Verification of Real-Time Systems

Loop Bound Analysis

Jan Reineke

Advanced Lecture, Summer 2015

Applications of Numerical Domains

As input to other analyses:

- Cache Analysis
- To detect dependencies between memory accesses in pipeline
- Loop Bound Analysis

State of the Art in Loop Bound Analysis

Multiple approaches of varying sophistication

- Pattern-based approach
- Data-flow based approach
- Slicing + Value Analysis + Invariant Analysis
- Reduction to Value Analysis

Loop Bound Analysis: Pattern-based Approach

Identify common loop patterns; derive loop bounds for pattern once manually

Initial value of x?

```
while (x < 6)
{
    …
    x++;
}
```

for (x=0; x<10; x++)

No modification of x.

→ Loop bound: 6 minus minimal value of x

Loop Bound Analysis: Data-flow-based Approach [Cullmann and Martin, 2007]

Combination of multiple analyses:

1. Identify possible loop counters
2. “Change analysis”: determine how loop counters may change in one loop iteration
3. Bound calculation: combine results from step 2 with branch conditions

Loop Bound Analysis: Data-flow-based Approach [Cullmann and Martin, WCET 2007]

Example:

```
for (x < 6) {
    y++;
    if (y % 2 = 0)
      x++;
    else
      x = x+2;
    z++;
}
```

1. x, y, and z are potential loop counters

2. Change analysis:
x'-x in [1,2]
y'-y in [1,1]
z'-z in [1,1]

3. Loop bound:
6 assuming x >= 0 initially

Slicing + Value Analysis + Invariant Analysis [Ermedahl et al., WCET 2007]

Combination of multiple analyses:

1. Slicing: eliminate code that is irrelevant for loop termination
2. Value analysis: determine possible values of all variables in slice
3. Invariant analysis: determine variables that do not change during loop execution
4. Loop bound = set of possible valuations of non-invariant variables

Program slicing *is the computation of the set of programs statements, the program slice, that may affect the values at some point of interest, referred to as a* ***slicing criterion****.*

Slicing + Value Analysis + Invariant Analysis [Ermedahl et al., WCET 2007]

Step 1: Slicing with slicing criterion (i <= INPUT)

```
int OUTPUT = 0;
int i = 1;
while (i <= INPUT) {
      OUTPUT += 2;
      i += 2;
}
```

```
int i = 1;
while (i <= INPUT) {
      i += 2;
}
```

Slicing + Value Analysis + Invariant Analysis [Ermedahl et al., WCET 2007]

Step 2: Value Analysis

Observation:

If the loop terminates, the program can only be in any particular state once.

→ Determine number of states the program can be in at the loop header.

```
int i = 1;
while (i <= INPUT) {
      i += 2;
}
```

Value Analysis:

INPUT in [10, 20] (assumption)

i in [1, 20], i % 2 = 1

19 → REDUCTION

→ *11 * 10 states*

→ *Loop bound 110!*

Slicing + Value Analysis + Invariant Analysis [Ermedahl et al., WCET 2007]

Step 3: Invariant Analysis

Observation:

Value of INPUT is not completely known, but INPUT does not change during loop.

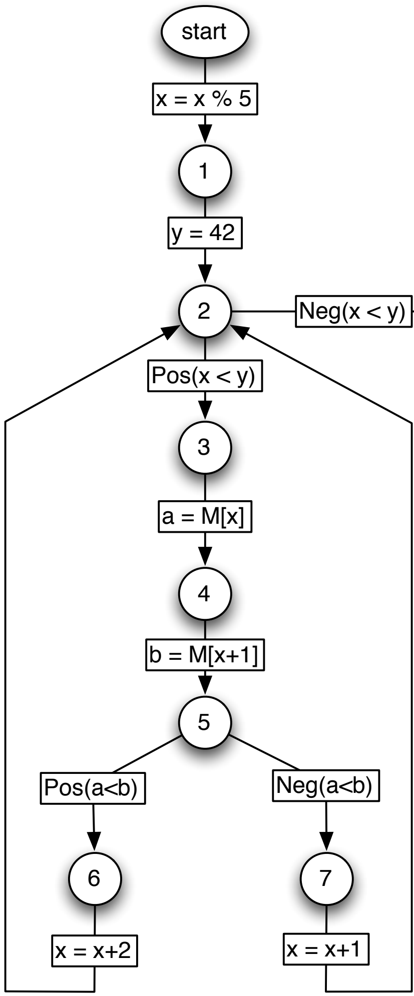
→ Determine variables that are invariant during loop.

```
int i = 1;
while (i <= INPUT) {
      i += 2;
}
```

Value Analysis:
INPUT in [10, 20] (assumption)
i in [1, 20] , i % 2 = 1

→ INPUT is invariant!
→ Loop bound 10!

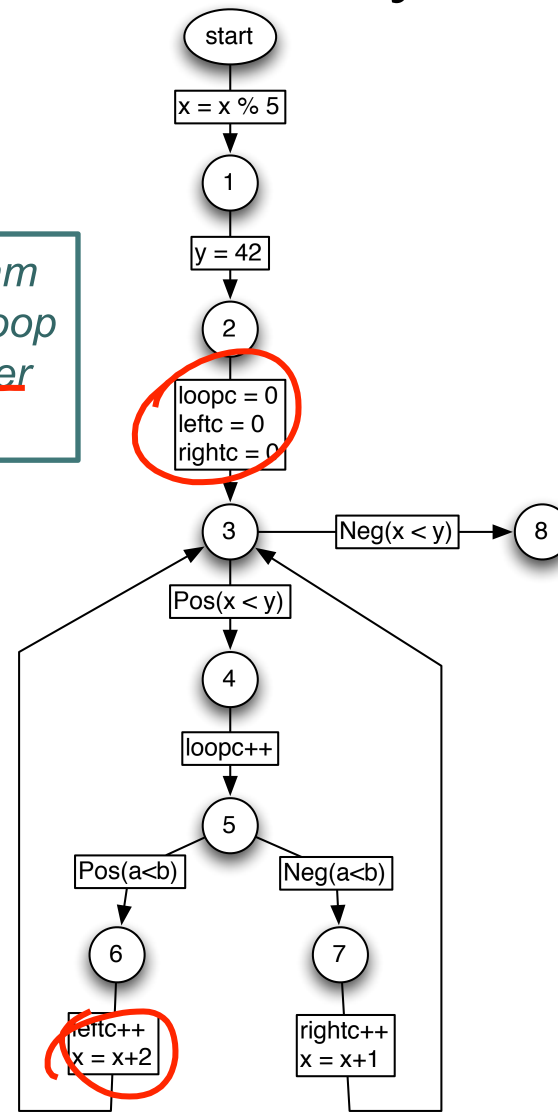
Reduction:
Loop Bound Analysis to Value Analysis



Instrument program with counters of loop iterations and other interesting events

Upper bound for loopc is loop bound!

Requires very powerful relational analysis…



Summary

Loop Bound Analysis Approaches

- Pattern-based approach
- Data-flow based approach
- Slicing + Value Analysis + Invariant Analysis
- Reduction to Value Analysis

Outlook

- Path Analysis
- Cache Abstractions
- Schedulability Analysis
- Cache-Related Preemption Delay
- Predictable Microarchitectures