A New Timing Schema for WCET Analysis

Stefan M. Petters Adam Betts Guillem Bernat
Department of Computer Science
University of York
United Kingdom
Stefan.Petters@cs.york.ac.uk

Abstract

The timing schemas proposed in various approaches for Worst Case Execution Time (WCET) estimation lack the ability of handling more path-based low level analysis and non-structured code. As it is extremely efficient in the computational stage, we propose a technique to handle these cases, while still retaining most of the efficiency of the syntax tree timing schema. This is achieved by changing the rules for the construction of the computational tree. As a result, the analysis becomes more path aware without affecting the safety of the approach.

1 Motivation

Tree-based calculation methods for WCET analysis were first proposed by Park and Shaw [1], based on a syntax tree representation of the program. A *timing schema* is attributed to certain high-level language constructs, which is essentially a formula for computing the upper bound of their execution time. Bernat et al. [2, 3] extended this approach to incorporate Execution Time Profiles (ETP) instead of integer values. Obtaining ETPs requires a tracing mechanism whereby the data are collected, but there are some related drawbacks. Some tracing mechanisms (e.g. via the NEXUS interface cf. [4]) do not provide compatible traces, so a path-based approach would be much more suitable, but it must cope with computational complexity.

Alternatively, instrumentation points (ipoints) can be manually or automatically placed into the code to generate a trace. There is generally no restriction on where ipoints can be placed in the code, thus basic blocks could have several ipoints, whilst others have none at all. Here we consider that each basic block contains one ipoint at most, without loss of generality. Nonetheless, this still implies that inaccuate placement exists. Inaccuracy in this context means an ipoint that is not placed at the very beginning of a basic block. If this happens for the first basic block of alternative paths, unnecessary overestimations occur.

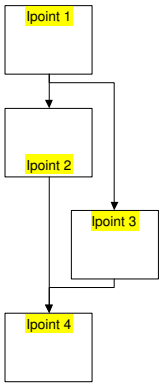

Figure 1. Ipoint Placement Example

For illustration purposes, the assumption of only the longest observed execution time being used for computation is made. Figure 1 provides a simple example of an `if-then-else` code. With the previous approach the block containing *ipoint 1* has two execution times, with only the longer being used. Additionally the block containing *ipoint 2* appears shorter and as a result is less likely to

contribute to the overall WCET in the computational stage. The assertion of potential overestimations applies to both probability distributions and integer values. Furthermore, unstructured code needs better support, as provided in the previous approaches. Unstructured code may arise from a couple of sources, ranging from deliberately or automatically set gotos in mission critical software (cf. e.g. [5]), multiple loop control conditions (e.g. ”if within range”), compiler optimisations or Ada exceptions.

2 Program Representation

To solve the problems described in the previous section an elemental change to the presentation is proposed. In a first move, the unit of computation is no longer basic blocks, but rather the transition from one ipoint to another ipoint. This corresponds to moving the weight of computation from the nodes of a control flow graph to the edges. As such the program representation and the timing schema are changed. In relation to the *extended syntax tree* which is used in [2] to generate the computation formulas, a computational tree is produced, which will be named CTR throughout this paper. Similar to the previous approach four constructs have to be considered.

2.1 Node

A leaf node in the CTR represent a transition from one ipoint to another ipoint. The content may either be an integer number or an ETP.

2.2 Sequence

A sequence of nodes is combined the same way as with the previous approaches. Dependent on the basic setup some sort of convolution or a simple addition may be used.

2.3 Alternatives

The representation of an alternative path now contains the transition from an external ipoint into the alternative path and the transition out of the alternative path to an external ipoint. As a result, a set of alternatives has to share the same starting and the same finishing ipoint (cf. Fig. 2 and Fig. 3).

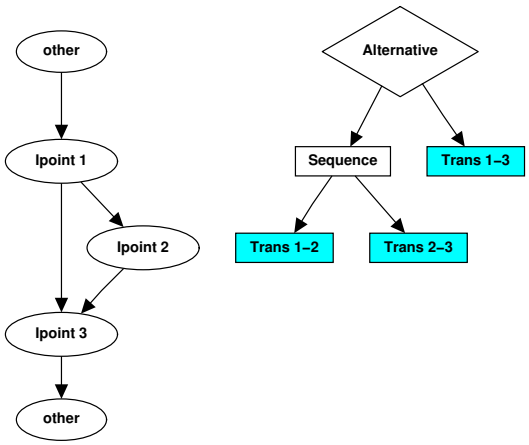

Figure 2. Optional Code

In the case of a simple `if-then` construct omitting the `else` part as depicted in Figure 2, the alternatives contain a single transition and a sequence of transitions respectively. An `if-then-else` construct has sequences of transitions in both alternatives. The two or more alternative parts – more alternatives may be the result of switch statements – are evaluated using the `max` operator as defined in the respective approaches.

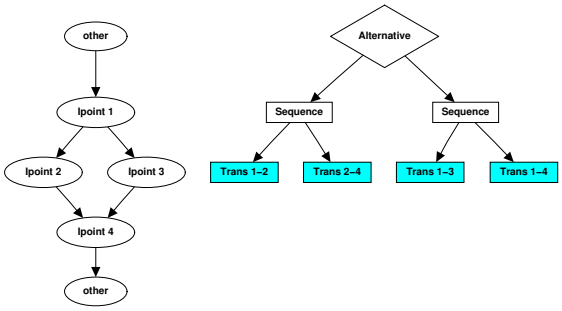

Figure 3. Alternative

2.4 Loop

A loop in this context consists of three parts. A loop entry, a loop iteration and a loop exit. Figure 4 provides an example of a loop and its representation. The basic requirement of these three blocks is that the end ipoint of the entry, the start ipoint of the exit and the start and end ipoint of the iteration have to be identical. The simple case of a loop is when the loop head contains an ipoint and this ipoint is part of every path leading into the loop, out of the loop and during any loop iteration. The loop head is defined as any code from the start of the loop to the point where an exit condition is met. In the computational stage, a loop is treated as a sequence of the entry node, N iterations of the iteration node and the exit node. For the iteration node, the previously proposed unrolling of iterations may take place.

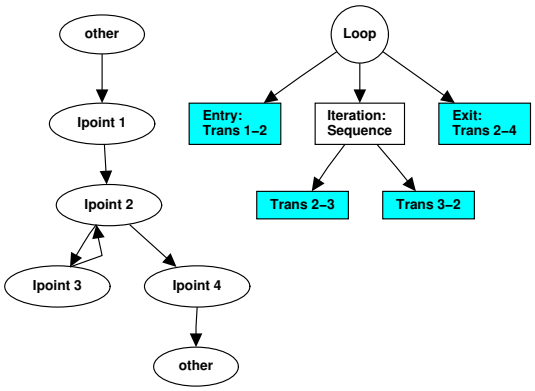

Figure 4. Loop

In the case of the loop head having no ipoint (which fulfils the the criteria in being part of any path as described above) an ipoint of the body that has to be part of any iteration may be chosen as the common ipoint. As the combination of the entry and the exit node contains already one iteration of the loop, two measures are necessary. On one hand the number of iterations N has to be adjusted for the computational stage. On the other hand, an alternative to the loop has to be created, which represents the fact that the loop may not iterate at all. Figure 5 provides the CTR for a graph similar to the one in Figure 4, where *Ipoint 2* is assumed to be in the loop body. However, this transformation will usually be straightforward, as the transition between *Ipoint 1* and *Ipoint 4* should already be visible in the control flow or ipoint graph.

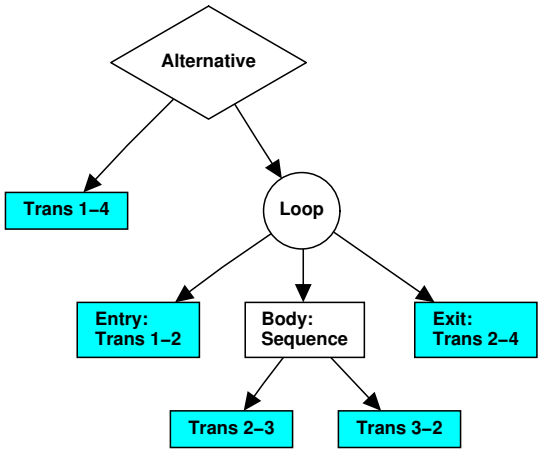

Figure 5. Loop with no Ipoint in Head

2.5 The Timing Schema

The timing schema provided uses the abstract operators $\otimes$ and $\odot$. In the calculation, these revert to a simple addition and multiplication in case of integer numbers, and a convolution and power operator in ETP-based approaches.

- $C_{i,j}$ describes the WCET of the transition from ipoint i to ipoint j which may be a measured leaf node or a
- Sequence (i, j, k): $C_{i,k} = C_{i,j} \otimes C_{j,k}$
- Alternative (i and j outside the alternative and $C_{i,j}^{1..o}$) representing the o alternative paths):
  $C_{i,j} = \max(C_{i,j}^{1}, C_{j,l}^{2}, \ldots, C_{i,k}^{o})$
- Loop (i and k outside then loop and j inside the loop):
  $C_{i,k} = C_{i,j} \otimes (C_{j,j} \odot n) \otimes C_{j,k}$

Within the loop expression n is equal to the number of loop iterations if the common point j lies within the loop head, or number of iterations minus one if the common point j lies within the loop body.

3 Discussion

The actual schema is not provided in this paper in terms of mathematical equations due to space restrictions, but the schema may be derived straight forward from the tree representation. The question arises, how the proposed approach supports the different aspects described in the motivation. The Nexus interface is supported by the loose definition of the contents of the alternatives. As only start and end ipoints are defined, the rigid construction rules are resolved. An enforced ipoint, by introducing an additional branch instruction, where deemed necessary, is comparably straight forward. The overhead of "inaccurately" set ipoints is removed by the transitional description of the nodes. This results overall in a tighter bound on the real values. Possibly most striking is the support of some non-structured code. Especially loops with more than one exit condition are a common problem. This is resolved as the entry and exit nodes of a loop may contain several paths leading to or from the common node. Code like exceptions may be expressed, but may lead to prohibitive execution times of the analysis, as the analysis moves towards a fully path-based approach with every exception considered.

The freedom of expression is possibly the biggest drawback of the approach. On the one hand, it is hard to "read" the CTR and associate its nodes with real code constructs, which is much more straight forward with a syntax tree representation. On the other hand there is more than one CTR solution for almost any real world program. In the extreme, a fully path-based CTR is possible with a set of alternatives at the top level, each representing a possible walk on the control flow graph. The move from a computationally feasible to a computationally infeasible analysis is easily done.

4 Conclusion

In this paper we have proposed a powerful and flexble program representation and a timing schema, which may be applied to any timing schema based approach. The only requirement is the concept of transitional computational cost being associated with the leaf nodes of the CTR. This schema may be applied independently of the low level analysis used to derive the values, without loss of safety of the results, while supporting tighter WCET bounds. Additionally it works equally for integer values or ETP-based approaches.

Future work in this area should focus on experiments to establish the actual gain, which may be expected by using this schema compared to the previously proposed schemas.

References

[1] C. Park and A. Shaw, "Experiments with a program timing tool based on source–level timing schema," *IEEE Transactions on Computers*, vol. 24, pp. 48–57, May 1991.

[2] G. Bernat, A. Colin, and S. M. Petters, "WCET analysis of probabilistic hard real–time systems," in *Proceedings of the 23rd Real-Time Systems Symposium RTSS 2002*, (Austin, Texas, USA), pp. 279–288, Dec. 3–5 2002.

[3] G. Bernat, A. Colin, and S. M. Petters, "pWCET: a tool for probabilistic worst case execution time analysis of real–time systems," technical report YCS353 (2003), University of York, Department of Computer Science, York, YO10 5DD, United Kingdom, Apr. 2003.

[4] S. M. Petters, "Comparison of trace generation methods for measurement based WCET analysis," in *3nd Intl. Workshop on Worst Case Execution Time Analysis*, (Porto, Portugal), July 1 2003.

[5] J. Engblom, "Static properties of commercial embedded real-time programs, and their implication for worst-case execution time analysis," in *Proceedings of the 5th IEEE Real-Time Technology and Applications Symposium (RTAS '99)*, (Vancouver, Canada), IEEE, June 1999.