Solving the Live-out Iterator Problem, Part I

Louis-Noël Pouchet
pouchet@cse.ohio-state.edu

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

September 2010

888.11
Reminder: step-by-step methodology

1. **Problem definition**: Understand and define the problem
2. **Examples**: Find various examples, and compute the desired output by hand
   → **Restriction**: Find an algorithm, maybe restricted to simpler cases
3. **Generalization**: Generalize the algorithm to work on all cases
4. **Proof**: Prove the algorithm is complete and correct
5. **Complexity**: Study the complexity of the algorithm
Outline for today

- Find a useful restriction of the problem
  - Typically, add extra properties on the input
  - And/or remove some properties on the output

- Build and solve the problem for it
  - Maximal reuse of existing solutions
  - Keep in mind the general problem
Summary of the problems

List the problems to solve

- Multiple statements
  - Which loop executes last?
- Min/max expressions
  - The value depends on the expressions
  - Need to substitute surrounding iterators with the last value for which the loop test is true, not necessarily the exit value of the loop iterator
- Conditionals
  - a loop may not execute, how to determine its last execution?
- Parametric loop bounds
  - The loop may not execute at all!
  - What is the value to use in the substitution? The exit value?
- Loop iterator symbols being assigned after the loop execution
  - How to compute the exit value in this case?
Another view: Solution-driven

Order the problems starting with the simplest solution

1. Start from the set of programs with:
   - no conditional,
   - no min/max,
   - no parameter,
   - no iterator symbol assigned in the loop body,
   - a single statement

2. Adding multiple statement support

3. Adding parameters

4. Adding conditionals

5. Adding min/max

6. Adding iterator symbol assigned in the loop body
A useful restriction of the problem

- What if a loop always iterates at least once?
  - Property: $lb \leq Ub$
  - The exit value is the last value for which the test is true + 1
  - Impact on conditionals, min/max, iterator assigned in body?

- What if a conditional is always true?
  - Property: the conditional is an affine form of the parameters only

- Under these assumptions, what about min/max expressions?
Overview of the approach

1. Find a good, general algorithm for our restricted case

2. Modify it to generalize to:
   - arbitrary conditionals
   - arbitrary loop bounds

3. Modify the input specification to cover only programs where iterator symbols are never assigned outside the loop
Reminder: algorithm writing 101

1. Determine the input and output
2. Find a correct data structure to represent the problem
   ▶ Don’t hesitate to convert the input to a suitable form, and to preprocess it
3. Try to reduce your problem to a variation of a well-known one
   ▶ Sorting? Path discovery/reachability? etc.
   ▶ Look in the litterature if a solution to this problem exists
4. Decide wheter you look for a recursive algorithm or an imperative one, or a mix
   ▶ Depends on how you think, how easy it is to exhibit invariants, what is the decomposition in sub-problems, ...
5. Write the algorithm :-)
6. Run all your examples on it, manually, before trying to prove it
Determine the input and output

Input:
- an AST $A$ of a program such that:
  - $A$ represents a Static Control Part
  - For each loop in $A$, the lower bound is always smaller than the upper bound
  - 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
Find a good representation for the problem

Example

```c
for (i = 0; i <= N; ++i)
    for (j = 0; j <= min(M, 4 * i - 4); ++j)
        if (3 * j >= -2 * i + 8)
            S(i, j);
```

![Diagram](image-url)
Polyhedral representation

- Model iteration domains using inequalities
  - inequalities for lower bounds, upper bounds, conditionals
  - min/max simply produces multiple inequalities
  - Warning: only executed instances are part of the iteration domain

- Using this representation, what is the geometric intuition of the exit value of iterators?
  - It is simply the lexicographic maximum of the iteration domain + 1!
  - Can we reuse existing algorithms to compute the lexicographic maximum of the iteration domain?
Reducing to a variation of a well-known problem

**PIP: Parametric Integer Programming [Fea88]**

In a nutshell:

- **PIP input:** A system of inequalities defining a parametric polyhedron

  \[
  \begin{align*}
  i & \geq 0 \\
  i & \leq N \\
  j & \geq 0 \\
  j & \leq M \\
  j & \leq 4 \times i - 4 \\
  3 \times j & \geq -2 \times i + 8
  \end{align*}
  \]

  *Example:*

  \[
  \begin{align*}
  i & \geq 0 \\
  i & \leq N \\
  j & \geq 0 \\
  j & \leq M \\
  j & \leq 4 \times i - 4 \\
  3 \times j & \geq -2 \times i + 8
  \end{align*}
  \]

- **PIP output:** the lexicographic **minimum** of the system

  *Example:*

  ```
  if (7*n >= 10) {
      if (7*m >= 12) {
          (i = 2, j = 2)
      } else {
          (i = -m-(m div 2)+4, j = m)
      }
  }
  ```
Problems to solve

▶ PIP outputs the lexicographic minimum, we want the maximum
  ▶ Simple: $max(x) = min(-x)$
  ▶ Need to insert variables $x' = -x$, $y' = -y$, etc. as the first variables of the system, and compute the lexmin of the new system

▶ PIP does not produce an AST explicitly, it uses its internal representation
  ▶ Need to convert PIPLib internal representation into an AST
  ▶ Need to dig into PIPLib documentation, should not be difficult
On the road to write the algorithm

In a nutshell:

1. Convert the AST into its polyhedral representation

2. For a given statement, create the PIP problem for the lexmax

3. Convert the solution to the system into an AST
Data structures [1/2]

Polyhedral representation:

- It is a array of elements of type `Statement`
- A `Statement` is a structure containing:
  - `Matrix : domain`, for the iteration domain, using the same representation as PIP input
  - `Matrix : schedule`, for the schedule
  - `integer : nbIter`, for the number of loops surrounding the statement
  - (and more, but not useful here)

- Available functions:
  - `Statement[] : extractPolyhedralRepresentation(AST : A)`
  - `Statement[] : orderInExecutionOrder(Statement[] : statementarray)`
Data structures [2/2]

PIP / PIPLib:

- PIPLib uses as an input a *Matrix*
- Calling PIPLib outputs a *QUAST* (quasi-affine solution tree)
  - It is a tree where the leaves are all possible values for the lexicographic minimum of the input system, the other nodes are conditions on parameters

- Available functions:
  - *QUAST* : `computeLexicographicMinimum(Matrix : system)`
  - *AST* : `convertQuastToAST(QUAST : solution)`
Exercise

Input:
- an AST $A$ of a program such that:
  - $A$ represents a Static Control Part
  - For each loop in $A$, the lower bound is always smaller than the upper bound
  - 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
Algorithm to create a Lexmax system

Algorithm extendSystemForLexmax

Input:
Matrix: A, in PIPLib format
integer: nbVars

Output:
Matrix: in PIPLib format, with extra columns and equalities such that lexmin(B) = lexmax(A) for the nbVars first variables

\[ B \leftarrow \text{duplicateMatrix}(A) \]

\[ \text{for } i \leftarrow 1 \text{ to } \text{nbVars do} \]
\[ \quad B \leftarrow \text{insertColumnAtPosition}(B, 1) \]
\[ \text{end for} \]

\[ \text{for } i \leftarrow 1 \text{ to } \text{nbVars do} \]
\[ \quad B \leftarrow \text{insertRowAtPosition}(B, B.\text{NbRows}) \]
\[ \quad B[B.\text{NbRows} - 1][i] \leftarrow -1 \]
\[ \quad B[B.\text{NbRows} - 1][i + \text{nbVars}] \leftarrow 1 \]
\[ \text{end for} \]

return \( B \)