp,StepExpStp) :-

(

for\_r(IDfor,X,InitExpStp,Stp,StepExpStp,Class,Method),

scalar\_var\_inst(CntVarIdent,InitExpStp,Class,Method),

ID=Stp,

asserta( count\_loop(ID,count\_loop(ID,X),CntVarIdent,InitExpStp,StepExpStp) ),!

);

(

while\_r(IDwhile,X,Stp,Class,Method),

control\_dep(InitStp,Stp,\_,Class,Method),

(

post\_decr(InitStp,VarDefStp,Class,Method);

post\_incr(InitStp,VarDefStp,Class,Method);

pre\_decr(InitStp,VarDefStp,Class,Method);

pre\_incr(InitStp,VarDefStp,Class,Method)

),

scalar\_var\_def(CntVarIdent,VarDefStp,Class,Method),

InitExpStp = InitStp,

StepExpStp = InitExpStp,

not( control\_dep(VarDefStp,Stp,\_,Class,Method) ),

ID=Stp,

asserta( count\_loop(ID,count\_loop(ID,X),CntVarIdent,InitExpStp,StepExpStp) ),!

);

(

do\_while\_r(IDdo\_while,X,Stp,Class,Method),

control\_dep(InitStp,Stp,\_,Class,Method),

(

post\_decr(InitStp,VarDefStp,Class,Method);

post\_incr(InitStp,VarDefStp,Class,Method);

pre\_decr(InitStp,VarDefStp,Class,Method);

pre\_incr(InitStp,VarDefStp,Class,Method)

),

scalar\_var\_def(CntVarIdent,VarDefStp,Class,Method),

InitExpStp = InitStp,

StepExpStp = InitExpStp,

not( control\_dep(VarDefStp,Stp,\_,Class,Method) ),

ID=Stp,

asserta( count\_loop(ID,count\_loop(ID,X),CntVarIdent,InitExpStp,StepExpStp) ),!

).

scalar\_map\_r(ArrayVarName) :- (

for\_r(IDfor,for(IDfor,ForStp),InitStp,ForStp,StepStp,Class,Method),

assign\_r(\_,assign(\_,InitStp),InitStp,Class,Method),

scalar\_var\_inst(IdentVarName,InitStp,Class,Method)

),

assign\_r(\_,assign(\_,VarStp),VarStp,Class,Method),

scalar\_var\_inst(IdentVarName,VarStp,Class,Method),

array\_reference(ArrayVarName,VarStp,[IdentVarName],Class,Method),

control\_dep(VarStp,ForStp,\_,Class,Method),

asserta( scalar\_map(ArrayVarName) ).

scalar\_array\_reduction\_r(ScalarVarName,ArrayVarName) :- assign\_r(ID,assign(ID,Stp),Stp,Class,Method),

scalar\_var\_inst(ScalarVarName,Stp,Class,Method),

array\_reference(ArrayVarName,Stp,\_,Class,Method),

asserta( scalar\_array\_reduction(ScalarVarName,ArrayVarName) ).

basic\_linear\_induction\_variable\_r(IdentVarName,Tipo) :- (

for\_r(IDfor,for(IDfor,ForStp),InitStp,ForStp,StepStp,Class,Method),

assign\_r(\_,assign(\_,InitStp),InitStp,Class,Method),

scalar\_var\_inst(IdentVarName,InitStp,Class,Method)

),

(

assign\_r(IDassign,assign(IDassign,VarStp),VarStp,Class,Method),

scalar\_var\_inst(IdentVarName,VarStp,Class,Method),

(

(

plus(VarStp,IdentVarName,Op2,Class,Method);

minus(VarStp,IdentVarName,Op2,Class,Method)

),

VarStp\==StepStp

),

not(

(

assign\_r(\_,assign(\_,Op2Stp),Op2Stp,Class,Method),

(

scalar\_var\_inst(Op2,Op2Stp,Class,Method);

array\_elem\_inst(Op2,Op2Stp,[IdentVarName],Class,Method)

),

control\_dep(Op2Stp,ForStp,\_,Class,Method)

)

)

),

control\_dep(VarStp,ForStp,\_,Class,Method),

Tipo = basic\_linear,

asserta( basic\_linear\_induction\_variable(IdentVarName,Tipo) ).

basic\_geometric\_induction\_variable\_r(IdentVarName,Tipo) :- (

for\_r(IDfor,for(IDfor,ForStp),InitStp,ForStp,StepStp,Class,Method),

assign\_r(\_,assign(\_,InitStp),InitStp,Class,Method),

scalar\_var\_inst(IdentVarName,InitStp,Class,Method)

),

(

assign\_r(IDassign,assign(IDassign,VarStp),VarStp,Class,Method),

scalar\_var\_inst(IdentVarName,VarStp,Class,Method),

(

(

times(VarStp,IdentVarName,Op2,Class,Method);

divide(VarStp,IdentVarName,Op2,Class,Method)

),

VarStp\==StepStp

),

not(

(

assign\_r(\_,assign(\_,Op2Stp),Op2Stp,Class,Method),

(

scalar\_var\_inst(Op2,Op2Stp,Class,Method);

array\_elem\_inst(Op2,Op2Stp,[IdentVarName],Class,Method)

),

control\_dep(Op2Stp,ForStp,\_,Class,Method)

)

)

),

control\_dep(VarStp,ForStp,\_,Class,Method),

Tipo = basic\_geometric,

asserta( basic\_geometric\_induction\_variable(IdentVarName,Tipo) ).

linear\_induction\_variable\_r(IdentVarName,Tipo) :- while\_r(IDwhile,while(IDwhile,WhileStp),WhileStp,Class,Method),

(

assign\_r(IDassign,assign(IDassign,VarStp),VarStp,Class,Method),

scalar\_var\_inst(CntVarIdent,VarStp,Class,Method),

(

(

plus(VarStp,CntVarIdent,Op2,Class,Method);

minus(VarStp,CntVarIdent,Op2,Class,Method)

)

),

not(

(

assign\_r(\_,assign(\_,Op2Stp),Op2Stp,Class,Method),

(

scalar\_var\_inst(Op2,Op2Stp,Class,Method);

array\_elem\_inst(Op2,Op2Stp,[CntVarIdent],Class,Method)

),

control\_dep(Op2Stp,WhileStp,\_,Class,Method),

scalar\_var\_def(Op2,Op2DefStp,\_,\_,Class,Method),

not( control\_dep(Op2DefStp,WhileStp,\_,Class,Method) )

)

)

),

control\_dep(VarStp,WhileStp,\_,Class,Method),

scalar\_var\_def(CntVarIdent,VarDefStp,\_,\_,Class,Method),

not( control\_dep(VarDefStp,WhileStp,\_,Class,Method) ),

IdentVarName = CntVarIdent,

Tipo = linear,

asserta( linear\_induction\_variable(IdentVarName,Tipo) ).

geometric\_induction\_variable\_r(IdentVarName,Tipo) :- while\_r(IDwhile,while(IDwhile,WhileStp),WhileStp,Class,Method),

(

assign\_r(IDassign,assign(IDassign,VarStp),VarStp,Class,Method),

scalar\_var\_inst(CntVarIdent,VarStp,Class,Method),

(

(

times(VarStp,CntVarIdent,Op2,Class,Method);

divide(VarStp,CntVarIdent,Op2,Class,Method)

)

),

not(

(

assign\_r(\_,assign(\_,Op2Stp),Op2Stp,Class,Method),

(

scalar\_var\_inst(Op2,Op2Stp,Class,Method);

array\_elem\_inst(Op2,Op2Stp,[CntVarIdent],Class,Method)

),

control\_dep(Op2Stp,WhileStp,\_,Class,Method),

scalar\_var\_def(Op2,Op2DefStp,\_,\_,Class,Method),

not( control\_dep(Op2DefStp,WhileStp,\_,Class,Method) )

)

)

),

control\_dep(VarStp,WhileStp,\_,Class,Method),

scalar\_var\_def(CntVarIdent,VarDefStp,\_,\_,Class,Method),

not( control\_dep(VarDefStp,WhileStp,\_,Class,Method) ),

IdentVarName = CntVarIdent,

Tipo = geometric,

asserta( linear\_induction\_variable(IdentVarName,Tipo) ).

basic\_polynomial\_induction\_variable\_r(BasicIdentVarName,OuterIdentVarName,Tipo) :-

(

for\_r(IDfor,for(IDfor,ForStp),InitStp,ForStp,StepStp,Class,Method),

assign\_r(\_,assign(\_,InitStp),InitStp,Class,Method),

scalar\_var\_inst(BasicIdentVarName,InitStp,Class,Method)

),

(

assign\_r(IDassign,assign(IDassign,BasicVarStp),BasicVarStp,Class,Method),

scalar\_var\_inst(BasicIdentVarName,BasicVarStp,Class,Method),

(

(

plus(BasicVarStp,BasicIdentVarName,Op2,Class,Method);

minus(BasicVarStp,BasicIdentVarName,Op2,Class,Method)

),

BasicVarStp\==StepStp,

BasicVarStp\==InitStp

),

not(

(

assign\_r(\_,assign(\_,Op2Stp),Op2Stp,Class,Method),

(

scalar\_var\_inst(Op2,Op2Stp,Class,Method);

array\_elem\_inst(Op2,Op2Stp,[BasicIdentVarName],Class,Method)

),

control\_dep(Op2Stp,ForStp,\_,Class,Method)

)

),

control\_dep(BasicVarStp,ForStp,\_,Class,Method)

),

(

scalar\_var\_def(OuterIdentVarName,OuterVarStp,\_,\_,Class,Method),

not( control\_dep(OuterVarStp,ForStp,\_,Class,Method) ),

assign\_r(ID2assign,assign(ID2assign,InnerVarStp),InnerVarStp,Class,Method),

scalar\_var\_inst(OuterIdentVarName,InnerVarStp,Class,Method),

(

(

plus(InnerVarStp,OuterIdentVarName,BasicIdentVarName,Class,Method);

minus(InnerVarStp,OuterIdentVarName,BasicIdentVarName,Class,Method)

),

InnerVarStp\==StepStp,

InnerVarStp\==InitStp

),

control\_dep(InnerVarStp,ForStp,\_,Class,Method)

),

Tipo = basic\_polynomial,

asserta( basic\_polynomial\_induction\_variable(BasicIdentVarName,OuterIdentVarName,Tipo) ).

polynomial\_induction\_variable\_r(IdentVarName,OuterIdentVarName,Tipo) :-

while\_r(IDwhile,while(IDwhile,WhileStp),WhileStp,Class,Method),

(

assign\_r(IDassign,assign(IDassign,VarStp),VarStp,Class,Method),

scalar\_var\_inst(CntVarIdent,VarStp,Class,Method),

(

(

plus(VarStp,CntVarIdent,Op2,Class,Method);

minus(VarStp,CntVarIdent,Op2,Class,Method)

)

),

not(

(

assign\_r(\_,assign(\_,Op2Stp),Op2Stp,Class,Method),

(

scalar\_var\_inst(Op2,Op2Stp,Class,Method);

array\_elem\_inst(Op2,Op2Stp,[CntVarIdent],Class,Method)

),

control\_dep(Op2Stp,WhileStp,\_,Class,Method),

scalar\_var\_def(Op2,Op2DefStp,\_,\_,Class,Method),

not( control\_dep(Op2DefStp,WhileStp,\_,Class,Method) )

)

),

control\_dep(VarStp,WhileStp,\_,Class,Method)

),

(

scalar\_var\_def(OuterIdentVarName,OuterVarStp,\_,\_,Class,Method),

not( control\_dep(OuterVarStp,WhileStp,\_,Class,Method) ),

assign\_r(ID2assign,assign(ID2assign,InnerVarStp),InnerVarStp,Class,Method),

scalar\_var\_inst(OuterIdentVarName,InnerVarStp,Class,Method),

(

(

plus(InnerVarStp,OuterIdentVarName,CntVarIdent,Class,Method);

minus(InnerVarStp,OuterIdentVarName,CntVarIdent,Class,Method)

)

),

control\_dep(InnerVarStp,WhileStp,\_,Class,Method)

),

IdentVarName = CntVarIdent,

Tipo = polynomial,

asserta( polynomial\_induction\_variable(IdentVarName,OuterIdentVarName,Tipo) ).