Solving the Live-out Iterator Problem, Part II

Louis-Noël Pouchet

pouchet@cse.ohio-state.edu

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

September 2010

888.11

Algorithm for the restricted case

Algorithm

Algorithm produceLiveOutIteratorValues
Input:
AST: A
Output:
AST: containing A *and the live-out iterator values*

Poly ← *extractPolyhedralRepresentation(A)*
Poly ← *orderInExecutionOrder(Poly)*
outAst ← *duplicateAST(A)*
for *i* ← *1* **to** *Poly.size* **do**
  S ← *extendSystemForLexmax(Poly[i].domain, Poly[i].nbIter)*
  Q ← *computeLexicographicMinimum(S)*
  outAST.append(convertQuastToAST(Q))
end for
return *outAST*

Generalization

Two problems to solve:

- Remove the restriction on lower/upper bound
  - Now, the loop may not execute at all
  - Its last iteration may not be Ub

- Remove the restriction on conditionals
  - Now, the loop may not execute at all
  - Its last iteration depends on the conditional

Remove the restriction on lower/upper bound

Example (Input program)

```
for (i = 1; i < N; ++i)
  S(i,j);
```

Example (PIP output)

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

Example (Desired output)

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

Modification of PIP output

Example (Input program)

```
for (i = 1; i < N; ++i)
  S(i);
```

Example (PIP output)

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

Example (Edited PIP output)

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

Rule of thumb for the modification

- First part of the solution: given by PIP
  - Provides the conditions on parameters for the loop to iterate
  - Actually, provides the conditions for the *statement* to execute
  - In practice, this value is useful only if there is a loop enclosed
- Second part of the solution: syntactic editing
  - the exit value of a loop is simply $max(lb, ub+1)$
  - However, lb and Ub can use values of *iterations* of surrounding loops

Generic approach?

Example (PIP output)

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

Example (Input program)

```
for (i = 1; i < N; ++i)
  for (j = i; j < N - 1; ++j)
    S(i,j);
```

Example (Edited PIP output)

```
if (N - 1 > 1)
  i = N - 2;
  j = N - 2;
  // j executes only when i executes
  j = max(i, N - 1);
}
i = max(1, N);
```

What about the value of j when $N = 2$?

Maybe close...

Example (Input program)

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

Example (PIP output)

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

Example (Edited PIP output)

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

Scheme for the previous example

1. The exit value of the i loop is the maximum of its lower and upper bound
2. Compute the lexmax for the first statement (ie, the first loop)
3. This lexmax is the last executed instance of the second loop, that is, the value i takes when the j loop is executed for the last time
4. The exit value of the j loop is the maximum of its lower and upper bound, when i reaches its lexmax
5. We don't need the lexmax of $S2$: the "iteration domain" of the j loop is the same as $S1$
6. The lexmax of $S2$ is needed only if some loop is enclosed by the j loop
7. But for the input to be a syntactically correct program, we need $S2$...

A more complex 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;
    }
  }
```

A more complex 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);
```

Exercise 1

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

Exercise: write an algorithm which implements the above description

Exercise 2

Input:

- an AST A of a program such that:
  - A represents a Static Control Part
  - 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

Exercise: write an algorithm which implements the above description