The W-SEPT Project: Towards Semantic-Aware WCET Estimation

Claire Maiza[1], Pascal Raymond[2], Catherine Parent-Vigouroux[3], Armelle Bonenfant[4], Fabienne Carrier[5], Hugues Cassé[6], Philippe Cuenot[7], Denis Claraz[8], Nicolas Halbwachs[9], Erwan Jahier[10], Hanbing Li[11], Marianne De Michiel[12], Vincent Mussot[13], Isabelle Puaut[14], Christine Rochange[15], Erven Rohou[16], Jordy Ruiz[17], Pascal Sotin[18], and Wei-Tsun Sun[19]

1 **Grenoble Alps University, VERIMAG, Grenoble, France**
claire.maiza@univ-grenoble-alpes.fr
2 **Grenoble Alps University, VERIMAG, Grenoble, France**
pascal.raymond@univ-grenoble-alpes.fr
3 **Grenoble Alps University, VERIMAG, Grenoble, France**
catherine.parent-vigouroux@univ-grenoble-alpes.fr
4 **IRIT, University of Toulouse, Toulouse, France**
bonenfant@irit.fr
5 **Grenoble Alps University, VERIMAG, Grenoble, France**
fabienne.carrier@univ-grenoble-alpes.fr
6 **IRIT, University of Toulouse, Toulouse, France**
Hugues.Casse@irit.fr
7 **Continental AG, Toulouse, France**
philippe.cuenot@continental-corporation.com
8 **Continental AG, Toulouse, France**
denis.claraz@continental-corporation.com
9 **Grenoble Alps University, VERIMAG, Grenoble, France**
nicolas.halbwachs@univ-grenoble-alpes.fr
10 **Grenoble Alps University, VERIMAG, Grenoble, France**
erwan.jahier@univ-grenoble-alpes.fr
11 **Inria, IRISA, Rennes, France**
hanbing.li@inria.fr
12 **IRIT, University of Toulouse, Toulouse, France**
Marianne.De-Michiel@irit.fr
13 **IRIT, University of Toulouse, Toulouse, France**
mussot@irit.fr
14 **University of Rennes 1, IRISA, Rennes, France**
isabelle.puaut@irisa.fr
15 **IRIT, University of Toulouse, Toulouse, France**
rochange@irit.fr
16 **Inria, IRISA, Rennes, France**
Erven.Rohou@inria.fr
17 **IRIT, University of Toulouse, Toulouse, France**
Jordy.Ruiz@irit.fr
18 **IRIT, University of Toulouse, Toulouse, France**
Pascal.Sotin@irit.fr
19 **IRIT, University of Toulouse, Toulouse, France**
wsun@irit.fr

17th International Workshop on Worst-Case Execution Time Analysis (WCET 2017).
Editor: Jan Reineke; Article No. 9; pp. 9:1–9:13

Open Access Series in Informatics
OASICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Abstract

Critical embedded systems are generally composed of repetitive tasks that must meet hard timing constraints, such as termination deadlines. Providing an upper bound of the worst-case execution time (WCET) of such tasks at design time is necessary to guarantee the correctness of the system. In static WCET analysis, a main source of over-approximation comes from the complexity of the modern hardware platforms: their timing behavior tends to become more unpredictable because of features like caches, pipeline, branch prediction, etc. Another source of over-approximation comes from the software itself: WCET analysis may consider potential worst-cases executions that are actually infeasible, because of the semantics of the program or because they correspond to unrealistic inputs. The W-SEPT project, for "WCET, Semantics, Precision and Traceability", has been carried out to study and exploit the influence of program semantics on the WCET estimation. This paper presents the results of this project : a semantic-aware WCET estimation workflow for high-level designed systems.

1998 ACM Subject Classification F.3.2 Semantics of Programming Languages, D.2.2 Design Tools and Techniques, D.2.4 Software/Program Verification, D.2.5 Testing and Debugging, C.3 Special-Purpose and Application-Based Systems, B.4.4 Performance Analysis and Design Aids

Keywords and phrases Worst-case execution time analysis, Static analysis, Program analysis

Digital Object Identifier 10.4230/OASIcs.WCET.2017.9

1 Introduction

Critical embedded systems are generally composed of repetitive tasks that must meet strong timing constraints, such as termination deadlines. Providing an upper bound of the worst-case execution time (WCET) of such tasks at design time is necessary to guarantee the correctness of the system.

Test based methods, widely used in practice, provide actual execution times but cannot guarantee that the worst case has been reached. Static analysis methods aim at providing a guaranteed upper bound to the WCET, by considering an abstract model of the program execution. In order to be safe, and also to keep the analysis tractable, the models are necessarily pessimistic and often lead to a possibly large over-approximation of the WCET.

In static WCET estimation, a main source of over-approximation comes from the complexity of the modern hardware platforms: their timing behavior tends to become more unpredictable because of features like caches, pipeline, test prediction, etc. Another source of over-approximation comes from the software itself: WCET analysis may consider potential worst-case executions that are actually infeasible, because of the semantics of the program or because they correspond to unrealistic inputs.

For instance, in the automotive application (Engine Management System : EMS) of Continental Corporation the modules of the application are mostly implementing generic algorithms that use calibration data for possible adaptation : a worst-case path could correspond to an unrealistic system state like high-engine speed with low-injection set point.

In the classical WCET estimation framework, the *data-flow analysis* is in charge of discovering infeasible execution paths. It must at least provide constant bounds for all the loops in the program, otherwise a finite WCET is not even guaranteed to exist. Apart from loop-bounds, control-flow analysis usually identifies simple semantics properties such as tests exclusions, that may prune infeasible execution paths when computing the WCET.

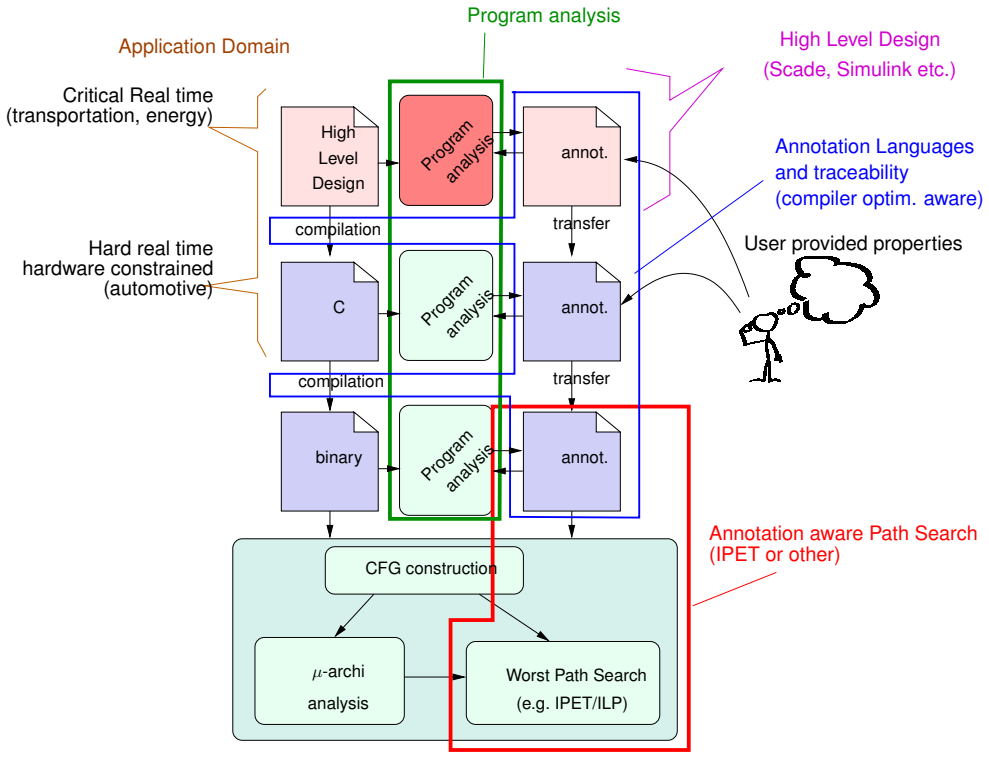

Figure 1 Work-flow and general organization of a semantic aware WCET estimation tool.

The W-SEPT project, for "*WCET, Semantics, Precision and Traceability*", has been carried out to study and exploit the influence of program semantics on the WCET estimation. This paper presents the results of this project: a semantic-aware WCET estimation workflow for high-level designed systems.

1.1 Workflow of a semantic-aware WCET estimation tool

The goal of the W-SEPT project[1] was to define and prototype a complete semantic-aware WCET estimation workflow [1]. It gathers researchers in the domain of timing and program analysis, together with an industrial partner from the real-time domain. The project mainly focuses on the semantic aspects, and thus, the pruning of infeasible paths. As far as possible, the idea is to extend and adapt the classical WCET estimation workflow. In particular, all that concerns the hardware analysis is inherited from previous work, through the use of the tool OTAWA[2].

This paper summarizes the main achievements of the project. We give the general picture: more details can be found in referenced papers. These achievements are structured according to the general workflow of the project, depicted by Figure 1.

It retains the general organization of classical existing tools [24]. The bottom block is the WCET computation tool itself, organized in three steps: Control-Flow graph (CFG) construction, micro-architecture analysis, and worst-path search on the CFG. Generally, this

[1] `http://wsept.inria.fr`
[2] `http://www.otawa.fr`

last step uses the classical Implicit Path Enumeration Technique (IPET) [14]. This WCET estimation takes as input the binary code of the program, and a set of semantic informations classically named *annotation file*, and containing at least the loop bounds.

These binary annotations come from program analysis. This analysis is generally performed at the source level, C language most of the time, rather than at the binary level. Indeed, analyzing C code is technically much simpler than analyzing binary code, but more importantly, the analysis often requires extra information that only the user can provide (e.g., inputs ranges, exclusion, implications). The user can probably express these properties in terms of the C variables, but it would be much harder or even impossible to do it in terms of the compiled binary code. This two-layers description raises the well-known problem of *traceability* of annotations when transferring information between layers.

So far, the principles depicted in Figure 1 are rather classical. An innovation of the project was to take into account a third layer in the design flow: the use of high-level design languages that are common in the domain of (critical) real-time applications. Classical examples of high-level design tools are Scade suite[3], used in avionics, energy or transportation, and Simulink/Stateflow[4] widely used in control engineering systems. These high-level design tools provide automatic code generation to C, which is no longer the source code, but only an intermediate code. A consequence is that user annotations and program analysis can be expressed and performed at the design level. The coupling of timing analysis and high level design is not new in itself. For instance the tool aIt (from the Absint company) has been coupled with the Scade Tool Suite[5]. However, this integration does not consider the extraction and exploitation of properties at the Scade level for enhancing the analysis of aIt.

The project proposed to focus on three main issues depicted by enclosing boxes in Figure 1:

- Program analysis, that can be performed at high-level design, C or binary level, and may take into account information provided by the user.
- Annotations and traceability between the language levels, which strongly involve the compilers: as far as possible, the compilation process should be annotation-aware, in the sense that the program transformations performed by the compiler should be reflected as annotation transformations.
- WCET estimation tool and the worst-path search step, must be adapted to take into account the richer kind of annotations produced by the workflow.

In this summary, we briefly describe the obtained results concerning each step of this workflow. In Section 2, we present how, at each stage, we can generate properties (automatically extracted) in order to discard infeasible paths. Then we show how to annotate these properties and automatically translate them through the compilation process, in Section 3. In Section 4, we describe how an existing WCET estimation tool was adapted in order to exploit this new kind of annotations. Even enhanced thanks to semantic information, the WCET estimation is still necessarily a pessimistic upper bound. Section 5 presents two methods for assessing the pessimism, by finding a guaranteed (big) lower bound, and thus an interval containing the actual WCET. Indeed, the smaller is the interval, the better is the estimation.

[3] `http://www.esterel-technologies.com/products/scade-suite`
[4] `http://mathworks.com/products/simulink/`
[5] `http://www.absint.com/ait/scade.htm`

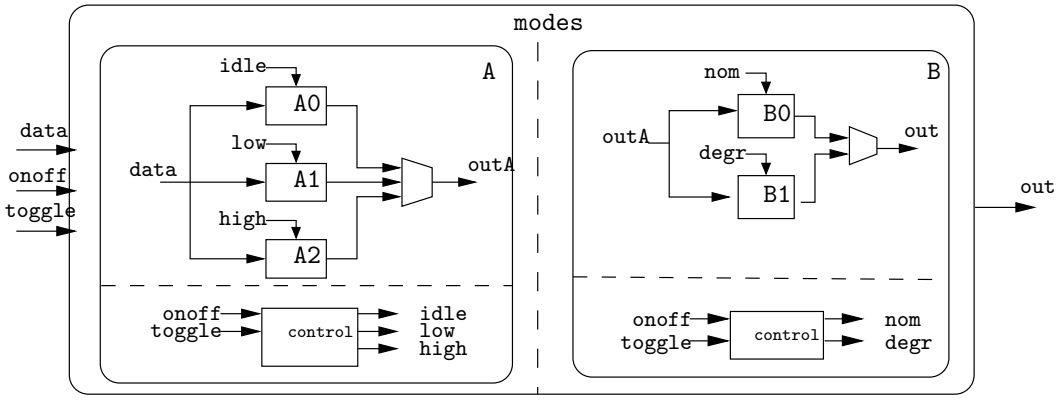

Figure 2 A typical high-level dataflow design.

2 Extraction of semantic properties

In this section, we explain what kind of semantic properties may help to enhance the WCET estimation: where do they come from and which step of the application development do they refer to (binary, code, design). We consider the automatic extraction of properties. The set of analyses should lead to a cumulative improvement as the kind of properties they cover should be exclusive. The annotation language, necessary to express and transfer user assumptions and discovered properties, is presented in the next section.

2.1 High-level properties

Critical embedded systems are often designed using high level modeling languages, such as Scade or Simulink. The system is then automatically compiled into classical imperative code (C in general), and then into binary code (cf. Fig 1).

Figure 2 shows a typical high-level data-flow design. For simplicity, it is represented as a diagram, while the actual program is written in Lustre [7], the academic textual language which is the ancestor of the industrial Scade language. This application consists of two sub modules, `A` and `B`, each of them consisting in two parts: a control part and a data processing part. The data processing part has different computation modes (e.g., `A0`, `A1` and `A2`), controlled by a *clock* (e.g., `idle`, `low` and `high`). An important property of such a design is that these modes are exclusive: at each reaction exactly one of the modes is activated. This information, obvious at the design level, may or may not be obvious at the C or binary level: depending on the compilation process, the (high level) mode exclusion may result or not into structurally exclusive pieces of code. In a more subtle way, we also know, for this particular program, that there is a logical exclusion between the modes of the two sub-modules: if `A` is not `idle` (`A1` or `A2`), then `B` is necessarily in degraded mode (`B1`). This property is neither structural nor obvious: it is an *invariant* of the infinite cyclic behavior of the application that holds if we suppose `toggle` and `onoff` are never true at the same time (which is an hypothesis on the system environment). It is therefore almost impossible to discover it at the low-level.

Based on these remarks, we have developed a prototype for discovering such properties, propagate them through the compilation process, and exploit them to enhance the WCET estimation. Details on how these properties are transferred and used to enhance the WCET are in Section 3.2 and 4.1.

To extract properties that may reveal infeasible paths, we identified the high-level expression that influences a branch at binary level. In a second step, we use a model-checker (Lesar [18]) to check the validity of properties at the Lustre level. We use two heuristics:

- a pairwise algorithm: The high level code is analyzed to find a set of interesting control variables, according to a simple heuristic: any Boolean variable that controls computation modes (often called the *logical clocks*) is likely to control big pieces of binary code, and thus, has a big influence on the computation time. In the example, the five control variables are selected. We "blindly" search for all possible pairwise relations (either exclusions or implications) between these variables. For n variables, there are $4(n*(n-1)/2) = 2n(n-1)$ such (potential) relations (40 in the example). For each relation proven by Lesar, we generate the corresponding constraints at the binary level thanks to the traceability information; in the example, 5 over 40 relations are proven.
- an iterative algorithm: according to the traceability information, the validity of the worst-case path is translated (if possible) into a logical condition on the high-level variables (e.g. $\neg$`idle` $\wedge$ `low` $\wedge$ `nom`); Lesar is called to check this condition; if the condition is unsatisfiable, the WCET path candidate is proven unfeasible, the corresponding constraint gives an infeasible path that we give to the WCET estimation tool; we restart to find a new candidate, and so on. If the condition is found satisfiable, the process stops with the current WCET.

The improvement on the WCET estimation is important and similar for both strategies (up to 50% on a realistic Lustre benchmark). The iterative algorithm may be relatively costly. The pairwise strategy has a constant overhead. The whole experiment is presented in details in [20].

2.2 C level properties

The discovery of bounds and relations on numerical variables is a classical goal in program analysis [5]. These bounds and relations can obviously be used to restrict the set of feasible paths considered in WCET evaluation. This can be helped by adding some counters to the code of the program: of course, adding a loop counter may result in finding a bound to this counter, and thus to the iteration number. Moreover, adding block counters, and finding relations between these counters can reveal subtle restrictions in the possible executions of the program.

An analysis of this instrumented program with counters using an analyser of linear relations (here, we used the tool PAGAI [8]), automatically discovers some linear relations on counters. This approach has been implemented in a prototype tool [2], and applied in combination with OTAWA to several examples. Results show improvements of the evaluated WCET (with or without counters) up to 50% on TACLeBenchs[6].

2.3 Low-level properties

Looking for infeasible paths at binary level benefits from the exact matching of the program with the hardware, and to inject found properties immediately in the WCET computation. The price is an increase of the analysis time caused by the program size and the loss of expressivity implied by machine instructions. Consequently, existing analyses either look for very simple infeasible paths [6, 22], or design a new WCET computation method [22]. Our

[6] `http://www.tacle.eu/index.php/activities/taclebench`

approach tries to get rid of these limitations by using SMT solvers (Satisfiability Modulo Theories) to generate infeasible path properties [21]. This approach finds a large set of infeasible paths on the TACLeBenchs: it cuts from 1 to some thousands of edges in the control flow graph.

2.4 Delta-guided extraction

In order to lower the real WCET, some approaches compute execution time profiling (using the estimation of program part execution time with respects to the global WCET) [3] or generate a static profile using probabilities for decisions at branching points [25]. The delta tool [27] aims at identifying the conditional statements that are unbalanced in terms of execution time weight (obtained so far by a naive counting of instructions). This highlights, to the user or the program analyzers, the parts of code where a semantic analysis or user annotation should focus to gain more accuracy on the WCET estimation. Branching statement analysis allows identifying parameters as important or not due to their unbalanced weight.

This method may be combined with any of the extraction method presented in this section. For instance, it may help reducing the number of pairwise properties to check at the high-level: if the validity of a pairwise property does not influence the WCET, there is no need to call the model-checker.

3 Annotation language and traceability

In order to express most of the properties, we use and extend the existing FFX annotation language [26]. FFX is an open, portable, and expandable annotation format. It allows combining flow fact information from different high-level tools. It is used as an intermediate format for WCET analysis; in particular it is both the source and target language for the traceability tools, that transfer information from one level to the next one.

3.1 Annotation language

The principle of FFX is to express a wide class of information that may be helpful to compute or enhance the WCET estimation by Otawa. It is specifically dedicated to sequential programs (C or binary), and allows expressing both data-flow and control-flow properties.

- Data-flow: FFX allows one to identify data and express properties such as the type, the range, the mutability status (local, global, input or output); such information is given "as is" to Otawa and, may (or may not) be used for the computation of the WCET.
- Control-flow: FFX allows one to identify control points and express constraints and relations between them. Control points are typically identified by line numbers in C code, or with address offsets in binary code. Classical flow information concerns the maximum occurence number of a program point (loop bound), the fact that a branch is always or never taken etc.

The FFX language and associated tools have been extended and adapted to meet the goals of the project:

- Transfer of ILP constraints: the principle of control point counters and constraints, already existing for expressing the loop bounds has been extended to any kind of linear relations between counters. This way, flow information discovered by analyzing tools (cf. Section 2) can be directly transferred to the worst-path analysis module.
- Logical paths constraints: the language has been extended to express path properties in a more *logical* way; the exclusion between several control points within a particular

scope. This kind of information can be translated later into classical ILP constraints, or be handled by the new concept of *Path Property Automata* (PPA), as presented in the next Section 4.

- Expression of specific scenarios: a strong requirement from the industrial partner is to be able to evaluate the WCET under some specific use cases. This notion is different from the classical constraints, since scenarios can contradict each other, and a single FFX file may contain several scenarios. Since FFX is not designed to be used by humans, a mini language of user annotations has been designed to be used directly as "pragmas" in the C code. These source-code annotations are extracted and automatically translated in FFX.

3.2 Traceability

From design level to source code, we transfer the properties by tracing them in the code generator (by inserting additional comments in the C code).

From C to binary, hundreds of compiler optimizations may have a strong impact on the structure of the code, making it impossible to match source-level and binary-level control flow graphs. This ends up in a loss of useful information. For this reason, the current practice is to turn off compiler optimizations, resulting in low average-case and worst-case performance. To safely benefit from optimizations (as in [11]), we propose a framework to trace and maintain flow information up-to-date from source code to machine code [12].

The transformation framework, for each compiler optimization, defines a set of formulas, that rewrite available semantic properties into new properties depending on the semantics of the concerned optimization. Supported semantic properties are *loop bounds* and linear inequations constraining the execution counts of basic blocks. Consider, for example, loop unrolling, that replicates a loop body k times to reduce loop branching overhead and increase instruction level parallelism. The associated rewriting rule divides the initial loop bound by k, and introduces constraints on the execution counts of the basic blocks within the loop (see [12, 13] for details).

We have implemented this traceability in the LLVM compiler infrastructure (local patches). Each LLVM optimization was modified to implement the rewriting rules corresponding to the optimization. Semantic information is initially read from a file in the FFX format, and then represented internally in the LLVM compiler, and finally transformed jointly with the code transformations. Note that, if a transformation happens to be too complex to trace the information, it can be disabled. This is a better situation than the general current practice which is disabling all optimizations.

4 Exploitation of semantic properties in WCET estimation

Sections 2 and 3 respectively presented how properties are extracted, expressed and traced. This section presents how the properties are taken into account in the WCET analysis.

There are basically two ways for handling infeasible path properties in WCET analysis. The explicit way proceeds by control-flow graph transformations and aims at pruning (any) infeasible paths from the model. This general method is virtually able to handle any kind of pruning properties, but may lead to the explosion of the model size. The implicit way, as formalized in the classical IPET/IPET method, prevents the model size explosion by summarizing "big" families of infeasible paths with "small" numerical relations. In the project, we have first considered the implicit way, and then proposed a more versatile method allowing a mix between explicit and implicit exploitation

4.1 Exploitation in ILP

In many cases, the properties found at the C or high level can be expressed relatively directly into ILP constraints, exploitable in the classical IPET framework.

This is typically the case for *branch conflicts* properties: a conflict is a set of control points in the program that cannot be all taken during the same execution. Expressing simple conflict patterns in ILP is rather classical, and numerous examples can be found in literature. For instance, a conflict between three control points a, b, c, within a loop bounded by the constant n, can be expressed with: $a + b + c \leq 2 \times n$.

We have presented in [19] a method that generalizes the encoding of conflicts with ILP constraints. In contrast with many existing approaches, it is not based on patterns, but on the ability of counting how many times a conflict occurs. Thanks to this principle, one can handle for instance conflicts occurring between different loop scopes. Consider for instance the following program structure, with two nested loops:

```
for i = 1 to n do
   if ... then a else ...
   for j = 1 to m do
      if .. then b else ...
```

and suppose that some analysis has established that, if a is taken during one outer iteration, b cannot be taken during the whole forthcoming inner loop. Our method gives that this particular conflict can be expressed with the following ILP constraint:

$$m \cdot a + b \leq n \cdot m\,.$$

The main advantage of the ILP encoding is that the results of semantic analysis can be directly exploited into existing WCET tools based on the Implicit Path Enumeration Technique. However, even if the ILP encoding is safe (only infeasible paths are pruned), it is not necessarily exact (it may still accept some infeasible paths).

4.2 Exploitation through automata

We propose a general, versatile, and non-intrusive process for the integration of the paths properties [15, 16, 17]. This process assumes that the WCET tool internally handles CFGs and integer linear constraints, which is the case of every IPET-based WCET analyzers. The internal representation of the program is extracted, improved according to the annotations and set back in the tool. The transformation relies on a novel automata-based formalism that can represent both the program CFG and the annotations. The transformation itself is defined as an automata product; it results is an automaton whose paths are both existing in the original CFG and valid with respect to the annotations[7].

Figure 3 shows two Path Property Automata (PPA). On the left, a PPA isomorphic to a program CFG. On the right, a corresponding PPA that additionally reflects the property "in each iteration of the loop starting with E and ending with X, at most one of A or B can be taken".

Note that our formalism mixes explicit (state/transition) and implicit (local counter α, bounded by one) concerns, and can be exploited accordingly. The purely explicit exploitation consists in producing a flat product of the graph and the property automaton. In this case,

[7] http://www.mrtc.mdh.se/projects/WTC/

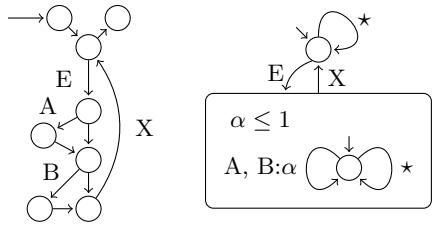

Figure 3 Path Property Automaton.

the resulting modified control graph is a graph where the core of the iteration is replaced by 3 exclusive paths: (1) A is executed and B is not, (2) A is not executed and B is, (3) none of them are executed.

In order to limit the size explosion, we have also defined a specific product that keeps, as far as possible, the counter constraints implicit. In this particular example, the algorithm results on an unchanged control graph decorated with constraints: $\alpha = A + B \quad \alpha \leq E$, which are equivalent to the classical conflict constraint $A + B \leq X$.

The analysis performed on the enriched CFG delivers a WCET improvement up to 10% on the benchmarks of the WCET Tool Challenge.

5 Assessment of WCET estimations

The approaches presented in this paper aim at increasing the precision of WCET estimations by enhancing the knowledge of possible execution paths. A reduction on the estimated WCET reflects a tighter analysis due to taking the program semantics into consideration. However, whether the new estimation is far from or close to the real WCET remains unknown. In order to evaluate the precision of static WCET analysis, two approaches have been studied. The first approach is based on simulation, and aims to assess the precision by comparing the estimated WCET with the longest observed execution time. The second approach is integrated to the estimation process, and aims at self-assessing the pessimism of the static analysis.

5.1 Simulation-based assessment

We have developed a simulator (OSIM [23]) that uses the same hardware model as OTAWA, in order to provide execution times that are consistent with those provided by the static analysis. Therefore, for a given program, the simulation provides a *guaranteed lower bound*, M_{WCET}, that forms with the guaranteed upper bound E_{WCET} an uncertainty interval. The relative size of this interval can be measured by an *over-estimation ratio*: ρ=(E_{WCET}- M_{WCET}) / M_{WCET}. The first way to reduce ρ is to decrease E_{WCET} with more precise static analysis. The second way is to increase the $M_{WCET}$with more thorough test generation.

In the case of reactive programs, which continuously interact with their environment, performing intensive test generation requires to simulate environments that can react to the system outputs. This is achieved via the use of LUTIN [9, 10], a language for specifying and playing random constrained reactive scenarios.

The scenario written for simulation must at least integrate the same hypothesis as the static analysis. If it is not the case, M_{WCET} and E_{WCET} are considering different sets of realistic executions, and their results are not inconsistent. Consider for instance, the example

in Section 2.1: the estimation E_{WCET} is obtained under the assumption that the inputs `toggle` and `onoff` are exclusive. When simulating the program without this assumption we obtain a M_{WCET} which is 60% greater than the E_{WCET} [23].

Writing in Lutin a random scenario, that takes only input constrains into account, requires very little effort. In many cases, such a simple scenario gives good results. For instance, for the example of Section 2.1, we have obtained this way a ratio ρ of about 5%.

In some cases, simple random testing is not sufficient to get close to the worst case, and the user expertise is necessary to write more sophisticated scenarios, that drive the generation to uncommon and costly executions.

5.2 Quantification of static analysis pessimism

The simulation-based approach provides a safe upper bound on the pessimism of the estimated WCET, but it does not give any insight into the sources of pessimism: it could be due to highly dynamic hardware schemes, the behavior of which must be approximated at analysis time, or to under-specified flow information.

In [4], we introduced a framework that extends static WCET analysis to quantify the possible overestimation. The approach consists in identifying, during the analysis, whether the intermediate timing information is certain or uncertain. This identification is done when over-approximation is necessary (e.g., when merging abstract program states). This way, two WCETs estimations are computed: one that is a classical and pessimistic upper bound of the real WCET, and one that results from program/hardware states that are known to be reachable.

6 Conclusion and future work

In this paper, we summarized the semantic-aware WCET estimation workflow proposed in the W-SEPT project. From semantic properties extracted at high level, C level or binary level by static analysis or user annotation, we express them with the FFX annotation language, and trace them down to the binary level. The WCET estimation tool OTAWA has been adapted to integrate those properties through CFG modifications or ILP-based constraints. Generally, the results are good and show that the semantic-aware WCET workflow is a good opportunity to gain precision in WCET estimation. Moreover, we propose two methods for assessing the estimated WCET, by computing an *over-estimation* ratio that measures its (possible) relative pessimism; one method is based on constrained random simulation, while the other proceeds directly during the analysis.

This W-SEPT project highlighted that in the context of reactive systems, the semantic-aware WCET analysis may largely gain precision. In a future project, we aim at focusing on the specific context of reactive systems and synchronous languages, and consider the relations between timing analysis and certified code generation (e.g., DO 178C in avionics)

References

1 Armelle Bonenfant, Fabienne Carrier, Hugues Cassé, Philippe Cuenot, Denis Claraz, Nicolas Halbwachs, Hanbing Li, Claire Maiza, Marianne De Michiel, Vincent Mussot, Catherine Parent-Vigouroux, Isabelle Puaut, Pascal Raymond, Erven Rohou, and Pascal Sotin. When the worst-case execution time estimation gains from the application semantics. In *8th European Congress on Embedded Real-Time Software and Systems (ERTS2 2016)*, Toulouse, France, 2016.

2 Remy Boutonnet and Mihail Asavoae. The WCET analysis using counters – a preliminary assessment. In *8th JRWRTC, in conjunction with RTNS*, Versailles, France, 2014.

3 Florian Brandner, Stefan Hepp, and Alexander Jordan. Static profiling of the worst-case in real-time programs. In *20th International Conference on Real-Time and Network Systems (RTNS 12)*, Pont à Mousson, France, 2012.

4 Hugues Cassé, Haluk Ozaktas, and Christine Rochange. A framework to quantify the overestimations of static wcet analysis. In *15th International Workshop on Worst-Case Execution Time Analysis (WCET 2015)*, volume 47 of *OASIcs*, pages 1–10. Schloss Dagstuhl – Leibniz-Zentrum fuer Informatik, 2015. doi:10.4230/OASIcs.WCET.2015.1.

5 Patrick Cousot and Nicolas Halbwachs. Automatic discovery of linear restraints among variables of a program. In *5th ACM Symposium on Principles of Programming Languages (POPL'78)*, Tucson, Arizona, January 1978.

6 Jan Gustafsson, Andreas Ermedahl, Christer Sandberg, and Björn Lisper. Automatic derivation of loop bounds and infeasible paths for WCET analysis using abstract execution. In *27th IEEE Real-Time Systems Symposium (RTSS 2006)*, Rio de Janeiro, Brazil, 2006.

7 Nicolas Halbwachs, Paul Caspi, Pascal Raymond, and Daniel Pilaud. The synchronous data-flow programming language Lustre. *Proceedings of the IEEE*, 79(9):1305–1320, September 1991.

8 Julien Henry, David Monniaux, and Matthieu Moy. Pagai: A path sensitive static analyser. *Electronic Notes in Theoretical Computer Science*, 289:15–25, 2012.

9 Erwan Jahier, Simplice Djoko-Djoko, Chaouki Maiza, and Eric Lafont. Environment-model based testing of control systems: Case studies. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2014)*, Grenoble, France, 2014.

10 Erwan Jahier, Nicolas Halbwachs, and Pascal Raymond. Engineering functional requirements of reactive systems using synchronous languages. In *International Symposium on Industrial Embedded Systems (SIES 2013)*, 2013.

11 Raimund Kirner, Peter Puschner, and Adrian Prantl. Transforming flow information during code optimization for timing analysis. *Journal on Real-Time Systems*, 45(1-2), 2010.

12 Hanbing Li, Isabelle Puaut, and Erven Rohou. Traceability of flow information: Reconciling compiler optimizations and WCET estimation. In *22nd International Conference on Real-Time Networks and Systems (RTNS'14)*, Versailles, France, 2014.

13 Hanbing Li, Isabelle Puaut, and Erven Rohou. Tracing flow information for tighter wcet estimation: Application to vectorization. In *21st IEEE International Conference on Embedded Systems and Real-Time Computing Systems and Applications, RTCSA'15*, Hong Kong, China, 2015.

14 Yau-Tsun Steven Li and Sharad Malik. Performance analysis of embedded software using implicit path enumeration. *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, 16(12), 1997.

15 Vincent Mussot, Armelle Bonenfant, Pascal Sotin, Philippe Cuenot, and Denis Claraz. From relevant high-level properties to WCET computation improvement. In *International Conference on Embedded Real Time Software and Systems (ERTS2 2013)*, Toulouse, France, 2013.

16 Vincent Mussot, Jordy Ruiz, Pascal Sotin, Marianne de Michiel, and Hugues Cassé. Expressing and exploiting path conflicts in wcet analysis. In *16th International Workshop on Worst-Case Execution Time Analysis (WCET 2016)*, volume 55 of *OASIcs*, pages 3:1–3:11. Schloss Dagstuhl – Leibniz-Zentrum fuer Informatik, 2016. doi:10.4230/OASIcs.WCET.2016.3.

17 Vincent Mussot and Pascal Sotin. Improving WCET analysis precision through automata product. In *21st IEEE International Conference on Embedded Systems and Real-Time Computing Systems and Applications, RTCSA'15*, Hong Kong, China, 2015.

18 Pascal Raymond. Synchronous program verification with lustre/lesar. In S. Mertz and N. Navet, editors, *Modeling and Verification of Real-Time Systems*, chapter 6. ISTE/Wiley, 2008.

19 Pascal Raymond. A general approach for expressing infeasibility in implicit path enumeration technique. In *International Conference on Embedded Software (EMSOFT 2014)*, New Dehli, India, 2014.

20 Pascal Raymond, Claire Maiza, Catherine Parent-Vigouroux, Fabienne Carrier, and Mihail Asavoae. Timing analysis enhancement for synchronous program. *Real-Time Systems*, pages 1–29, 2015. doi:10.1007/s11241-015-9219-y.

21 Jordy Ruiz and Hugues Cassé. Using smt solving for the lookup of infeasible paths in binary programs. In *15th International Workshop on Worst-Case Execution Time Analysis (WCET 2015)*, volume 47 of *OASIcs*, pages 95–104. Schloss Dagstuhl – Leibniz-Zentrum fuer Informatik, 2015. doi:10.4230/OASIcs.WCET.2015.95.

22 Vivy Suhendra, Tulika Mitra, Abhik Roychoudhury, and Ting Chen. Efficient detection and exploitation of infeasible paths for software timing analysis. In *43rd annual Design Automation Conference (DAC'06)*, San Francisco, California, 2006.

23 Wei-Tsun SUN. A framework for simulate synchronous reactive programs and measure execution times to aid wcet analysis. Technical Report 27-06-2016, Verimag Research Report, 2016.

24 Reinhard Wilhelm, Jakob Engblom, Andreas Ermedahl, Niklas Holsti, Stephan Thesing, David Whalley, Guillem Bernat, Christian Ferdinand, Reinhold Heckmann, Tulika Mitra, Frank Mueller, Isabelle Puaut, Peter Puschner, Jan Staschulat, and Per Stenström. The worst-case execution-time problem – overview of methods and survey of tools. *ACM Transactions on Embedded Computing Systems (TECS)*, 7(3), 2008.

25 Youfeng Wu and James R. Larus. Static branch frequency and program profile analysis. In *27th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO94)*, San Jose, California, 1994.

26 Jakob Zwirchmayr, Armelle Bonenfant, Marianne de Michiel, Hugues Cassé, Laura Kovács, and Jens Knoop. FFX: A portable WCET annotation language (regular paper). In *20th International Conference on Real-Time and Network Systems (RTNS 12)*, Pont à Mousson, France, 2012.

27 Jakob Zwirchmayr, Pascal Sotin, Armelle Bonenfant, Denis Claraz, and Philippe Cuenot. Identifying relevant parameters to improve WCET analysis. In *14th International Workshop on Worst-Case Execution Time Analysis (WCET 2014)*, volume 39 of *OASIcs*, pages 93–102. Schloss Dagstuhl – Leibniz-Zentrum fuer Informatik, 2014. doi:10.4230/OASIcs.WCET.2014.93.