A Formal Framework for Precise Parametric WCET Formulas*

Benedikt Huber, Daniel Prokesch, and Peter Puschner

Institute of Computer Engineering
Vienna University of Technology, Austria
{benedikt,daniel,peter}@vmars.tuwien.ac.at

Abstract

Parametric worst-case execution time (WCET) formulas are a valuable tool to estimate the impact of input data properties on the WCET at design time, or to guide scheduling decisions at runtime. Previous approaches to parametric WCET analysis either provide only informal ad-hoc solutions or tend to be rather pessimistic, as they do not take flow constraints other than simple loop bounds into account. We develop a formal framework around path- and frequency expressions, which allow us to reason about execution frequencies of program parts. Starting from a reducible control flow graph and a set of (parametric) constraints, we show how to obtain frequency expressions and refine them by means of sound approximations, which account for more sophisticated flow constraints. Finally, we obtain closed-form parametric WCET formulas by means of partial evaluation. We developed a prototype, implementing our solution to parametric WCET analysis, and compared existing approaches within our setting. As our framework supports fine-grained transformations to improve the precision of parametric formulas, it allows to focus on important flow relations in order to avoid intractably large formulas.

1998 ACM Subject Classification C.3 Real-Time and Embedded Systems

Keywords and phrases Worst-case execution time analysis, parametric WCET analysis, path expressions, frequency expressions, algebraic framework

Digital Object Identifier 10.4230/OASIcs.WCET.2012.91

1 Introduction

In hard real-time systems, it is crucial to guarantee that timing constraints are met. Consequently, determining the maximum time it might take to execute a task, its so called Worst-Case Execution Time (WCET), is both a necessary task in certification, and an important metric in the design of hard real-time systems. Since the early days of WCET analysis, there is a vital interest in parametric WCET analysis, which attempts to calculate a closed-form formula for the WCET, parametrized over an abstraction of the input space.

Formulas describing the WCET are particularly interesting during development. For example, formulas are well-suited to specify the timing behavior of components, or to classify the impact of input on the timing behavior [9]. Furthermore, WCET formulas can be used to determine the WCET depending on the actual input at runtime [18, 7, 15], which can guide dynamic scheduling, or facilitate the early detection of timing problems. We are particularly interested in the application of these techniques to optimization and the classification of the impact of architectural parameters.

* This work was partially funded under the European Union's 7th Framework Programme under grant agreement no. 288008: Time-predictable Multi-Core Architecture for Embedded Systems (T-CREST).

12th International Workshop on Worst-Case Execution Time Analysis (WCET 2012).
Editor: Tullio Vardanega; pp. 91–102

OpenAccess Series in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

In recent years, there has been active development on techniques for determining symbolic flow facts, which characterize possible execution paths depending on the input data and reduce the search space for the worst-case execution path. Interesting examples include the automated computation of symbolic loop bounds [19] and solving of recurrence relations derived from abstract programs [1]. However, as already noted in the article of Chapman [6] in 1994, parametric WCET formulas are often not precise enough to allow for the derivation of tight bounds, if they fail to take additional flow facts (besides loop bounds) into account.

Flow facts of particular interest are bounds of inner loops in nested loops, where the bound of an inner loop varies with the iterations of the outer loop. Infeasible pairs model that two statements are mutually exclusive with respect to an execution context, and are common in embedded system code, especially in auto-generated code from synchronous languages or MATLAB Simulink [5].

The Implicit Path Enumeration Technique (IPET, [13, 16]) currently is the most widely-used technique for calculating the WCET. It generates an integer linear program (ILP), for which sophisticated solvers are available. In ILP, it is relatively easy to model flow facts besides loop bounds, improving the precision of the WCET bound. There is a parametric variant of the ILP problem called Parametric Integer Programming [14]; experiments with solvers for this powerful problem description language have been disappointing though [4, 3]. A function generating ILPs for each concrete input data configuration can also be viewed as a form of parametric ILP. While this approach has some interesting applications [5], its applicability is limited as the number of concrete ILP problems to solve increases exponentially in the number of input bits.

1.1 Motivating Example

The motivating example in Figure 1 illustrates the problem of taking additional flow facts into account, in this case a so-called triangle loop.

```
/* Nested loops */
void f(int N) {
  /* Loop Bound: N */
  for (i = 0; i < N; i++)
    if rand()
      B4();
    else
      B5();
      /* Loop Bound: i (N-1) */
      for (j = 0; j < i; j++)
        /* Global Bound: N (N-1) / 2 */
        B7();
      end
    end
  end
}

/* Symbolic Frequency Bounds depending on N */

f_B4 + f_B5 ≤ N f_B1     (1)
f_B7 ≤ (N-1) f_B5        (2)
f_B7 ≤ ½ N (N-1) f_B1    (3)
```

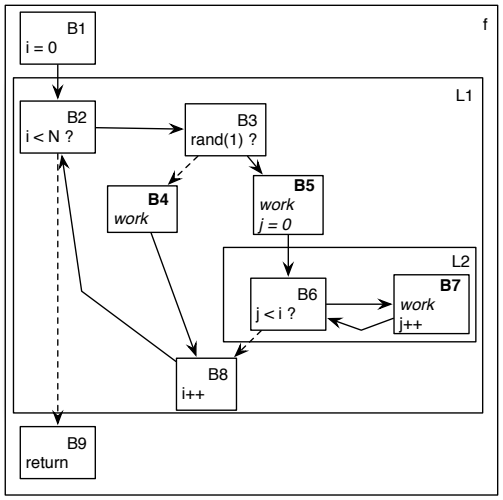

Figure 1 Example: Triangle Loop

In order to simplify the presentation, we will assume that the cost of all nodes but B_4, B_5 and B_7 is zero. Considering each iteration of the outer loop in turn, it is not difficult to

see that a precise bound for the WCET of f is

$$\mathrm{WCET}(f, N) = \sum_{i=0}^{N} \max(B_4, B_5 + iB_7)\,. \tag{1}$$

While automatically calculating a closed formula for the exact parametric WCET of f is hard, it is interesting to investigate different approximations.[1] If no flow facts but symbolic loop bounds are taken into account [6, 2], the formula closely corresponds to a regular expression describing the set of paths, replacing concatenation by addition, and set union by the **max** operator. For loops, the loop body is multiplied by the symbolic loop bound, in this case N for the outer loop, and $N-1$ for the inner loop. After simplification, we thus obtain

$$\mathrm{WCET}(f, N) = \max(NB_4, NB_5 + N(N-1)B_7)\,. \tag{2}$$

For larger values of N or B_7, the quadratic term dominates, leading to a WCET overestimation of up to 100%. Another recent approach is that of [4], which propagates constraints for each node, and then computes the sum of all nodes, multiplied by their symbolic execution bound. In this case, this would lead to the approximation

$$\mathrm{WCET}(f, N) = NB_4 + NB_5 + \frac{N(N-1)}{2}B_7\,. \tag{3}$$

This approach works well for single-path programs, but is rather pessimistic otherwise, as choice is mapped to addition instead of the maximum. Both of the previous approaches are easy to model in our framework. The techniques presented in this article allow to derive the following approximation, and prove it correct:

$$\mathrm{WCET}(f, N) = \max(NB_4, NB_5) + \frac{N(N-1)}{2}B_7\,. \tag{4}$$

This leads to a slight over-approximation if B_4 is expensive, but is a better approximation than the first one in the common case. In order to further improve the preciseness of the formula, one may introduce a case distinction, and either use Equation 2, if B_4 is executed at least $\frac{N}{2}$ times, or Equation 4 if it is executed less often. This approach increases the size of the formula, however, and thus in general may lead to intractably large formulas.

We believe that there is no single optimal strategy for constructing parametric formulas, and thus lobby for a formal framework, presented next, which allows to selectively refine formulas. In contrast to the related framework of Colin and Bernat [8], we chose an algebraic approach which is decoupled from the semantics of the analyzed program, and permits relatively simple correctness proofs. In Section 4, we describe the construction of formulas, the refinement of formulas using given, symbolic constraints, as well as the simplification and evaluation of formulas. Our experiments are described in Section 5, followed by our conclusion in Section 6.

[1] For this particular problem, the exact WCET can be computed manually, by determining the smallest integer k, such that if $i = k$, executing the nested loop is more expensive than executing B_4. With $k = \mathbf{min}(N, \mathbf{max}(0, \lceil \frac{B_4 - B_5}{B_7} \rceil))$, the WCET is then given by $\mathrm{WCET}(f, N) = kB_4 + (N-k)B_5 + \frac{N(N-1)-k(k-1)}{2}B_7\,.$

2 Background

In this work, we are mainly concerned with the construction of parametric WCET formulas from a given program representation, and the refinement of formulas using given program flow constraints. Therefore, we briefly review control flow representation and flow constraints, but do not address other relevant issues here, such as program-flow or processor-behavior analysis.

2.1 Control Flow Representation

The representation of programs in our analysis closely follows the machine code representation in the LLVM compiler framework[2], which is a central component of our evaluation framework.

A *control flow graph* (CFG) $\mathcal{G} = \langle V, E \rangle$ models the possible execution sequences of a function. Each node $v \in V$ corresponds to the execution of a sequence of instructions, and an edge (v, v') from $E \subseteq V \times V$ to a possible change of control from the last instruction of v to the first instruction of v'. The successors of a node v are given by $\text{succ}(v) = \{v' \mid (v, v') \in E\}$, the predecessors by $\text{pred}(v) = \{v' \mid (v', v) \in E\}$. A node v_d dominates a node v (denoted as v_d dom v) if every path from the start node to v must go through v_d. v_d strictly dominates v if v_d dom v and $v_d \neq v$.

We require several properties of the CFG representation, which are established in a preprocessing step. Each CFG $\mathcal{G}$ has a unique entry node $s_\mathcal{G}$, the only node where the set of predecessors is empty, and a unique exit node $t_\mathcal{G}$, with $\text{succ}(t_\mathcal{G}) = \emptyset$.

We only consider reducible CFGs [11] and thus irreducible loops need to be eliminated before WCET analysis. In a reducible CFG, loops $L \subseteq V$ are identified by a unique *header node* h_L, which dominates all loop members $v \in V_L$. A *back edge* is an edge from a member of a loop L to its loop header h_L. The acyclic *forward CFG* is obtained from a CFG by removing all back edges. Loops may be nested: The loop L_2 is an *inner loop* of L_1, if the header h_{L_2} is a member of loop L_1.

2.2 Flow constraints

In order to be WCET-analyzable, the maximum number of iterations of any loop in a program must be statically determinable. Applying to the CFG representation of a program, we define the *loop bound* of a loop L to be the maximum number of times the loop header node h_L of L is entered via any of its backedges in every execution.

Considering only the control-flow structure of the program together with simple numeric loop bounds will in most cases lead to an imprecise WCET bound. A reason for this are *infeasible paths*, i.e., structurally possible program paths that are not taken in any execution due to functional dependencies of program variables. Most prominent examples are mutually exclusive statements, and *triangle loops*, i.e., nested loops in which the loop bound of the inner loop depends on the iteration counter of the outer loop.

Linear flow constraints are the basis for IPET-based WCET calculation methods. In the corresponding ILP problem, the control-flow structure is expressed by means of linear relations between execution frequencies of CFG edges. Additional restrictions, for example to express relations between edge execution frequencies in different loop scopes, can easily be added into the system of linear constraints. Linear flow constraints have the form $\sum_i a_i \cdot f_{e_i} \leq C$, where f_{e_i} is the execution frequency of edge e_i, and $a_i, C \in \mathbb{Z}$ are constants.

[2] `http://www.llvm.org`

In case of symbolic a_i or C, the ILP problem is parametric and cannot be solved by standard IPET methods. Flow facts are either provided through manual annotations by the programmer or automatically derived by static program analysis.

3 Formal Framework

3.1 Path Expressions

Based on the work of [17], we can regard any path π in a directed graph $\mathcal{G} = \langle V, E \rangle$ as a string over E. A *path expression* P of *type* (v, w) with $v, w \in V$ (denoted as $P(v, w)$) is a regular expression over E such that every string π in the language $\sigma(P)$ is a path from v to w. Let $P(v, w)$ be a path expression of type (v, w). Then, all subexpressions P_1 and P_2 of path expression P are also path expressions, whose type can be defined recursively as follows [17]:

1. If $P = P_1 \cup P_2$, then P_1 and P_2 are of type (v, w) (alternative paths).
2. If $P = P_1 \cdot P_2$, there must be a unique vertex u such that P_1 is of type (v, u) and P_2 is of type (u, w) (path concatenation).
3. If $P = P_1^*$, then $v = w$ and P_1 is of type (v, v) (loop).

We call a path expression *complete* iff $\sigma(P(v, w))$ is the set of *all* paths from v to w in $\mathcal{G}$. For any given CFG $\mathcal{G}$ with entry node s and exit node t, the set of all structurally feasible (possibly infinite) paths through the CFG is defined by a complete path expression $P(s, t)$.

The underlying algebraic structure of path expressions is a Kleene algebra, i.e., an idempotent semiring (dioid) over E with the two binary operations of alternative paths $\cup$ as addition with neutral element $\emptyset$ and path concatenation $\cdot$ as multiplication with neutral element ε (empty string), and the additional unary operation of repetition * (cf. Table 1).

a^* is equivalent to the infinite expansion to $(\varepsilon \cup a \cup aa \cup aaa \cup \ldots)$. We exploit the algebraic structure to keep the a path expression P compact, as equivalence transformations on P do not change the language $\sigma(P)$.

Due to the associativity of both (binary) operations $\cup$ and $\cdot$ and by assigning operator precedences in the order *, $\cdot$, $\cup$ (starting with the highest), we can omit most brackets in path expressions. We also omit $\cdot$ in the notation, as usual for multiplication.

3.2 Frequency Expressions

Instead of calculating a cost formula directly from path expressions, we first introduce an abstraction from the set of paths to node frequencies. This abstraction lies at the heart of the successful IPET analysis, and is the basis for the concept of linear flow constraints. In our context, it permits us take flow constraints into account, to reason about node frequencies and prove the correctness of formula transformations.

The frequency $f_\pi(e)$ of an edge e on a path π is defined as the number of occurrences of e in π; that is, f_π is a function mapping edges to their occurrence count in π. A frequency constraint is a predicate on the frequencies of edges on a path, and acts a filter selecting valid paths from the set of all structurally possible ones. Formally, given a path expression $P(v, w)$, and a set of frequency constraints $\mathcal{C}$, the set of *valid paths* for $P(v, w, \mathcal{C})$ is given by $\sigma(P, \mathcal{C}) = \{\pi \in \sigma(P) \mid \forall C \in \mathcal{C} : C(f_\pi)\}$.

An expression P' is a *sound approximation* of $P(v, w, \mathcal{C})$, if $\sigma(P') \supseteq \sigma(P, \mathcal{C})$, that is, each valid path is included in the language described by P'. An approximation of an expression P' is called *exact* with respect to C, if $\sigma(P') = \sigma(P, \mathcal{C})$.

Frequency expressions are syntactically similar to path expressions; we just introduce a bounds notation $P^{[L,U]}$ which is equivalent to the expansion $\bigcup_{L \le i \le U} P^i$, with $P^i = P^{[i,i]} = PP \cdots P$ (i times) and $P^0 = \varepsilon$. Consequently we replace a path expression P^* by $P^{[0,\infty]}$ to simplify notation during constraint refinement (see Section 4.2). However, concatenation is commutative for frequency expressions. For example, while $P_1 = e_1 e_2$ and $P_2 = e_2 e_1$ are different path expressions, they are equivalent if interpreted as frequency expressions.

Frequency expressions are useful for two reasons. First, they allow us to take non-local constraints into account, as discussed in Section 4.2. Second, frequency expressions allow us to limit the growth of a formula when splitting subexpressions. For example, suppose that e_i and e_j are mutually exclusive, that is $\{\neg(f_{e_i} > 0 \wedge f_{e_j} > 0)\} \in \mathcal{C}$. Then $P = e_i \cdot Q \cdot e_j$ can be refined to $(e_i \cup e_j) \cdot Q$, while for path expressions we would be stuck with $(e_i \cdot Q) \cup (Q \cdot e_j)$.

3.3 Cost Expressions

Let $c : E \to \mathbb{N} \cup \{-\infty\}$ be a cost function, which assigns to each edge $e \in E$ of a CFG $\mathcal{G} = \langle V, E \rangle$ its maximum execution cost c_e. [3] Then we can derive a symbolic expression for the (possibly approximated) maximum cost $c(P(s,t))$ over all paths from the entry node s to the exit node t from the frequency expression $P(s,t)$ (or a sound approximation thereof), by replacing the underlying algebraic structure with $\langle \mathbb{N} \cup \{-\infty\}, \mathbf{max}, +, -\infty, 0 \rangle$, which we denote as $\mathbb{N}_{\max}$ in the following. $\mathbb{N}_{\max}$ is also a commutative dioid like the algebra of frequency expressions, with a total order defined on its elements by the order of the natural numbers, or equivalently, using the **max** operation: $a \le b$ if $\mathbf{max}(a,b) = b$.

While in general for the frequency expression $(a \cup b)^N = \bigcup_{k=0}^{N} a^k\, b^{N-k}$, the total order of the elements in $\mathbb{N}_{\max}$ allows for following simplification due to monotonicity: $N * \mathbf{max}(a,b) = \mathbf{max}(N*a, N*b)$. Furthermore, as $\mathbf{max}(N*a, (N+1)*a) = (N+1)*a$, only the upper bound U in a frequency expression $P^{[L,U]}$ needs to be considered when calculating the maximum cost. As a consequence, in $\mathbb{N}_{\max}$ we cannot conveniently reason about edge (or node) frequencies, but only about (possibly approximated) path costs.

Table 1 provides a comparison of path-, frequency- and cost expressions.

4 Construction and Evaluation of WCET Formulas

In this section, we first give a concise description on how to construct path expressions in our framework, then present the refinement of frequency expressions to account for flow constraints, and finally describe the normalization of frequency expressions and our partial evaluation framework.

4.1 Building Path Expressions

We first consider an acyclic CFG $\mathcal{G} = \langle V, E \rangle$ with entry node s and exit node t. We want to obtain a complete path expression $P(s,t)$, approximating the set of valid paths from s to t. We recursively define $P(v,w)$ as

$$P(v,w) = \begin{cases} \varepsilon & \text{if } v = w \\ \bigcup_{(w',w) \in E} \; (P(v,w') \cdot (w',w)) & \text{if } v \neq w \end{cases}$$

[3] Given execution costs c_v of basic blocks $v \in V$ of the CFG, edge costs can be derived by attaching c_v either to all of its incoming edges or all of its outgoing edges.

Table 1 Comparison of path expressions, frequency expressions and cost expressions.

Path expressions	
Interpretation	Language of structurally possible program paths
Algebraic structure	Kleene algebra $\langle E, \cup, \cdot, ^*, \emptyset, \varepsilon\rangle$ (idempot. semiring with Kleene closure) $\cup$ is associative, commutative and idempotent; $\cdot$ is associative $\cdot$ is distributive w.r.t. $\cup$: $(a \cup b)\cdot c = (a\cdot c)\cup(b\cdot c)$, $a\cup(b\cdot c) = (a\cdot b)\cup(a\cdot c)$ Zero element: $a \cup \emptyset = \emptyset \cup a = a$; Identity element: $a \cdot \varepsilon = \varepsilon \cdot a = a$ $\emptyset$ annihilates E w.r.t. $\cdot$ $(a \cdot \emptyset = \emptyset \cdot a = \emptyset)$ $\emptyset^* = \varepsilon^* = \varepsilon$
Example	$(e_{3,4}e_{4,8}) \cup (e_{3,5}e_{5,6}(e_{6,7}e_{7,6})^*e_{6,8})$
Frequency expressions	
Interpretation	Execution frequencies of edges in the CFG
Algebraic structure	Similar to path expr., but comm. dioid: $\cdot$ is commutative $(a \cdot b = b \cdot a)$ Bounds notation for $P^{[L,U]}$ repetition bounds, $P^* \mapsto P^{[0,\infty]}$ $\emptyset^{[0,U]} = \emptyset^{[0,0]} = \varepsilon$, while $\emptyset^{[1,U]} = \emptyset$
Example	$(e_{3,4}e_{4,8}) \cup (e_{3,5}e_{5,6}e_{6,8}(e_{6,7}e_{7,6})^{[0,N]})$
Cost expressions	
Interpretation	Formula for (maximum) execution cost
Algebraic structure	Commutative dioid $\langle \mathbb{N}_{\cup\{-\infty\}}, \mathbf{max}, +, -\infty, 0\rangle$ Due to total order and monotonicity: $N{*}\mathbf{max}(a,b) = \mathbf{max}(N{*}a, N{*}b)$
Example	$\max(NB_4, NB_5) + \frac{N(N-1)}{2}B_7$

As we assumed the CFG to be acyclic, there is a partial order $\prec$ on the nodes of the graph with $w \in \text{pred}(v) \Rightarrow w \prec v$, for all $v, w \in V$. By determining this topological order, and representing each expression $P(s, v)$ only once in memory, we obtain a closed form for $P(s, t)$ in time and space linear in the size of the CFG.

Now consider a CFGs $\mathcal{G} = \langle V, E\rangle$ with reducible loops. We observe that the set of cycle-free paths (i.e., those which do not include back edges) is generated by calculating $P(s, t)$ for the forward CFG $\mathcal{G}^F = \langle V, E^F\rangle$. Furthermore, all cycles are composed of paths starting from a loop header and ending at the corresponding back edge. Therefore, in the general case we recursively define $P(v, w)$ as

$$P(v,v) = \begin{cases} \varepsilon & \text{if } v \text{ is not a loop header} \\ \bigcup_{(v',v)\in E\setminus E^F} P(v,v')\cdot(v',v) & \text{if } v \text{ is header of loop } L \end{cases}$$

$$P(v,w) = \bigcup_{(w',w)\in E^F} (P(v,w')\cdot(w',w))\cdot P(w,w)^* \qquad \text{if } v \neq w$$

The construction of path expressions simply takes the union of all paths leading to predecessors. However, those paths will often have a common prefix, namely the path expression from the entry to the dominator of predecessors. Therefore, we factor out common prefixes during construction, using the equivalence $\bigcup_j (P(s,d)\cdot P(d,j)) = P(s,d)\cdot\left(\bigcup_j P(d,j)\right)$.

▶ **Example 1.** Consider the example presented in Section 1.1. Let $e_{i,j}$ denote the edge from B_i to B_j. Then applying the construction algorithm, and factoring out the common prefix

$P(B_2, B_3)$ leads to the following path expressions:

$$P(B_1, B_9) = e_{1,2}P(B_2, B_2)^* e_{2,9}$$
$$P(B_2, B_2) = e_{2,3}P(B_3, B_8)e_{8,2}$$
$$P(B_3, B_8) = (e_{3,4}e_{4,8}) \cup (e_{3,5}e_{5,6}P(B_6, B_6)^* e_{6,8})$$
$$P(B_6, B_6) = e_{6,7}e_{7,6}$$

4.2 Constraint Refinement of Frequency Expressions

The formulas derived so far express possible paths constrained only by the program structure as obtained from the CFG. In order to calculate a (precise) WCET bound, we need to take flow constraints into account. As argued before, frequency expressions are a well-suited formalism for this task. Therefore, we interpret the initial path expression as frequency expression before constraining the formula.

4.2.1 Constraint Refinement

The following observation illustrates how to take local frequency bounds into account.

▶ Observation 1 (Constraint Refinement). Given a a frequency expression $P(s,t) = R \cdot Q(u,v)^{[A,B]}$, and a frequency constraint $C = \sum_{(u',u)\in E} f_{(u',u)} \leq N$, $C \in \mathcal{C}$. Then the frequency expression $P'(s,t) = R \cdot Q(u,v)^{[A,N]}$ is a sound approximation of $P(s,t,\mathcal{C})$.

Due to the way frequency expressions are constructed, it is thus always possible to account for constraints which limit the frequency of a node relative to its immediate dominator. For non-empty frequency expressions $Q(v,v)$, corresponding to the body of a loop, the constraint only needs to refer to total frequency of the loop's back edges. Local constraints, in particular simple loop bounds, should always be applied to frequency expressions, in order to facilitate the calculation of a numeric WCET bound.

▶ **Example 2.** Consider the path expressions presented in Example 1, interpreted as frequency expressions. In the motivating example, we have $f(e_{8,2}) \leq N \in \mathcal{C}$ for all paths from B_1 to B_9. Applying Observation 1, we thus get

$$P'(B_1, B_9, \mathcal{C}) = e_{1,2}P(B_2, B_2)^{[0,N]}e_{2,9}$$

4.2.2 Global Bounds

Frequency expressions also allow us to take non-local frequency bounds into account. We start with the following two observations:

▶ Observation 2. Given a frequency expression $P(s,t) = (Q \cdot R)^{[A,B]}$, the expression $P'(s,t) = Q^{[A,B]} \cdot R^{[A,B]}$ is a sound approximation of $P(s,t)$.

▶ Observation 3. Given a frequency expression $P(s,t) = ((Q \cdot R) \cup S)^{[A,B]}$. Then $P'(s,t) = Q^{[0,B]} \cdot (R \cup S)^{[A,B]}$ is a sound approximation of $P(s,t)$.

These observations allow to move subexpressions to an outer scope, which in turn enables the inclusion of non-local loop bounds. The following example illustrates this technique.

▶ **Example 3.** Consider the frequency expression from Example 2. Applying both transformations presented above to lift $P(B_6, B_6)$ gives

$$P''(B_1, B_9) = e_{1,2}P''(B_2, B_2)^{[0,N]}P(B_6, B_6)^{[0,N(N-1)]}e_{2,9}$$
$$P''(B_2, B_2) = e_{2,3}\left((e_{3,4}e_{4,8}) \cup (e_{3,5}e_{5,6}e_{6,8})\right)e_{8,2}$$

Then applying constraint refinement to $P(B_6, B_6)$ results in

$$P'''(B_1, B_9) = e_{1,2} P''(B_2, B_2)^{[0,N]} P(B_6, B_6)^{[0, \frac{1}{2}N(N-1)]} e_{2,9}$$

Note, that while using the first observation might improve the WCET bound, applying the second or third one may make it worse. Thus it is necessary to devise a heuristic which decides whether a non-local constraint should be applied. Another alternative to overcome possible degradation is to mimic the IPET approach and split the formula, distinguishing the case when the non-local constraint is useful, and the one when the local one is better.

4.2.3 Infeasible Pairs

An infeasible node $x \in V$ is a node that does not lie on any feasible path from s to t. Given a frequency expression $P(s,t)$ and an infeasible node x, the set of valid paths $\sigma(P, \mathcal{C})$ satisfies the constraint $C = \{f_e = 0 \mid e \in E_x\}$, $C \subseteq \mathcal{C}$, where E_x is the set of edges incident to x, i.e., the frequency of all incoming edges and all outgoing edges of x is zero.

The frequency expression $P(s,t)[e \mapsto \emptyset]$, $e \in E_x$ is a sound approximation for $P(s,t,\mathcal{C})$.[4] Recall that $\emptyset^{[0,U]} = \emptyset^{[0,0]} = \varepsilon$, while $\emptyset^{[1,U]} = \emptyset$. With this transformation, every subexpression P' of the form $P' = \prod_k P_k$ where some $P_k = \emptyset$ will thus be annihilated in P, i.e., any path that formerly contained x at least once, is pruned from the set $\sigma(P)$, while all other paths are preserved.

An infeasible pair (x, y) is a pair of nodes $x, y \in V$ such that any path from s to t that contains both x and y is infeasible. We can restrict the set of valid paths by taking the union of the path set in which x is infeasible and the path set in which y is infeasible. Formally, we obtain a sound approximation for $P(s,t,\mathcal{C})$ where $C = \{\neg\left(f_{e_x} > 0 \wedge f_{e_y} > 0\right), e_x \in E_x, e_y \in E_y\}$, $C \subseteq \mathcal{C}$ by the frequency expression $P(s,t)[e_x \mapsto \emptyset] \cup P(s,t)[e_y \mapsto \emptyset]$, $e_x \in E_x$, $e_y \in E_y$. The resulting expression describes the set of paths pruned only of paths that contain both nodes x and y, and hence is an exact approximation.

Obviously, the size of the formula increases when taking infeasible pairs into account. Indeed, as WCET calculation is NP-complete in the presence of infeasible pairs[5], it is not possible to obtain a compact formula in the general case. However, by exploiting commutativity in frequency expressions to reduce the size of the duplicated part of the formula, and by taking only those infeasible pairs into account which improve the WCET significantly, we hope keep the size of the formula in reasonable limits.

4.3 Simplification and Partial Evaluation

We use the properties of the frequency expression algebra to normalize frequency and cost expressions, which provides the basis for the partial evaluation described at the end of this section.

4.3.1 Normalized Frequency Expressions

In the normalized form, every frequency expression is either a single node, a union $\bigcup_i P_i$ of normalized frequency expressions, or a product $\prod_i P_i^{[L_i,U_i]}$. The normalized form of frequency expressions has the following properties:

[4] $P[e \mapsto e']$ denotes the frequency expression P, with all occurrences of the subexpression e replaced by e'.

[5] As can be shown by a polynomial-time reduction of W2SAT.

- *Subsumption (Unions)*: Given two products $P = \prod_i P_i^{[L_i,U_i]}$ and $Q = \prod_j Q_j^{[L_j,U_j]}$, if for every $Q_j^{[L_j,U_j]}$ there is a $P_i^{[L_i,U_i]}$ with $P_i = Q_j$, $L_i \leq L_j$ and $U_j \leq U_i$, then P subsumes Q ($\sigma(Q) \subseteq \sigma(P)$) and thus $P \cup Q = P$. For every normalized frequency expression $\bigcup_i P_i$, each P_i is a product, and if P_i subsumes P_j, P_j is not present in the union.
- *Frequency Product (Products)*: The product $P^{[L_1,U_1]} \cdot P^{[L_2,U_2]}$ simplifies to $P^{[L_1+L_2,U_1+U_2]}$, and $(\prod_i P_i^{[L_1,U_1]})^{[L_2,U_2]}$ simplifies to $\prod_i P_i^{[L_1 L_2,U_1 U_2]}$. Furthermore, we apply $P \cdot \bigcup_\emptyset = \bigcup_\emptyset$ whenever possible. Therefore, in every frequency expression $\prod_i P_i^{[L_i,U_i]}$, all P_i are distinct, and each P_i is either a non-empty union of frequency expressions or a single node.

4.3.2 Partial Evaluation

Given a frequency expression $P(s,t)$ for a control flow graph, we perform partial evaluation to obtain a numeric or parametric formula for the WCET. We call this step *partial* evaluation, as the evaluation functions c and f are allowed to be partial.

For evaluation purposes, we introduce a special node $\underline{c}$, which represents one unit of cost. The evaluation algorithm takes a frequency expression $P(s,t)$, a partial cost function c (assigning costs to edges), and a frequency evaluation function f, transforming symbolic frequencies (either into numeric ones or different symbolic frequencies, e.g. to reflect some parameter of interest, like size of an input array).

First, all edges e where $c(e)$ is defined are replaced by $\underline{c}^{c(e)}$, and all expressions $P^{[L,U]}$ are simplified to $P^{[f(L),f(U)]}$. The actual evaluation then corresponds to the normalization of the frequency expression, as described before. This is sufficient for full evaluation, while for partial evaluation the size of the formula can be further reduced by additional simplifications, which exploit the total order of numeric cost values.

Given the result of the partial evaluation, we would like to obtain information on node frequencies in the evaluated formula. In principle, this can be achieved by keeping references to original expressions during simplifications, and keeping track of selected branches in unions, though this has not been elaborated yet.

5 Experiments

In order to validate the applicability of our parametric execution time formula framework, we implemented a prototype on top of the Open Timing Analysis Platform [12]. This evaluation framework is based on the LLVM compiler framework, and extracts description of machine code CFGs from the internal compiler representation. Flow facts are provided by the SWEET [10] analysis tool, developed at the Mälardalen Real-Time Research Center (MRTC) which is integrated in our evaluation framework.

For our preliminary experiments, we generate machine code for the ARM instruction set and use a simple cost model (one cycle per instruction). In Table 2, we present results for three benchmarks which feature triangle loops. One is adapted from the motivating example in Section 1.1, and two are taken from the MRTC benchmark set.

In these experiments, formulas are parametric with respect to loop bounds (first column). Local bounds L_i constrain the loop iteration count relative to the loop entry frequency, global bounds G_i are relative to the function entry. The fourth column displays the result of evaluating the formulas with the specified numeric loop bounds.

We compare the standard approach which only uses simple loop bounds (*local bounds only*), the result which only uses bounds relative to the function entry (*global bounds only*),

Table 2 Results of the comparison of different approaches to parametric analysis.

	approach	formula	cycles
`intro_example`	local bounds only	$6 + 12L_1 + L_1 * \max(28, 19L_6)$	1836
$L_1 = 10, L_6 = 9,$	global bounds only	$6 + 45L_1 + 19G_6$	1311
$G_6 = 45$	lifting inner loops	$6 + 40L_1 + 19G_6$	1261
	IPET	—	1121
`insert_sort`	local bounds only	$29 + 14L_1 + 11(L_1\, L_3)$	1046
$L_1 = 9, L_3 = 9,$	lifting, global	$29 + 14L_1 + 11G_3$	650
$G_3 = 45$	IPET	—	650
`janne_complex`	local bounds only	$32 + 14L_4 + 22L_2 + 14(L_2\, L_4)$	1216
$L_2 = 8, L_4 = 8,$	global bounds only	$18 + 8L_2 + 15G_4$	262
$G_4 = 12$	lifting inner loops	$18 + 8L_2 + 14G_4$	250
	IPET	—	250

and the transformation moving nested loops to the outer scopes (*lifting inner loops*), described in Section 4.2.2.

6 Conclusion

We presented a framework for the calculation of symbolic execution time formulas, which provides a solid foundation for parametric WCET analysis. We construct path expressions from control flow graphs, and then add commutativity to obtain frequency expressions. The key idea is that while frequency expressions are easier to work with than path expressions, they still represent a sound approximation to the set of possible execution paths. By the virtue of this property, it is possible to selectively include additional flow constraints, for example global loop bounds or infeasible pairs, and prove this refinements correct.

The preliminary experiments have been encouraging, and the resulting formulas indeed reflect our intuition, and suggest that this approach is not only theoretically sound, but also works well in practice. We are eager to extend it to a fully-featured implementation, supporting context-sensitive supergraphs, feedback on worst-case frequencies, and additional formula refinements. There are many interesting applications to parametric WCET analysis, especially at design time, and we believe that this framework provides a good basis for further improving the state-of-the art in this area.

References

1 Elvira Albert, Puri Arenas, Samir Genaim, and Germán Puebla. Closed-form upper bounds in static cost analysis. *J. Autom. Reason.*, 46(2):161–203, February 2011.

2 Ernst Althaus, Sebastian Altmeyer, and Rouven Naujoks. Precise and efficient parametric path analysis. In *LCTES '11: Proceedings of the ACM SIGPLAN/SIGBED 2011 conference on Languages, compilers, and tools for embedded systems*, pages 141–150, New York, NY, USA, April 2011. ACM.

3 Sebastian Altmeyer. Parametric WCET analysis, parameter framework and parametric path analysis. Master's thesis, Universität des Saarlandes, 2006.

4 Stefan Bygde, Andreas Ermedahl, and Björn Lisper. An efficient algorithm for parametric wcet calculation. *Journal of Systems Architecture - Embedded Systems Design*, 57(6):614–624, 2011.

5 Susanna Byhlin, Andreas Ermedahl, Jan Gustafsson, and Björn Lisper. Applying static WCET analysis to automotive communication software. In *ECRTS*, pages 249–258. IEEE Computer Society, 2005.

6 Roderick Chapman. Worst-case timing analysis via finding longest paths in spark ada basic-path graphs. Technical Report Technical Report YCS-94-246, Department of Computer Science, University of York, October 1994.

7 Joel Coffman, Christopher Healy, Frank Mueller, and David Whalley. Generalizing parametric timing analysis. In *Proceedings of the 2007 ACM SIGPLAN/SIGBED conference on Languages, compilers, and tools for embedded systems*, LCTES '07, pages 152–154, New York, NY, USA, 2007. ACM.

8 Antoine Colin and Guillem Bernat. Scope-tree: A program representation for symbolic worst-case execution time analysis. In *Proceedings of the 14th Euromicro Conference on Real-Time Systems*, ECRTS '02, pages 50–, Washington, DC, USA, 2002. IEEE Computer Society.

9 S.V. Gheorghita, S. Stuijk, T. Basten, and H. Corporaal. Automatic scenario detection for improved WCET estimation. In *Design Automation Conference, 2005. Proceedings. 42nd*, pages 101 – 104, june 2005.

10 Jan Gustafsson, Andreas Ermedahl, Christer Sandberg, and Bjorn Lisper. Automatic derivation of loop bounds and infeasible paths for wcet analysis using abstract execution. In *Proceedings of the 27th IEEE International Real-Time Systems Symposium*, RTSS '06, pages 57–66, Washington, DC, USA, 2006. IEEE Computer Society.

11 Paul Havlak. Nesting of reducible and irreducible loops. *ACM Trans. Program. Lang. Syst.*, 19(4):557–567, July 1997.

12 Benedikt Huber, Wolfgang Puffitsch, and Peter Puschner. Towards an open timing analysis platform. In *11th International Workshop on Worst-Case Execution Time Analysis*, July 2011.

13 Yau-Tsun Steven Li and Sharad Malik. Performance analysis of embedded software using implicit path enumeration. In *Proceedings of the 32nd annual ACM/IEEE Design Automation Conference*, DAC '95, pages 456–461, New York, NY, USA, 1995. ACM.

14 Björn Lisper. Fully automatic, parametric worst-case execution time analysis. In Jan Gustafsson, editor, *WCET*, volume MDH-MRTC-116/2003-1-SE, pages 99–102. Department of Computer Science and Engineering, Mälardalen University, Box 883, 721 23 Västerås, Sweden, 2003.

15 Sibin Mohan, Frank Mueller, Michael Root, William Hawkins, Christopher Healy, David Whalley, and Emilio Vivancos. Parametric timing analysis and its application to dynamic voltage scaling. *ACM Trans. Embed. Comput. Syst.*, 10(2):25:1–25:34, January 2011.

16 Peter P. Puschner and Anton V. Schedl. Computing maximum task execution times - a graph-based approach. *Real-Time Systems*, 13(1):67–91, 1997.

17 Robert Endre Tarjan. A unified approach to path problems. *J. ACM*, 28(3):577–593, July 1981.

18 Emilio Vivancos, Christopher A. Healy, Frank Mueller, and David B. Whalley. Parametric timing analysis. In *LCTES/OM*, pages 88–93. ACM, 2001.

19 Florian Zuleger, Sumit Gulwani, Moritz Sinn, and Helmut Veith. Bound analysis of imperative programs with the size-change abstraction. In *Proceedings of the 18th international conference on Static analysis*, SAS'11, pages 280–297, Berlin, Heidelberg, 2011. Springer-Verlag.