Bounding Loop Iterations for Timing Analysis

Christopher A. Healy
Mikael Sjödin
Viresh Rustagi
David Whalley

Motivation: The Problem

- Accurate timing analysis requires knowing minimum and maximum bounds on loop iterations.
- Current timing analysis techniques require user to enter this information.

  — tedious, error prone

  — code generation strategies

Motivation: Our Goals

- Less Work for the User
- More Reliable Bounds on Loop Iterations
- Tighter Bounds on BCET / WCET
- Continue to Give Fast Response to User

Steps to Bound Number of Iterations

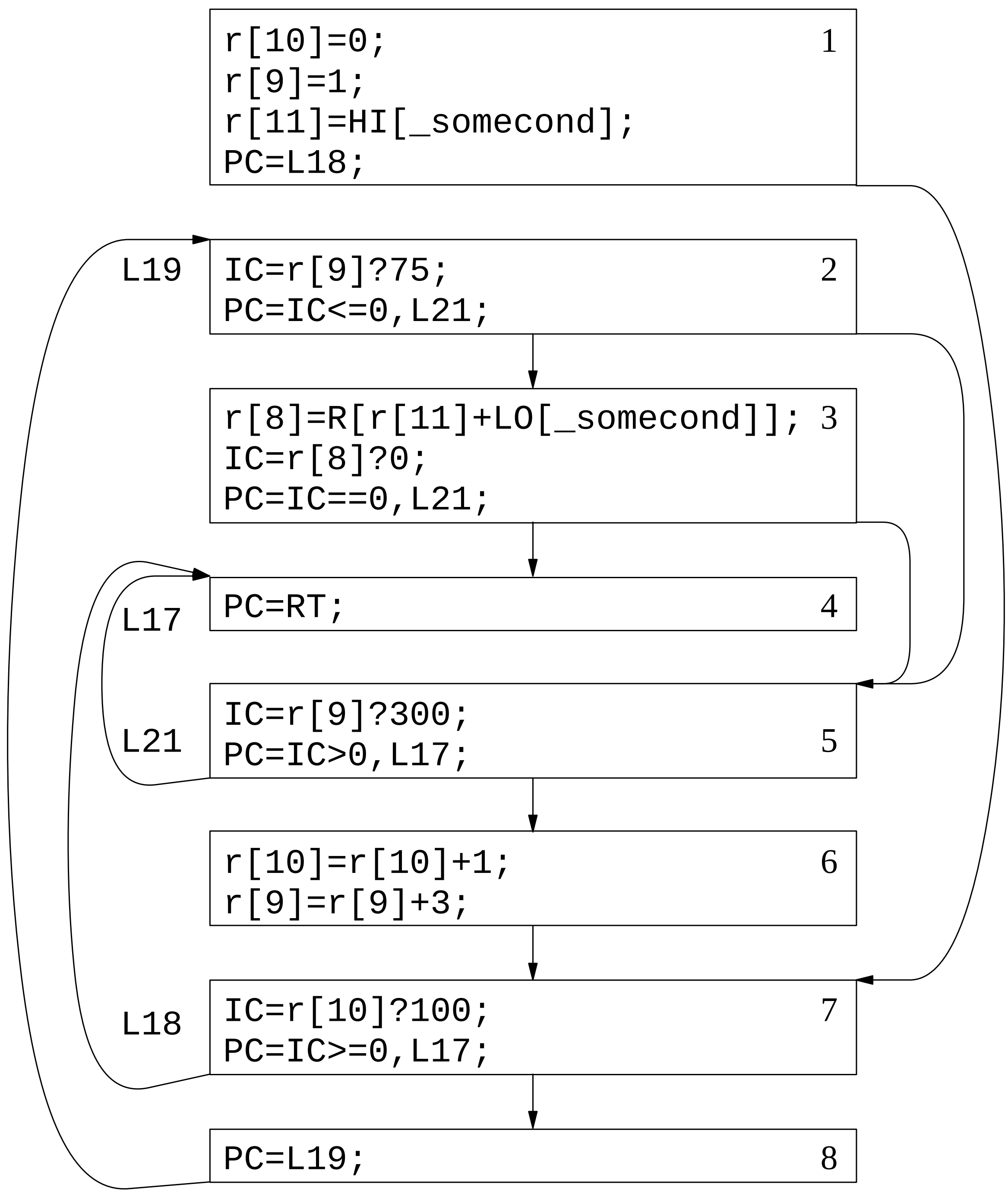
- Identify Iteration Branches
- Calculate When Each Iteration Branch Changes Direction
- Determine Range of Iterations When These Branches Can Be Reached
- Calculate Maximum and Minimum Number of Iterations

Identifying Iteration Branches

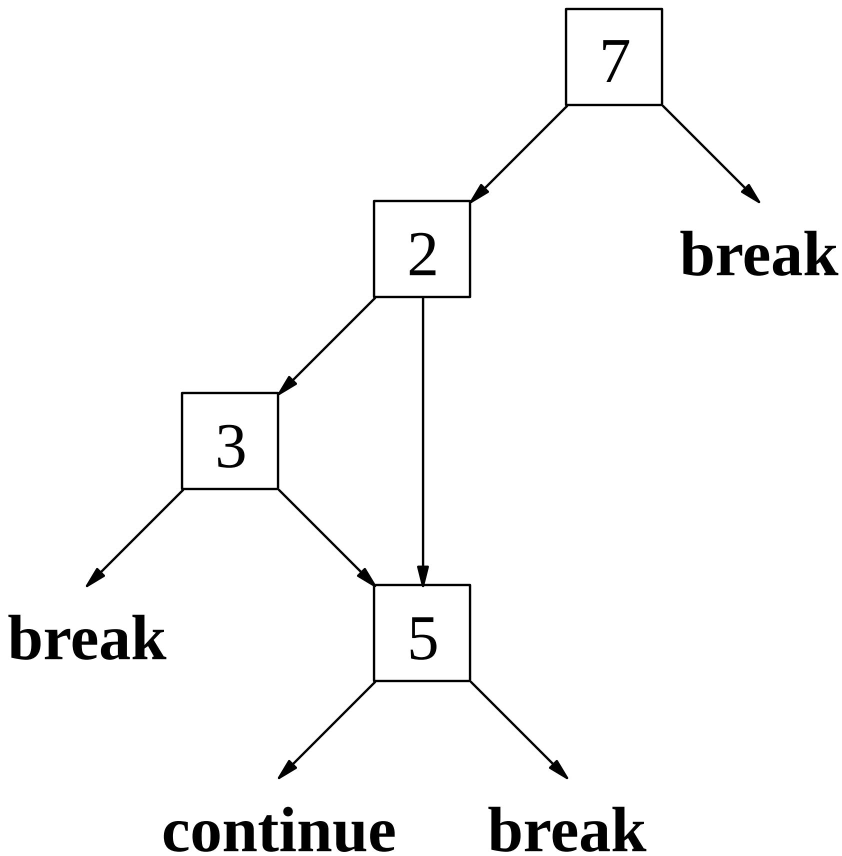
These are branches that:

- have a transition to a basic block outside the loop, *or*
- have a transition to the header, or to a block that is always followed by the header, *or*
- can conditionally reach blocks containing different iteration branches

SPARC Instructions for Example Loop



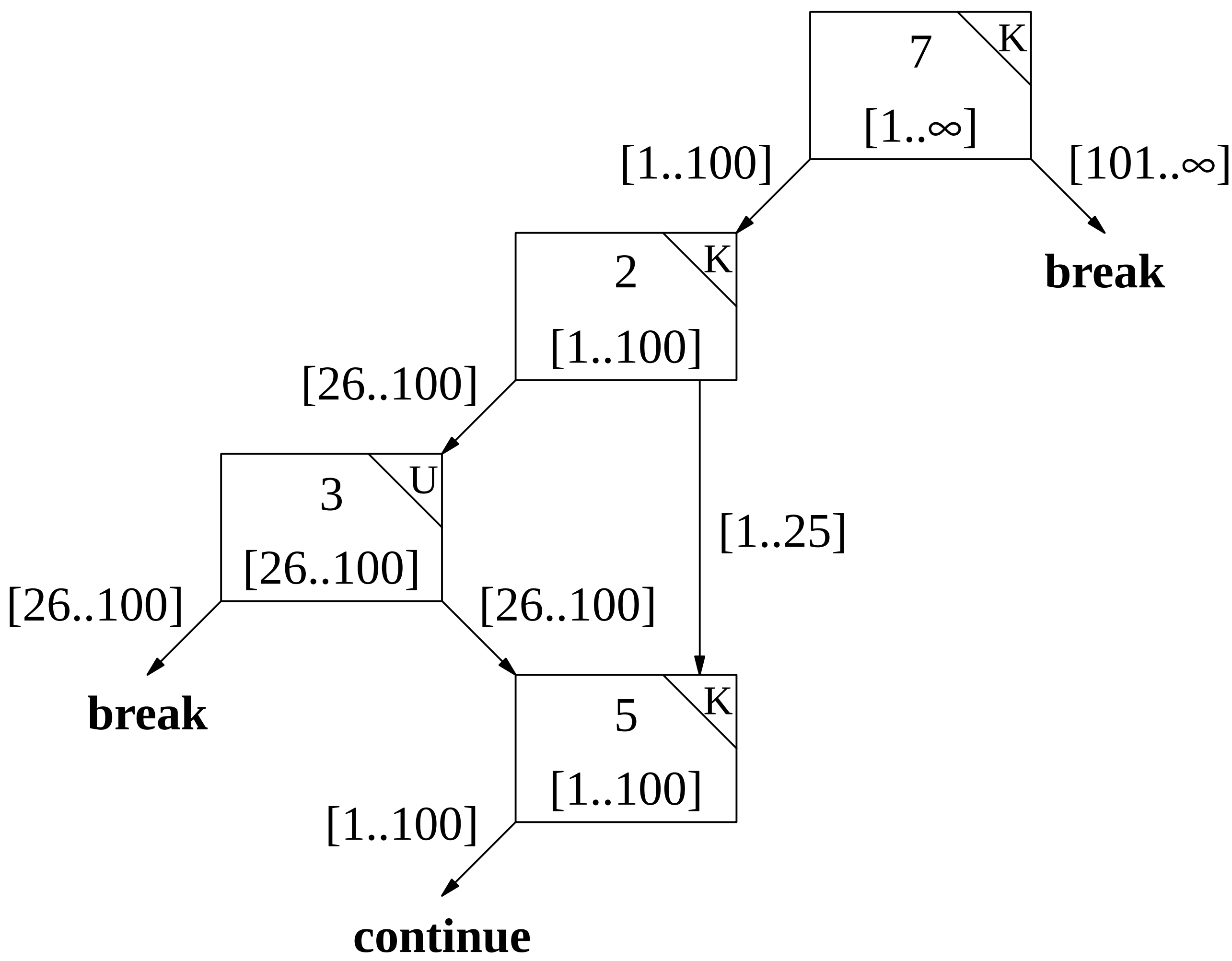
Precedence Relationship Between Iteration Branches



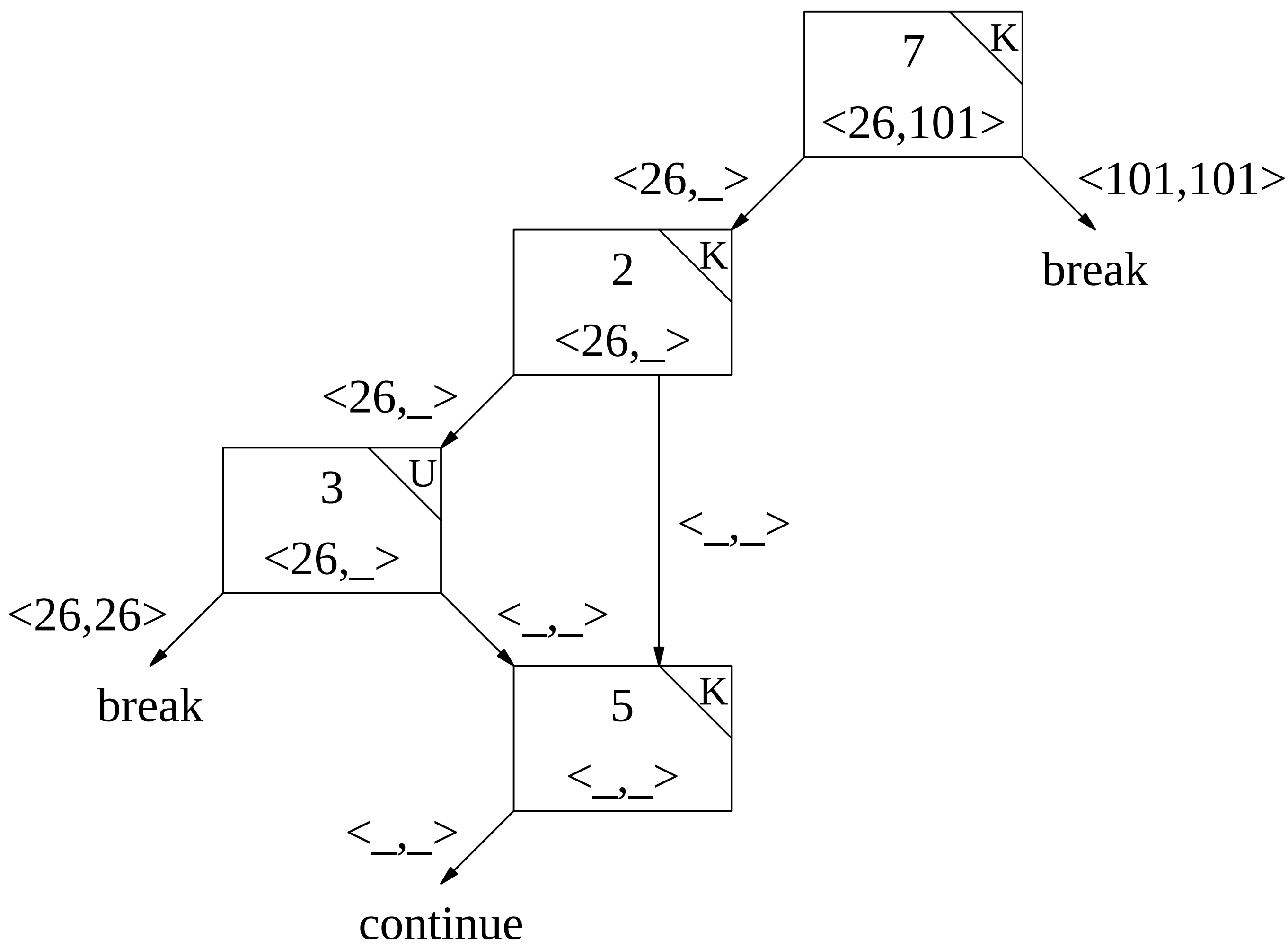
Determining Iterations When Iteration Branches Can Be Reached

- Associate a range of iterations to each node and edge in the DAG.
- Nodes
  - — head of DAG: range $[1..\infty]$
  - — other nodes: range is union of ranges of incoming edges
- Edges from an arbitrary node *i*
  - — If branch *i* is *known*, edges correspond to $[1..N_i - 1]$ and $[N_i..\infty]$. Intersect each edge range with node *i*'s range.
  - — If branch *i* is *unknown*, both outgoing edges are assigned same range as *i*.

DAG of Branches with Ranges of Iterations



DAG of Iteration Branches with Maximum and Minimum Iteration Values



Example Loop

```
int sumarray(a, m)
int a[], m;
{
   int i, sum;
   extern int n;

   sum = 0;
   valuebnd m[10:100] n[20:80]
   for (i = 1; i < m+n; i++)
      sum += a[i];
   return sum;
}
```

$$N = \left\lfloor \frac{limit - (initial + before)}{before + after} \right\rfloor + adjust + 1$$

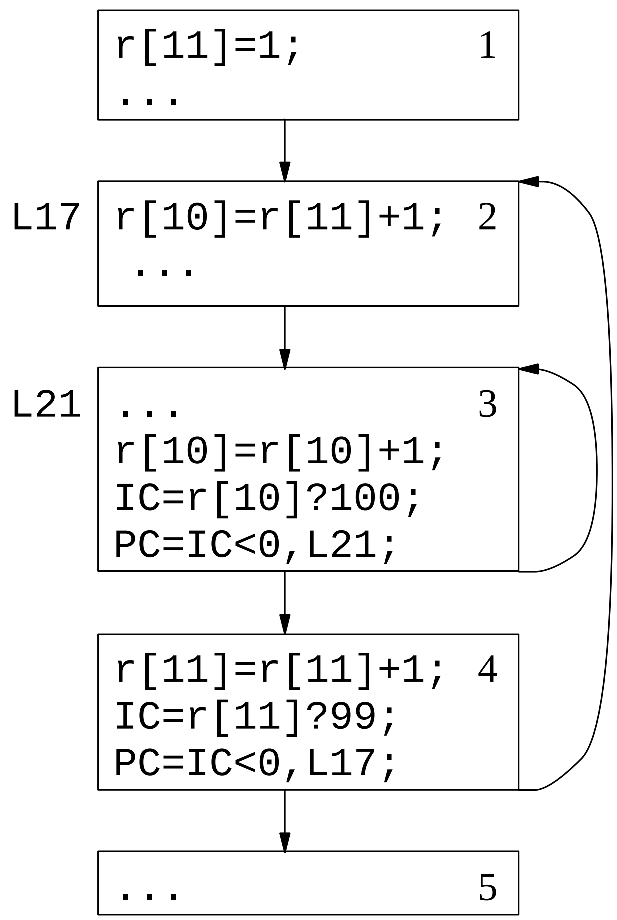
$$= \left\lfloor \frac{m + n - (1 + 1)}{1 + 0} \right\rfloor + 0 + 1$$

$$= m + n - 1$$

Example of Nested Loop

```
...
for (i = 1; i < 99; i++)
   for (j = i+1; j < 100;
        j++)
      ...
...
```

(a) Source Code



(b) Corresponding SPARC Instructions

Number of Iterations

— f is the number of iterations of the inner loop on the first iteration of the outer loop,

— d is the difference in the number of inner loop iterations for each successive iteration of the outer loop (note that d may be negative), and

— n is the number of times that the outer loop iterates

$$N_{avg}(f, d, n) = \frac{f + f + (n-1)d}{2}$$

In our example, we obtain the average number of iterations to be:

$$N_{avg}(f, d, n) = \frac{98 + 98 + (98-1)(-1)}{2} = 49.5$$

Incorporating Number of Iterations

- Best Case
  - Absolute minimum number of iterations (1) is used to compute BCET of the inner loop.
  - Average number of iterations (49.5 ---> 49) is used for computing BCET in context of ancestor nodes in the timing tree.
- Worst Case
  - Absolute maximum number of iterations (99) is used to compute WCET of the inner loop.
  - Average number of iterations (49.5 ---> 50) is used for computing WCET in context of ancestor nodes in the timing tree.

Future Work

- Nonconstant Increment or Decrement of Counter Variable
- Multiple Nested Loops Depending on Outer Counter Variables
- Detect Infeasible Paths

Conclusion

- Bounding Number of Loop Iterations
  - Loops with Multiple Exits
  - Nonconstant, Loop-Invariant Number of Iterations
  - Dependence on Outer Loop Counter Variable
- Benefits
  - Less Work for the User
  - More Reliable Predictions
  - Tighter Bounds on WCET and BCET