Algebraic Inequality Transformation - Pseudo Code

Contact: thanh.truong@it.uu.se

```
function AQIT fixpoint loop
input:  A query pred
output: A transformed predicate if possible,
        otherwise the original pred
begin
    set oldpred = pred
    set newpred = transform_pred(oldpred)
    while  (oldpred is not equal to newpred)
        set oldpred = newpred
        set newpred = transform_pred(oldpred)
    end while
    return newpred
end
```

Listing 1 AQIT FIXPOINT LOOP

```
function transform_pred(pred):
input:    A predicate pred
output:   A transformed predicate if possible,
          otherwise the original pred
begin
    if pred is disjunctive then
        set failure = false
        /*result list of transformed branches*/
        set resl = null
        do /*transform each branch*/
            set b = the first not transformed branch in pred
            set nb = transform_pred(b)/*new branch*/
            if  nb not null then add nb to resl
            else  set failure = true
        until failure or no more branch of pred to try
         if not failure then
              /*return a disjunction from resl*/
            return  orify(resl)
         end if
    else if pred is conjunctive then
        set path = chain(pred)
        if path not null then
            set exposedpath = expose(path)
            if  exposedpath not null then
                return substitute(pred, path, exposedpath)
            end if
        end if
```

```
    end if
    return pred
end
```

Listing 2 TRANSFORM PREDICATE

```
function chain(pred):
input :  A  conjunction of predicate pred
output: An IIP if found, otherwise null
begin
    set orblock = get OR predicate in pred if any
    set andblock = remove orblock from pred
    /*Initialize an IIP and visited list of arcs*/
    set iip =visited = null
    set iv = the first not yet exposed indexed variable in
          andblock sorted decreasingly by selectivities of the indexed attributes
    set p = the first indexed transformable predicate using iv
            in andblock
    add node p and variable iv to iip
    add iv to visited
    set rest = remove p from andblock
    set eiip = extend_partial_iip(iip, visited, rest)

    if eiip not null then
       return eiip
    /* No IIP has found, chain in OR predicates*/
    else if orblock not null then
        set failure = false
        set ldiip = null  /* list of  disjunct iip*/
        set norblock = distribute andblock into each
                          branch of orblock
        do
            set b = not yet tried branch in norblock
            set diip = chain(b)
            if  diip  is null then set failure = true
            else set ldiip = add diip to ldiip
        until  (failure or no more branch to try)
        if not failure then
         /*IIPs found on all branches*/
           return orify(ldiip)
        end if

     end if
```

```
    return null
end
```

Listing 3 Chain

```
function extend_partial_iip (iip, visited, rest):
input : A partial iip, a list of visited variables visited, a list
        of untried predicates rest
output: A complete niip if possible, otherwise null
begin
  get the last node and variable (p, v) from iip.
  set candvars = all not yet visited (candidate) variables
                 of the predicate p.
  do
    pop cv from candvars
    set nvisited = push cv into visited
    set q = get the first transformable predicate in rest
    if q exists then
      set niip = push (q, cv) to iip
      if (q, cv) is not a destination node then
        /* continue to extend*/
        set nrest = remove q from rest
        set niip = extend-partial-iip(niip, nvisited, nrest)
      end if
    end if
  until (niip is complete or no more candidate cv to try)
  if niip is complete then
    return niip
  end if
  return null
end if
```

Listing 4 Extend_partial_iip

```
function substitute (oiip, eiip, pred):
input: An original oiip, an exposed eiip, the predicate pred
output: A transformed query tpred
begin
  set tpred = remove all predicates in oiip from pred
  set tpred = append tpred with all predicates in eiip
  return tpred
end
```

Listing 5 Substitute

```
function expose (ciip):
input:  A complete ciip or a complete disjunction ciip
output: A single exposed IIP or a disjunction of exposed
        IIPs. Otherwise, null.
Begin
  if ciip is a disjunction of IIPs then
    set failure = false
    set lis-exp-iips = null /* list of exposed IIPs */
    do
      set iip = the first not yet tried disjunct of ciip
      set eiip = expose (iip)
      if eiip is null then
        set failure = true
      else
        add eiip to list-exp-iip
      end if
    until (failure or no more disjunct to try)
    if not failure then return orify(list-exp-iips)
  else /* ciip is a single complete IIP */
    if no intermediate nodes in ciip then
      return ciip
    else
      set inode = get the last node from ciip
      set tnode = get the second last node from ciip
      set eiip = remove inode and tnode from ciip
      set success = false
      do
        set R = the first not yet tried rule from
                a set of algebraic-rules
        if test-LHS(R, inode, tnode) then
          /*Create new destination node*
          set desnode =
          apply-RHS(R, inode, tnode, eiip)
          set eiip = add desnode  to eiip
          set success = true
        end if
      until (success or no more rule R to try)
      if success then return expose(eiip)
    end if
  end if
  return null
end
```