TOWARDS AN AUTOMATIC PARAMETRIC WCET ANALYSIS[1]

Stefan Bygde and Björn Lisper
School of Innovation, Design and Engineering, Mälardalen University
Box 883, S-721 23 Västerås, Sweden
{stefan.bygde, bjorn.lisper}@mdh.se

Abstract
Static WCET analysis obtains a safe estimation of the WCET of a program. The timing behaviour of a program depends in many cases on input, and an analysis could take advantage of this information to produce a formula in input variables as estimation of the WCET, rather than a constant. A method to do this was suggested in [12]. We have implemented a working prototype of the method to evaluate its feasibility in practice. We show how to reduce complexity of the method and how to simplify parts of it to make it practical for implementation. The prototype implementation indicates that the method presented in [12] successfully can be implemented for a simple imperative language, mostly by using existing libraries.

1. Introduction

Most state-of-the-art static WCET analyses derive a constant time estimation of the WCET of a program. In the general case, this estimation has to be pessimistic in order to be safe. However, many tasks are re-configurable and/or reused in different applications and the worst case derived by the analysis applies only to one configuration/input. The real WCET for a fixed configuration or input data might be significantly tighter than the global worst case. One analysis could be made for each configuration or input data, and new analyses can be made as needed. This, however, is tedious, inflexible and requires that the analysis tools are available whenever the data or configuration of the program is changed. In [12] a static analysis which provides a safe parametric estimation of the WCET as a formula of input variables rather than as a constant value is presented. We have done a prototype implementation of this analysis and we will present some of the technical issues that arise in a practical implementation. The reason for the implementation is to test the feasibility of the analysis framework in practice. The contributions of this paper are

- Showing that the analysis can be implemented for simple programs (see Section 2)
- Reducing complexity of the analysis by reducing the number of variables
- Simplifying the method in [15] for counting solutions of presburger formulae to count points inside convex polyhedra
- Identifying the need for simplification of the resulting parametric WCET formula

[1]This work is supported by the Swedish Foundation for Strategic Research via the strategic research centre PROGRESS

Section 2 briefly summarises the method given in [12] and in Section 3 we show how we have implemented it. Section 4, shows how to simplify symbolic counting of the number of solutions based on [15]. Then, in section 5, we show how to reduce the complexity of the problem by eliminating variables from the calculation phase. In Section 6, we discuss the problem with a big and complex solution and suggestions on how to solve it. Finally we present related work in Section 7 and conclusions and future work in Section 8.

2. Method

We first introduce some terminology and make a few assumptions. By *program* we shall mean a small program, a task or a function which has a well defined set of input parameters. Furthermore, we assume that the analysed program is terminating and deterministic, and that variables that affect the program flow are integers or Booleans. We will refer to *program points* as the edges of the control flow graph (CFG). The analysed program will be denoted P, and its set of program points $\mathcal{Q}_P$. We shall assume that any program uses a set of variables $\mathcal{V}_P$ and that each program has a set of *input parameters* $\mathcal{I}_P \subseteq \mathcal{V}_P$ which has two properties: 1) They affect the program flow and 2) They are not changed during execution of P. As long as these properties are fulfilled the set of input variables can be chosen arbitrarily. Now we will for completeness summarise parts of the method described in [12].

The method assumes that there exists an analysis producing structural constraints[2] of the program and a low-level analysis that produces a WCET estimation for all atomic parts (program points) of the program. The idea here is to have the execution count of each program point bounded from above by a parameter and to perform an parametric IPET calculation. Other constraints, such as constraints on paths can also be bounded by parameters. The result of the parametric IPET calculation is a function expressed as a formula

$$\mathrm{PC}_P : \mathbb{N}^{|\mathcal{Q}_P|} \rightarrow \mathbb{N}$$

where the domain is a vector where each component represents the maximum bound on the execution count for a program point. We call the parameters in this vector *execution bound parameters*. Furthermore, we call the function PC_P *Parametric Calculation*. The maximum execution count for individual program points is often dependent on input parameters. The arguments of PC_P will therefore be computed in terms of $\mathcal{I}_P$. Formally, the arguments will be computed by a function

$$\mathrm{MEC}_P : \mathbb{Z}^{|\mathcal{I}_P|} \rightarrow \mathbb{N}^{|\mathcal{Q}_P|}$$

where the domain vector is an instantiation of the input parameters. We call this function the *Maximum Execution Count* function. When the parametric calculation and the maximum execution count functions have been computed, we can obtain the WCET estimation as a parametric formula in the input variables as the function

$$\mathrm{PWCET}_P : \mathbb{Z}^{|\mathcal{I}_P|} \rightarrow \mathbb{N} = \mathrm{PC}_P \circ \mathrm{MEC}_P \, .$$

The analysis boils down to finding the functions PC_P and MEC_P for a given program P. The function PC_P can be obtained by performing parametric IPET calculation using parametric ILP [9]. The MEC_P function can be obtained by observing the fact that in a deterministic, terminating program, each run-time state has to be unique[3]. Thus, the number of times a program point can be visited

[2] Constraints imposed by the graph structure of the CFG only

[3] If exactly the same state is encountered twice, the determinism of the program enforces the program to repeat the same sequence of states infinitely.

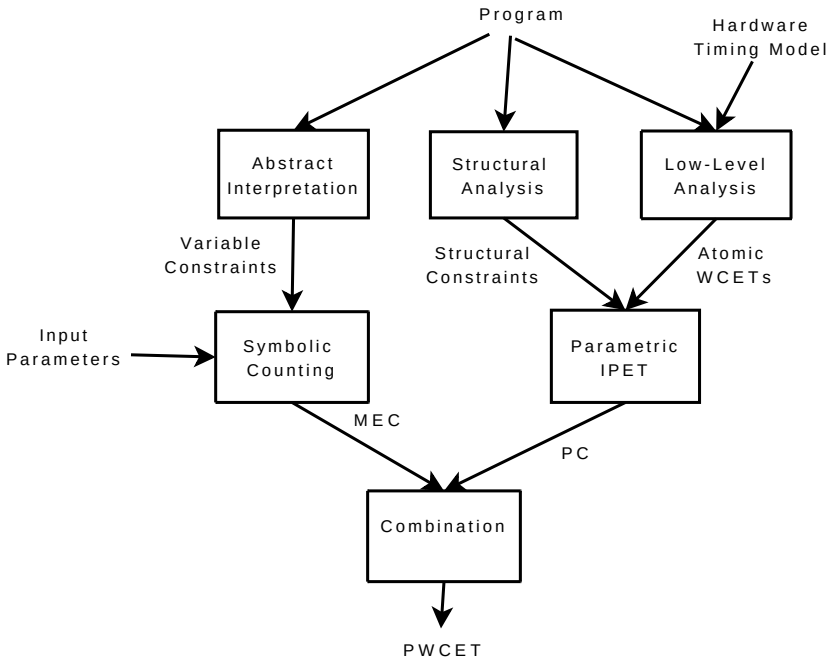

Figure 1. Work flow

is upper bounded by the number of distinct run-time states. Abstract Interpretation [6] can derive a super set of the set of possible run-time states at each program point. A relational domain is an abstract domain that preserves some of the relations among variables in the analysis. If the abstract domain used in the abstract interpretation is relational, then some of the variables can be used to parameterise the set of possible run-time states w.r.t. other variables. The maximum execution count function can then be obtained by counting the number of states contained in the solutions of the abstract interpretation with respect to the parameters. The work flow is shown in Figure 1.

3. Implementation

We have made a prototype implementation of the analysis presented in the last section. We represent programs as CFGs where nodes are start, stop, assignment or conditionals. Furthermore, all variables are assumed to be integers or Booleans. We do not consider function calls or pointers and the expressions in assignments and conditionals have to be linear. The reason for this restriction is that most relational abstract domains can only capture linear relations among variables. Since the computational tasks required to realise this are quite modular, we have implemented different parts independently of each other and mostly by reusing existing code and libraries. We present for each box in Figure 1 the corresponding implementation, except for structural analysis and low-level analysis which are assumed to exist. In our prototype we simply assume that all atomic parts of the program have the same constant WCET of ten clock cycles, but this could easily be replaced by the real results of a low-level analysis. Since we represent our program as a CFG, we can easily derive the structural constraints for each node by requiring that its in-flow should be equal to the out-flow.

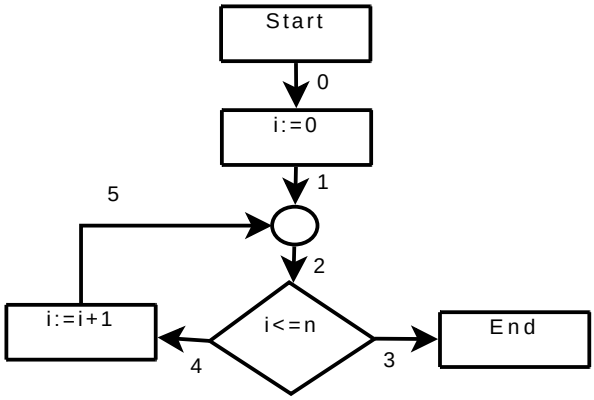

Figure 2. Example program L where all program points have been labeled

3.1. Abstract Interpretation

The main design decision of the abstract interpretation is the choice of abstract domain. The requirements of the abstract domain is that, in order to be parametric, it should be relational and that the domain can derive bounds on the values (to be able to count states). We have implemented the abstract interpretation with the polyhedral domain [7] using the new polka library [13]. Such an abstract interpretation can derive linear constraints among variables, enclosing the possible values of the program variables at a program point inside a convex polyhedron in n dimensions, where n is the number of variables. A simple framework for abstract interpretation for our CFGs has been implemented in C++. The implementation allows support for other abstract domains to be added with little effort. The polyhedral domain has exponential complexity in the number of dimensions (in this case, variables) but has a quite good precision. As an example, consider the CFG of a program we call L in Figure 2. Performing polyhedral abstract interpretation will yield a vector $\vec{a}$ of abstract values for each program point as follows:

$$\begin{array}{ll} a_0 = \top & a_3 = \{i \geq 0, i \geq n+1\} \\ a_1 = \{i = 0\} & a_4 = \{0 \leq i \leq n\} \\ a_2 = \{i \geq 0\} & a_5 = \{1 \leq i \leq n+1\} \end{array} \quad (1)$$

where $\top$ means "no information".

3.2. Counting States

To obtain the MEC_P function from the result of the abstract interpretation, we need to count the number of integer solutions for each system, parameterized in the set of chosen input parameters $\mathcal{I}_P$. We have implemented a simplification of the method in [15]. The method counts the number of solutions to a given presburger formula, of which a system of linear inequalities is a subset. We describe our modelling and simplification of the method in Section 4. The result from the symbolic counting represent the function MEC_P. For completeness we now sketch parts of the method in [15], it computes the result of generalised sums $(\Sigma V : P : x)$ where V is a set of variables to sum over, P is a presburger formula (the guard) and x is any formula. The result of such a sum is the sum for all variables $v \in V$ which satisfy P of x. As a simple example, $(\Sigma\{v\} : l \leq v \leq u : v)$ is equal to $\sum_{v=l}^{u} v$. The general method to compute $(\Sigma V : P : x)$ is to choose a variable $v \in V$ and compute the general sum

$$(\Sigma V \setminus \{v\} : P' : (\Sigma\{v\} : l \leq v \leq u : x))$$

where P' is P where all information about v is removed. Since $(\Sigma\{v\} : l \leq v \leq u : x)$ is equivalent to $\sum_{v=l}^{u} x$, known formulae of summations over the form of x can be used to simplify it. If $V \setminus \{v\}$ is non-empty another variable is chosen and the procedure is repeated until $V = \varnothing$, and the result is a sum of generalised sums $(\Sigma : G : x')$ which should be read as "x' if G holds, else 0". This result is symbolic in the variables occurring free in x or P but not in V. The situation is however not always this easy; variables can have several lower/upper bounds or be unbounded and bounds can be negative and/or rational. All these cases are handled in [15].

Given a vector $\vec{a} = (a_0\ a_1\ ...\ a_{m-1})$, with $m = |\mathcal{Q}_P|$ components, where each component is a convex polyhedron obtained from the abstract interpretation, we define the image through MEC_P of the k-th component as

$$\mathrm{MEC}_P(\vec{i})_k = (\Sigma(\mathcal{V}_P \setminus \mathcal{I}_P) : a_k : 1)$$

for $k = 0...m-1$. The right hand side is a formula of the elements in $\mathcal{I}_P$ which are instanced with the elements of $\vec{i}$. Consider the CFG L in Figure 2 and the result $\vec{a}$ from the abstract interpretation in (1). For L we have $\mathcal{V}_L = \{i, n\}$ and $\mathcal{I}_L = \{n\}$ so by symbolically counting the number of integer points inside the polyhedra of will give the MEC_L function as follows (we will here write Σi for $\Sigma\{i\}$).

$$\begin{aligned}
&\mathrm{MEC}_L(n)_0 = (\Sigma i : \varnothing : 1) = \infty \text{ (unbounded sum)}\\
&\mathrm{MEC}_L(n)_1 = (\Sigma i : i = 0 : 1) = 1\\
&\mathrm{MEC}_L(n)_2 = (\Sigma i : i \geq 0 : 1) = \infty\\
&\mathrm{MEC}_L(n)_3 = (\Sigma i : i \geq 0, i \geq n+1 : 1) = \infty\\
&\mathrm{MEC}_L(n)_4 = (\Sigma i : 0 \leq i \leq n : 1) = (\Sigma : n \geq 0 : n+1)\\
&\mathrm{MEC}_L(n)_5 = (\Sigma i : 1 \leq i \leq n+1 : 1) = (\Sigma : n \geq 0 : n+1)
\end{aligned} \tag{2}$$

Now MEC_L maps the value of n to the number of possible states (number of points inside polyhedra) at each program point of the program.

3.3. Parametric IPET

Parametric Integer Programming [9] is essentially a parameterized version of the (dual) simplex algorithm and may be used to solve parametric IPET problems. A tool called PipLib exists [14], it can be used to solve a parametric IPET problem, where the answer is given in terms of the parameters. With structural constraints $A\vec{x} \leq \vec{b}$ and atomic WCETs from the low-level analysis $\vec{c}$, we can use PipLib to maximise

$$\vec{c}^{\mathrm{T}}\vec{x} \text{ subject to } A\vec{x} \leq \vec{b} \text{ and } \vec{x} \leq \vec{p}$$

where $\vec{p}$ is a vector of parameters where each component p_i corresponds to a symbolic upper bound to the execution count of x_i. PipLib gives as solution a so-called *quast*, which is a formula expressing the unknown variables in terms of the parameters. The quast is a tree of nested if-statements where the leaves correspond to solutions for the unknown variables. Both the conditionals and the solutions are linear expressions of the parameters, see [14] for further information. The quast can be used to obtain PC_P.

If we assume that each program point of L in Figure 2 has a constant WCET of ten clock cycles then we want to maximise $y = 10\sum x_i$ subject to the structural constraints of L and $x_i \leq p_i$. Using PipLib to solve this IPET problem will give us PC_L:

$$\mathrm{PC}_L = \lambda(p_0, p_1, p_2, p_3, p_4, p_5).\mathrm{if} p_2 \leq p_4 + 1 \text{ then}(\mathrm{if} p_2 \leq p_5 + 1 \text{ then } 30p_2 + 10 \text{ else...} \tag{3}$$

The full quast contains eight leaves and is too big to show here. The reason for the many leaves is that the possible relations among the parameters are many.

3.4. Combination

Finally, we need to express the WCET estimation as a parametric formula in terms of the input parameters in $\mathcal{I}_P$, formally computing the function $\mathrm{PWCET}_P = \mathrm{PC}_P \circ \mathrm{MEC}_P$. This is done by parsing the solution quast given from PipLib (corresponding to PC_P) and substituting the parameters $\vec{p}$ with $\mathrm{MEC}_P(\vec{i})$, resulting in a formula parameterized in $\mathcal{I}_P$.

The final result of the analysis for L will after combining (3) and (2) is,

$$\mathrm{PWCET}_L = \lambda n.\mathrm{if} c_2 \leq c_4 + 1 \text{ then}(\text{if } c_2 \leq c_5 + 1 \text{ then } 30c_2 + 10 \text{ else...}$$

where $c_k = \mathrm{MEC}_L(n)_k$. Since $c_5 = c_4$ and $c_1 = 1$ we can see that simplification is needed, but not yet implemented.

4. Symbolic Counting

The method in [15] gives a symbolic formula of the number of solutions to a given presburger formula. Convex polyhedra (or equivalently, systems of linear inequalities) are a strict subset of presburger formulae and we will show to restrict and simplify the method for this special case. We will assume two restrictions of the generalised sums; first, rather than having the guard as a presburger formula, we have it as a system of linear inequalities in the variables of $\mathcal{V}_P$ (since this is exactly what the polyhedral abstract interpretation will give). The other restriction is that we model the formulae to sum over as polynomials, simplifying both representation and computation. Polynomials can easily be modelled as a sum of terms, where a term is a vector representing an integer coefficient and variable powers. As an example we can model the polynomial $3a^2b^3 + 5a^4$ (assuming $\mathcal{V}_P = \{a, b\}$) as the sum of the terms $(3\ 2\ 3)$ and $(5\ 4\ 0)$. This also makes arithmetical operations on these vectors straightforward to implement. Furthermore, since the guards are polyhedra, the lower and upper bound of any variable will be sets of linear expressions. Summing a linear expression over a polynomial is again a polynomial, so this model is closed under summations. However, these restrictions sometimes require the result to be slightly over-approximated. The constraint $3a - b \leq 0$ gives an upper bound for a as $a \leq \lfloor \frac{b}{3} \rfloor$, since a and b are integers. As seen, the upper bound is not a polynomial and therefore problematic in our model. Since $\frac{b}{3}$ is a safe upper bound for $\lfloor \frac{b}{3} \rfloor$ and on polynomial form we can use it as approximation. We handle lower bounds accordingly.

5. Reducing the Number of Variables

As seen above, we need an unknown variable and an execution count parameter for each program point in the program in the parametric IPET calculation. Parametric ILP has exponential complexity in the number of variables in the worst-case, making scalability problematic. The structural constraints of the program in general produce an under determined system. In an under determined system with n variables, the solution space is the span of a set of $n-r$ vectors (where r is the rank of the constraint matrix). The variables can be expressed as linear combinations of these vectors and we can only solve the problem in terms of these. Let $A\vec{x} = \vec{b}$ be a system of structural and possible other linear equations obtained from flow analyses and $y = \vec{c}^{\mathrm{T}}\vec{x}$ be the cost function. The constraints together with the cost

function is

$$\begin{pmatrix} 1 & -\vec{c} \\ 0 & A \end{pmatrix} \begin{pmatrix} y \\ \vec{x} \end{pmatrix} = \begin{pmatrix} 0 \\ \vec{b} \end{pmatrix}$$

If we perform Gauss-Jordan elimination on the above (including the right-hand side by augmenting the constraint matrix by $(0\ \vec{b})^{\mathrm{T}}$) and re-arrange the columns of A and the components of $\vec{x}$ such that all pivot columns are to the left, and $\vec{x}$ is re-arranged accordingly, we get

$$\begin{pmatrix} 1 & 0 & -\vec{c'} \\ 0 & I_r & A' \end{pmatrix} \begin{pmatrix} y \\ \vec{x}_{\mathrm{BV}} \\ \vec{x}_{\mathrm{FV}} \end{pmatrix} = \begin{pmatrix} z \\ \vec{b'} \end{pmatrix}$$

where I_r is the $r \times r$ identity matrix and r is the rank of A. Furthermore, z is the last column of the solution of the augmented matrix after elimination. The vector $\vec{x}$ has now been partitioned into a vector of r basic variables $\vec{x}_{\mathrm{BV}}$ which corresponds to the columns of I_r and the vector $\vec{x}_{\mathrm{FV}}$ of $n - r$ free variables (where n is the number of columns of A) which corresponds to the columns of A'. Note that this transformation also removes any redundant constraints from the system. From this we can derive two important equations. One is the objective function expressed in terms of the free variables

$$y = z + \vec{c'}\vec{x}_{\mathrm{FV}}$$

and a way to express the basic variables in terms of the free ones

$$\vec{x}_{\mathrm{BV}} = -A'\vec{x}_{\mathrm{FV}} - \vec{b'}$$

As we model the parametric upper bounds as the constraints $\vec{x} \leq \vec{p}$, we can now simply model our IPET problem as

$$\begin{pmatrix} z + \vec{c'}\vec{x}_{\mathrm{FV}} \\ -A'\vec{x}_{\mathrm{FV}} - \vec{b'} \\ \vec{x}_{\mathrm{FV}} \end{pmatrix} \leq \begin{pmatrix} y \\ \vec{p}_{\mathrm{BV}} \\ \vec{p}_{\mathrm{FV}} \end{pmatrix}$$

where we have partitioned and re-arranged $\vec{p}$ exactly as for $\vec{x}$. Now it suffices to solve the IPET with these constraints, thus reducing the number of unknowns by the rank of A. Note that we cannot reduce the number of parameters in the same way, since they directly correspond to the output of MEC_P. This method of eliminating variables is not restricted to the parametric case, but can be used to reduce the dimensionality of any IPET problem. As an example, using this on (3) reduces the system from eight to two unknown variables (the variable of the cost function included).

6. The Result of Parametric Integer Programming

The possible number of relations between parameters in the IPET calculation are large, and yields large and complex quasts as result, even for small program examples (as seen in Section 3.3). Calculating WCET_P as a composition of the quasts with the result of the symbolic counting will substitute guarded generalised sums for all parameters in the quasts and the final result will be very complex and large. This suggests that a simplification of PC_P is needed as well as of the final formula PWCET_P. As presented in Section 5, we already simplified the problem by reducing the number of unknowns, but there are more possibilities for simplifications. A way to reduce the number of possible relations among parameters would be to restrict the parameters in the IPET calculation with constraints. Restrictions of parameters could potentially be found directly from MEC_P (e.g. some counts may impose absolute upper and lower bounds on parameters) and in some cases from the CFG structure (e.g. start and stop nodes restrict the parameters). Some parameters could also potentially be eliminated. The final PWCET_P obtained after substitution can be simplified by finding equal sub trees and eliminating tautologies and contradictions in the conditionals.

7. Related Work

The analysis in [8] is as [12] based on calculating the number of run-time states to find the WCET but without parameters. An approach to parametric WCET analysis was presented in two joint master theses [1, 11] and also in [2]. The analysis of these theses uses PipLib as tool for a parametric IPET calculation but uses parameters only for loop bounds. The loop bound parameters is then substituted for parameterized intervals. The parameterized intervals are obtained by identifying loop counter variables and loop invariants based on their relative values to input parameters of the program. Other approaches to parametric WCET analysis can be found in [3, 5, 16].

Abstract interpretation [6] is a general theory for soundly abstracting program semantics and is commonly used in static analysis. The polyhedral domain [7] has been used in different contexts than this, such as in verification of linear hybrid systems in [10]. Different methods for symbolic counting are [15, 4]. Parametric integer linear programming was first presented in [9], see also [14].

8. Conclusions and Future Work

We have done a prototype implementation of the framework for parametric WCET analysis presented in [12]. The framework was implemented for simple CFGs. The prototype is in the current state not powerful enough to fully evaluate the method, but shows that the method successfully can be implemented, mostly by using existing libraries. We have presented a simplified method for symbolically counting integer points inside convex polyhedra based on the method in [15]. We have also presented a general method to reduce the number of variables used in a IPET calculation where linear equations are used as constraints. We conclude that an important challenge of making a successful implementation for real programs is to deal with complexity; both computational and representational. The final WCET formula needs to be simplified in several steps. As future work, the prototype will be enhanced to be able to deal with more realistic program (by adding arrays and pointers) in order to evaluate and make comparative studies on concrete benchmarks and to compare the precision/complexity trade-off by using different abstract domains and calculation methods.

References

[1] S. Altmeyer. Parametric wcet analysis, parametric framework and parametric path analysis. Master's thesis, Saarland University, Department of Computer Science, Oct 2006.

[2] Sebastian Altmeyer, Christian Hümbert, Björn Lisper, and Reinhard Wilhelm. Parametric timing analysis for complex architectures. In *RTCSA '08: Proc. 14th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications.*, 2008.

[3] G. Bernat and A. Burns. An approach to symbolic worst-case execution time analysis. In *Proc. 25th Workshop on Real-Time Programming*, Palma, Spain, May 2000.

[4] Philippe Clauss. Counting solutions to linear and nonlinear constraints through ehrhart polynomials: Applications to analyze and transform scientific programs. In *International Conference on Supercomputing*, pages 278–285, 1996.

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

[6] Patrick Cousot and Radhia Cousot. Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Proc. 4th ACM Symposium on Principles of Programming Languages*, pages 238–252, Los Angeles, January 1977.

[7] Patrick Cousot and Nicholas Halbwachs. Automatic discovery of linear restraints among variables of a program. In *Proc. 5th ACM Symposium on Principles of Programming Languages*, pages 84–97, 1978.

[8] Andreas Ermedahl, Christer Sandberg, Jan Gustafsson, Stefan Bygde, and Björn Lisper. Loop bound analysis based on a combination of program slicing, abstract interpretation, and invariant analysis. In *Proc. 7^{th} International Workshop on Worst-Case Execution Time Analysis, (WCET'2007)*, July 2007.

[9] P. Feautrier. Parametric integer programming. *Operationnelle/Operations Research*, 22(3):243–268, 1988.

[10] Nicholas Halbwachs, Yann-Eric Proy, and Pascal Raymond. Verification of linear hybrid systems by means of convex approximations. In Baudouin Le Charlier, editor, *Proc. International Symposium on Static Analysis*, Vol. 864 of *Lecture Notes in Comput. Sci.*, pages 223–237, Namur, September 1994. Springer-Verlag.

[11] C. Humbert. Parametric wcet analysis, parameter analysis and parametric loop analysis. Master's thesis, Saarland University, Department of Computer Science, Oct 2006.

[12] Björn Lisper. Fully automatic, parametric worst-case execution time analysis. In Jan Gustafsson, editor, *Proc. 3^{rd} International Workshop on Worst-Case Execution Time Analysis, (WCET'2003)*, pages 77–80, Porto, July 2003.

[13] New polka webpage, 2008. `http://pop-art.inrialpes.fr/people/bjeannet/bjeannet-forge/newpolka/index.html`.

[14] Piplib website, 2008. `http://www.piplib.org/`.

[15] William Pugh. Counting solutions to Presburger formulas: How and why. In *SIGPLAN Conference on Programming Language Design and Implementation*, pages 121–134, 1994.

[16] E. Vivancos, C. Healy, F. Mueller, and D. Whalley. Parametric timing analysis. In Jay Fenwick and Cindy Norris, editors, *Proc. ACM SIGPLAN Workshop on Languages, Compilers and Tools for Embedded Systems (LCTES'01)*, pages 88–93, Snowbird, Utah, June 2001.