General algorithm for the Live-out Iterator Problem

Louis-Noël Pouchet

pouchet@cse.ohio-state.edu

Dept. of Computer Science and Engineering, the Ohio State University

September 2010

888.11

Running example

Example (Input program)

```
for (i = 1; i < N; ++i) {
  S1(i);
  for (j = i; j < M; ++j)
    S2(i,j);
    for (k = j; k < M; ++k)
      S3(i,j);
}
```

Example (PIP output)

```
if (N > 1)
  i = N - 1;
if (N > 1)
  if (M > 1) {
    if (M >= N) {
      i = N - 1;
      j = M - 1;
      k = M - 1;
    }
    if (M < N) {
      i = M - 1;
      j = M - 1;
      k = M - 1;
    }
  }
```

Running example

Example (Input program)

```
for (i = 1; i < N; ++i) {
  S1(i);
  for (j = i; j < M; ++j)
    S2(i,j);
    for (k = j; k < M; ++k)
      S3(i,j);
}
```

Example (Edited PIP output)

```
if (N > 1){
  i = N - 1;
  if (N > 1) {
    if (M > 1) {
      if (M >= N) {
        j = M - 1;
        k = max(j, M);
      }
      if (M < N) {
        j = M - 1;
        k = max(j, M);
      }
    }
  j = max(i, M);
}
i = max(1, N);
```

Proposed approach

1. Create a synthetic program with one statement per loop
   - Remove all existing statements
   - Insert a fake statement at the beginning of each loop body

2. Template structure for a loop l with iterator i:

```
{
  ... code for inner loops of l, if any ...
}
i = max(lowerbound(l), upperbound(l) + 1);
```

3. Compute the lexmax problem for each statement
   - Each leaf gives a case where an inner loop would be executed for the last time
   - If there are inner loops, recursively insert the template:

```
{
  ... values for lexmax of l ...
  {
    ... values for lexmax of l + 1 ...
    k = max(lowerbound(l + 2), upperbound(l + 2) + 1);
  }
  j = max(lowerbound(l + 1), upperbound(l + 1) + 1);
}
i = max(lowerbound(l), upperbound(l) + 1);
```

Algorithm

Input:

- an AST A of a program such that:
  - A represents a Static Control Part
  - Conditionals are always true
  - There is no loop iterator symbol assigned outside its defining loop

Output:

- an AST B containing A which is appended another AST that assigns to each loop iterator in A the value it takes when A is executed

Main algorithm

Algorithm

Algorithm produceLiveOutIteratorValues

Input:

AST: A

Output:

AST: containing A *and the live-out iterator values*

```
B ← createSyntheticProgram(A)
Poly ← extractPolyhedralRepresentation(B)
outAst ← duplicateAST(A)
map ← emptyMapByAddress()
ast ← emptyAST()
outAst.append(createLiveOutIteratorsFromAST(B.root, Poly, map, ast))
return outAST
```

Algorithm `createSyntheticProgram`

Algorithm

Algorithm createSyntheticProgram

Input:

AST: A

Output:

AST: synthetic AST with one statement per loop

```
B ← duplicateAST(A)
forall $n \in nodes(B)$ do
  if nodeType(n) = StatementNode then
    B.deleteNode(n)
  elseif nodeType(n) = ForNode then
    n.getLoopBody().prepend(createDummyStatement())
  end if
end for

return B
```

Algorithm createLiveOutIteratorValues

Algorithm

Algorithm createLiveOutIteratorValues

Input:

AST node: n

Statement[]: Poly

MapByAddress(AST node, AST node): map

AST : main

Output:

AST: containing the live-out iterator values

```
ret ← emptyAST()
if isLeaf(n) then
  stmt ← getStatementFromList(Poly, n)
  S ← extendSystemForLexmax(stmt.domain, stmt.nbIter)
  Q ← computeLexicographicMinimum(S)
  ret.append(convertQuastToFinalAST(Q, stmt, n.getParent()))
  insertMap(map, (n, ret))
else
  for each $c \in successors(n)$ do
    ast ← produceLiveOutIteratorValues(c, Poly, map, main)
    astNode ← getMap(map, c.getParent().firstSucessor())
    if astNode ≠ ast then
      for each StatementNode : s ∈ astNode do
        s.append(duplicateAST(ast))
      end for
    else
      main.append(ast)
    end if
  end for
end if
if isLeaf(n) then
  loop ← n.getParent()
  ast ← createMaxCondition(loop, stmt)
  ret.append(ast)
end if

return ret
```

Algorithm convertQuastToFinalAST

Algorithm

```
Algorithm convertQuastToFinalAST
Input:
QUAST: $Q$
Statement: stmt
AST node: loop
Output:
AST: representing the quast

ast ← emptyAST()
iteratorSymbol ← stmt.iterators[stmt.nbIter - 1]
lowerBound ← ast.createSubstractExpression(getLowerBoundExpression(loop), 1)
ast.append(ast.createAssignment(iteratorSymbol, lowerBound)
ast.append(convertQuastToAST(Q))

return ast
```

Algorithm `createMaxCondition`

Algorithm

Algorithm createMaxCondition
Input:
AST node: $loop$
Statement: stmt
Output:
AST: representing the loop exit value

```
ast ← emptyAST()
iteratorSymbol ← stmt.iterators[stmt.nbIter - 1]
upperBound ← ast.createAddExpression(iteratorSymbol, 1)
maxExpression ← ast.createMaxExpression(getLowerBoundExpression(loop), upperBound)
ast.append(ast.createAssignment(iteratorSymbol, maxExpression))
return ast
```

An example

Example (Input program)

```
for (i = 1; i < N; ++i) {
  S1(i);
  for (j = i; j < N - 1; ++j)
    S2(i,j);
}
```

Example (PIP output for S1)

```
if (N > 1)
  i = N - 1;
```

Example (PIP output for S2)

```
if (N - 1 > 1) {
  i = N - 2;
  j = N - 2;
}
```

Example (Produced output)

```
i = -1;
if (N > 1){
  i = N - 1;
  j = i - 1;
  if (N - 1 > 1)
    j = N - 2;
  j = max(i, j + 1);
}
i = max(1, i + 1);
```