Alessandro Temil

CMPS 203 Course Project

Automated WCET estimation for a simple imperative programming language
When is it important to know the worst case execution time?

- In the Real Time domain, knowing the time bound for a function call is a key information to know whether the system will meet its deadlines.
- This problem is undecidable for the general case.
- The solution strongly depends on the given input.
- Loop bounds are difficult to predict.
The current approach

- For long time, the programmer was required to annotate the code with loop bounds
- Today's commercial grade tools are based on flow analysis abstract interpretation.
  - The code is internally represented as a graph, with a node for each conditional instruction
  - Various inferences are done on the values taken by the identifiers, to predict if a path will be taken or not, and if taken, how many times
What I've done

- The language I've been working on is a tiny subset of C.
- It has function calls, no pointers, no side effects, and the only iterating instruction is a for loop with a special syntax:
  - For (x; 1; x<10; x+1)
    - The first field tells the variable the loop is controlled by
    - The second is the value the variable is initialized to
    - The third expression is the continuation condition
    - The forth is the value the variable gets after each iteration
- I've taken a simple approach: The analysis function looks for some known patterns in the for loops and if one is found, a precise upper bound is given as output
- Otherwise it outputs a conservative “Unknown” upper bound
Properties

- It allows complexity by composition: a set of simple calculable routines can be combined in a program that maintains the calculability property.
- It's conservative: If for a program is given an upper bound, that's guaranteed to hold.
  - This is not true for some of the other approaches.
<table>
<thead>
<tr>
<th>Functions with finite upper bound</th>
<th>functions without finite upper bound</th>
</tr>
</thead>
<tbody>
<tr>
<td>Not empty</td>
<td>empty</td>
</tr>
<tr>
<td>&quot;Unknown&quot; is returned</td>
<td>Not empty</td>
</tr>
<tr>
<td>&quot;Unknown&quot; is returned</td>
<td>Not empty</td>
</tr>
</tbody>
</table>
Some examples of the calculation rules for simple constructs

\[
\begin{align*}
& WCET(\text{Var } x): \text{Klookup} \\
& WCET(\text{Integer } n): \text{Kint} \\
& WCET(\text{Boolean } b): \text{Kbool} \\
& WCET(\text{E1}, WCET(\text{E2})) \\
& WCET(\text{E1 PLUS E2}): WCET(\text{E1}) + WCET(\text{E2}) + \text{Kplus} \\
& WCET(\text{if BE then C1 else C2}) \\
& WCET(\text{FOR Var } x, E1, BE1, E2): \text{?}
\end{align*}
\]

This is easy.. but what about the looping construct?

\[
\text{(this makes use of pattern matching..)}
\]
One example of pattern matched FOR construct

\[
\begin{align*}
\text{(this makes use of pattern matching..)} \\
\text{(WCET(FOR(Var\,x,\,E1,\,BE1,\,E2,\,STMTLIST))::?)}
\end{align*}
\]

IF ( Var “x” is local AND 
    E1 is Integer(n1) AND 
    B1 is Var(“x”) LT Integer(n2) AND 
    E2 is Var(“x”) PLUS Integer(n3) AND 
    WCET(STMTLIST) != Unknown AND 
    Var(“X”) does not appear as an LVALUE in STMTLIST)
=> ((n2 – n1) / n3) x WCET(STMTLIST)
Conclusion

- WCET analysis can be done in this way on function operating on fixed size data sets
- Not good for programs whose running time is dependent on the input data
  - It would only be possible to output a value as a function of the input.