From Relevant High-level Properties to WCET Computation Improvement

Vincent Mussot[*], Armelle Bonenfant[*], Pascal Sotin[*], Denis Claraz[†], Philippe Cuenot[†]
[*]Université Toulouse, IRIT, Toulouse
[†]Continental Automotive, Toulouse

Abstract

The scheduling of real-time systems requires knowing the Worst Case Execution Time (WCET) of their tasks. WCET analysers compute timings by analysis of the low level behaviour of the target task. This document presents improvements in the WCET computation allowed by taking into account high level behaviours of the tasks. We first classify high level knowledge according to the relevance with respect to WCET estimation. We then propose a systematic method to bring this information back to the low level on which operate most WCET analysers. This approach separates the concerns of stating properties, integrating properties and computing a WCET aware of these properties.

Keywords: WCET computation, timing analysis, infeasible paths, annotation language

1 Imprecision in Timing Analysis of a Task

Critical hard real-time systems are composed of tasks which must imperatively finish before their deadline. In order to guarantee the safety of the system, a task scheduling analysis is performed and requires an a priori Worst-Case Execution Time (WCET) analysis of each task. Moreover for early evaluation of software application resource, it is a great benefit to be able to compute early Software Execution Time.

WCET analysers need to consider architectural details in order to be able to provide timing information. They reason on a Control Flow Graph (CFG) retrieved from the binary. Reasoning at the machine code level makes that information obvious at the design level or at the source code level have been lost. By ignoring such information, the analysers introduce pessimism in the computed WCET. For example, Figure 1 presents a CFG of a loop. Block 2 is the loop head and the worst case execution of the body is assumed to be *all* blocks from 4 to 8. In practice, this path never occurs because the CFG originates from a loop in the source code shown on Figure 2.

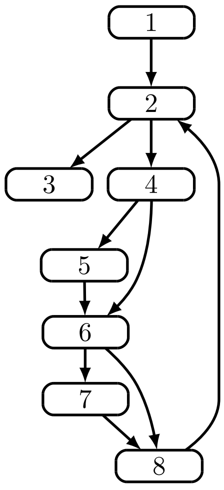


Figure 1: Loop CFG

The two main reasons for imprecision in timing analysis of a task are:

1. Pessimism at the architecture level (eg. cache accesses),
2. Pessimism in the considered control flow.

Numerous improvements have been suggested for 1. This article tries to tackle 2 with the following this philosophy:

- WCET analyses are by essence low-level analyses,
- Many flow information are known at design level or are easily retrieved at code source level,
- Imprecision in timing analysis can be reduced by transferring the relevant high level knowledge to the low level.

Such transfer already exists between the tools OTAWA and ORANGE developed in the IRIT Research Laboratory in Toulouse.

Otawa. Otawa is a toolbox for building WCET analyses [BCRS10]. It handles low level features like caches and pipelines for several architectures (including ARM, PowerPC and Tricore). Following the Implicit Path Enumeration Technique (IPET) the tool eventually converts the program CFG in an Integer Linear Program (ILP) which solution is an over-approximation of the WCET. Loops bounds are required from the user or from an external analyser.

```
for(i = 0; i < 100; i++) {
   if(i<s) { ... }
   if(i>s) { ... }
}
```

Figure 2: Loop C source

oRange. oRange is a static analyser for C mainly dedicated to infer context-sensitive bounds for the program loops. This tool generates precise source code level information. Its output format is called FFX([CdMB12]), an XML-based language for conveying flow information (FFX stands for Flow Fact in XML).

Contributions of the article. The main contributions of this article are:

- A classification of high level information capable of enhancing the WCET analysis,
- A generic method to transfer high level information encoded in FFX format to the Control Flow Graph representation of a WCET analyser,
- A proposal to early compute execution time of software application based on software context bounded by data flow definition.

Outline. Section 2 focuses on the user side of the WCET analyses. We list several categories of knowledge held by the system designer/developer. We also discuss how the knowledge harvesting can take place in an development process. Section 3 surveys relevant elements of the FFX format (Flow Facts in XML). This format has been designed to encode facts that hold for a given program or facts that should/can be assumed to hold. A full presentation can be found in [CdMB12]. Section 4 discuss the integration of high-level knowledge encoded in FFX into a WCET analysis by mean of CFG refinement through product automata. Section 5 presents preliminary results of the process on industrial code. We conclude in Section 6.

2 Useful Knowledge for Tightening WCET

Handling semantic aspects in timing analysis has been less studied compared to hardware aspects. The semantic analyses used by the WCET computation process are limited to determining loop bounds and rather simple infeasible paths. Once this information is gathered a bound can be found for the WCET even if it may be imprecise or unrealistic.

In order to increase the number of useful properties fed into the WCET computation, we have based our work on the following methodology: identification of relevant properties, extraction, expression, and eventually usage. In 2.1 we present a classification of properties and provide examples. In 2.2 we discuss how useful properties can be retrieved from the designer / developer in an industrial context.

2.1 Identification of the Useful Properties

Generic software properties. They are due to (known) design tools/methods (e.g. model-based design, synchronous languages) that may guarantee, by construction, important properties (e.g. bounded memory, bounded loops, memory privacy); these properties must be identified once for a given design method, and can then automatically be taken into account for a particular application.

Hypotheses on inputs/environment/context. Application-specific information may be provided by the software control engineer. It includes basic static information, such as numerical ranges (e.g. possible values from a sensor or on intermediate system data), commands exclusions (e.g. infeasible paths due to the environment, such as for instance different working modes); it may also include more dynamic information, such as known variations of loop indices, causality relations among time (e.g. initialization step, sensor behaviour determined by its environment and achieving a new state in the control). In the context of modular analysis, information about the execution context may also be provided by the software engineer (e.g. being in a certain mode is the consequence of a given execution sequence). When the execution context is unknown, the software engineer may define some execution modes (e.g. automatic or manual gearbox for a vehicle mode) leading to a partition of the parameters values.

We rely on experts to provide this kind of properties. The information they provide can be either asserted system knowledge and thus help to make the estimation more precise, or an execution mode or scenario for which the WCET needs to be computed. In the latter case the computed WCET should not be blindly trusted but can be a useful WCET estimator on the early stages of the system development.

Semantic properties. They derive from the program semantics: exclusion between test branches (e.g. the property of variable value inducing a certain path, excluding other paths), partition of loop executions (e.g. properties on frequencies of branches inside a loop), loop bounds, etc. Automatically extracting such properties requires to define a semantic model, and to use software static analysis techniques (inspired by/coming from model-checking or abstract interpretation).

In the community of timing analysis, most tools ([MBCS08, Abs07, Tid05, PSK08]) cover extraction and expression of semantic properties (most loop bounds and some infeasible paths). The automatic extraction of richer semantic properties is one of the goals of the project W-SEPT[1] but will not be addressed in this article.

2.2 Retrieving Properties from the Expert

Generic software properties have to be collected once. *Semantic properties* have to be kept up to date when the code evolves but this process can be automated. However, *hypotheses on inputs/environment/context* are numerous, evolving and cannot be systematically collected since they involve the system designers. In this section, we present some guidelines on where to spend the expert time according to our personal experiences.

The collaboration between Continental Automotive and IRIT aims to define the context for improving the WCET accuracy in order to be able to specify requirement for future industrial evaluation of system performance. The overall evaluation of the current work is as follows: Continental Automotive provides a scenario for a piece of code (and the corresponding code). This portion is known by the function developer to be time consuming (typically by measurement). However, due to the huge number of parameters and code branches linked to the system configuration it is laborious to locate the reason of this high consumption and to know whether some improvement can be done. IRIT takes the scenario and outputs the WCET and the corresponding path. The expert studies the path and excludes some part of it, for example incompatible values or exclusion of conditions. Then, the expert provides an improved scenario with these exclusions. Eventually, the developer may consider how to modify the code in order to improve the system performance. This is one possible and promising application of WCET estimation.

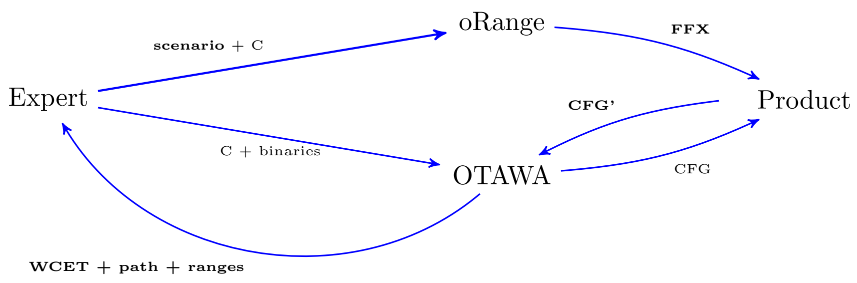


Figure 3: Overall process

The evaluation above is summarized in Figure 3. In general, the nature of the feedbacks will be:

[1]Funded by ANR, http://wsept.inria.fr

- Ranges of values leading to the WCET. The expert may find an incompatibility in the values leading to WCET. In that case, he can express formally the reason for that incompatibility (e.g. "if mode = critical then sensor greater than 1,000").
- Basic blocks involved in the worst-case path. Due to over-approximation, a worst-case path may not be feasible[2] (code structure or system behavior). If the expert realises that, he might tune the analyser by stating some exclusivity among blocks or feeding the data constraints leading to this exclusivity.
- Estimation of the potential improvements[3]

3 A Language for Flow Facts: FFX

FFX is a description format and an annotation language used to represent flow facts of a program.

The main concepts and ideas that drive the design and the development of FFX include – *freedom*: tools using FFX do not need to support all features. In the worst-case, unsupported features induce a loss of precision but never invalidate information; *expandability*: the use of XML allows to easily insert new elements or new attributes without breaking the compatibility with other tools; *soundness*: all provided information must be ruled by a precision order, ensuring that, at least, information in most generic cases must not be more precise than in particular cases; *flexibily*: possibility to define context dependent analysis i.e. reuse of component or context.

Listing 1: Extract of FFX grammar

```
<function LOCATION−ATTRS INFORMATION−ATTRS>
    STATEMENT∗
  </function>

STATEMENT ::= BLOCK | LOOP | CALL | CONDITION

LOOP ::= ... |
  <loop LOCATION−ATTRS INFORMATION−ATTRS>
    <iteration number="INT">
      STATEMENT∗
    </iteration>
  </loop>

CONDITIONAL ::=
  <conditional CONDITIONAL−ATTRS LOCATION−ATTRS>
    CONDITION?
    CASE+
  </conditional>

CONDITION ::=
  <condition CONDITION−ATTRS LOCATION−ATTRS>
    CONDITIONAL | STATEMENT
    STATEMENT∗
  </condition>

CASE ::=
  <case CASE−ATTRS LOCATION−ATTRS>
      CONDITIONAL | STATEMENT
  </case>

CONTROL−CONSTRAINT ::=
  <control−constraint> CONTROL−PREDICATE
  </control−constraint>
```

[2]In [KKZ13] Zwirchmayer et al. try to automate this process: the worst-case path is checked for feasibility.

[3]So far, a tool like OTAWA only delivers the WCET without giving information on possible improvements. Such information would be a great help for the industrial in order to know where to focus.

This format presents information in XML, structured as a tree, where tags represent structural elements of the program. Attributes can be attached corresponding to properties of these elements. FFX files are generated by a static analysis tool (in our case oRange [MBCS08]) on the program (in C) or provided/enriched by an expert. We provide in Listing 1 some details of tags and attributes used in the paper.

A STATEMENT can be either:

- BLOCK: A block element identifies a piece of code, possibly composed of several execution paths, but with a single entry point,
- CALL: A call element identifies the call to a function. Its location represents the caller and it must contain a function element representing the callee. Multiple call elements that embed functions allow to represent call-chain locations,
- CONDITION: A condition element represents a condition with several alternatives. In the C language, it applies to both `if` statements and `switch` statements,
- LOOP: A loop element matches a loop construct in the code. It may contain *iteration* elements in order to represent properties that are valid only during certain iterations of the loop.

The attributes of the LOOP tag are LOCATION-ATTRS and INFORMATION-ATTRS. LOCATION-ATTRS provides localisation information such as:

- IDENTIFICATION (id="TEXT") allows to make a reference to a part of code represented by another location attribute,
- ADDRESS-LOCATION (address="INT") gives the address but may be invalidated each time the application is compiled,
- SOURCE-LOCATION refers to line number and if necessary file name.

INFORMATION-ATTRS, for loops provides details on the loop bounds:

- **maxcount** the maximum number of time the loop iterates after each entry
- **totalcount** the maximum number of time the loop iterates for the whole program run
- **exact** true if the totalcount is exact or false if it is overestimated
- **executed** false if the loop is never executed, true if the loop executes in some cases

CONDITIONAL, CONDITION and CASE are used to describe a condition with several alternatives. They have attributes similar to LOOP.

The Listing 2 is an example of the simplified output of oRange on the initial example. In addition to the classical maximum of iteration of the loop 1 (maxcount="100"), the mutual exclusion of the two blocks is expressed in the tags <control−constraint>. Here, to simplify, block-IF1T (resp. block-IF2T), is the name of the block of the true condition 1 (resp. 2). The property in the example is that block-IF1T + block-IF2T ≤ 1 for each iteration of the loop.

Properties described in the file are to be exploited by any code analyser, in our case, by OTAWA.

In the current state, OTAWA is fetching the information on loop bounds (and seldom some variable domains), whereas FFX is able to describe a larger amount of properties. We propose a treatment of FFX files in order to fill the gap between expression and usage of properties in our tool chain.

4 From Knowledge Gain to Precision Gain

Once the high level information relevant for the WCET analysis has been identified, collected and expressed in FFX, it still needs to be integrated in the analysis to yield a precision gain.

The transfer technique sketched in this section relies on viewing the Control Flow Graph handled by the WCET analysers as an automaton accepting sequences of transition between blocks. For example, if we consider the CFG pictured on Figure 1, we can see that after accepting $2 \rightarrow 4$ the CFG can accept

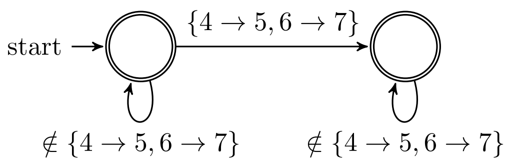
Listing 2: Simple example

```
<flowfacts>
    <function name="main" executed="true" extern="false">
          <loop id="1" line="4" source="src_c/simple_loop.c" exact="true"
          maxcount="100">
          <iteration number="*">
              <control−constraint>
                  <le>
                      <add><count id="block−IF1T"/><count id="block−IF2T"/></add>
                      <const int="1"/>
                  </le>
          </control−constraint>
          </iteration>
          </loop
     </function>
</flowfacts>
```

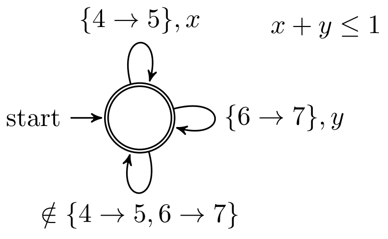
$4 \to 5$ or $4 \to 6$. The information contained in the FFX file can be seen as a restriction of what can really be accepted by the CFG. For example, the FFX shown on Listing 2 states that in every iteration of the loop, $4 \to 5$ and $6 \to 7$ cannot be accepted simultaneously.

The application of the restriction will be provided by computing a new CFG, more restrictive, by making an automata product of the CFG with an automaton encoding the FFX. Such operation can be unambiguously described and implemented according to a formal description.

Encoding exclusive paths using automata. The automaton Figure 4a encodes the sequences of edges containing no more than one edge $4 \to 5$ or $6 \to 7$. If we construct the product of this automata with the loop body CFG we obtain a new CFG where the spurious trace $2 \to 4 \to 5 \to 6 \to 7 \to 8$ no longer exists. This CFG can be seen on Figure 6a.



(a) Usual automaton



(b) Automaton with constraints

Figure 4: Two automata encoding: *either transition* $4 \to 5$ *or transition* $6 \to 7$

Encoding exclusive paths using automata and constraints. The issue of performing a product with automaton like the one of Figure 4a is that the size of the resulting CFG will grow with each information added by the user, therefore slowing the subsequent analysis.

Since we assume that the WCET analyser rely on IPET to compute its prediction we also consider automata enriched with counters and numerical constraints. The automaton of Figure 4b encodes the sequences of edges in which the count of edges $4 \to 5$ and $6 \to 7$ is less than one. This automaton directly derives from the <control−constraint> FFX element.

If we perform a product between the loop body CFG and this automaton, we obtain the same CFG, carrying the counters x and y (see Figure 6b). The spurious trace $2 \to 4 \xrightarrow{x} 5 \to 6 \xrightarrow{y} 7 \to 8$ contains one occurence of counter x and counter y and will thus be rejected by the constraint $x + y \leq 1$.

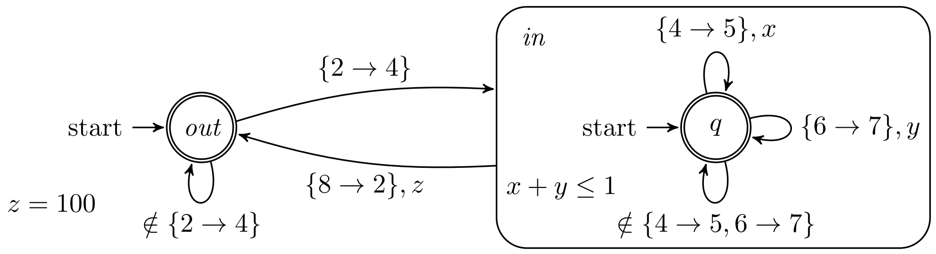
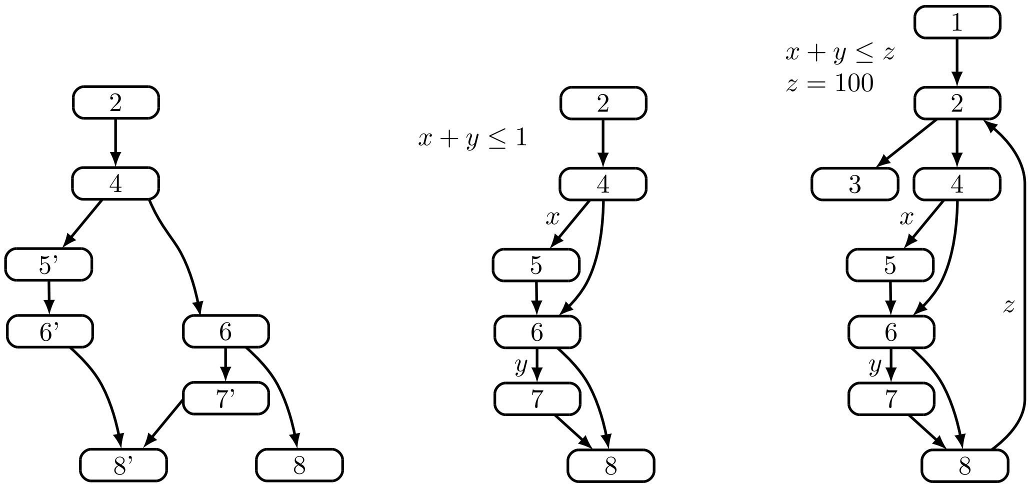


Figure 5: Hierarchical automaton



(a) Loop body after product with Automaton 4a

(b) Loop body after product with Automaton 4b

(c) Program CFG after product with Automaton 5

Figure 6: Constrained CFG

Encoding the contexts. The FFX language offers the notion of context which allows to state that a given property is true in a given situation. The automata shown on Figure 4 states an exclusivity property but it remains to take into account that this property holds at each iteration of the loop. To do so we rely on hierarchical automata which can recursively carry automata in their nodes. The subautomata are "activated" each time the state containing them is reached.

Figure 5 shows a hierarchical automaton which encodes the FFX information stated by Listing 2. Since OTAWA CFG do not have a hierarchical nature, the product of the hierarchical automaton and the CFG have to be flattened. The new CFG is depicted on Figure 6c. Due to the flattening, the spurious trace $2 \to 4 \xrightarrow{x} 5 \to 6 \xrightarrow{y} 7 \to 8$ is allowed but if that trace is considered taken in 50 iterations then the 50 other iterations must execute $2 \to 5 \to 8$.

Once the generic product and flattening are defined and implemented by a WCET analyser like OTAWA, all the burden of expressing the FFX construct in term of automaton is left to the FFX language designers. In [EE00], Engblom and Ermedahl present a similar integration of flow facts in a CFG.

5 Experimentation

To demonstrate the effectiveness of our path suppression technique, we use a sample of Continental Automotive code dedicated to the control of the engine speed regulation of an engine management. This code contains mostly conditions and no loops so we are able to obtain a WCET directly. However we used a scenario file where most variables were initialized, in order to correspond to a state of the engine, mostly depending on the engine rotation speed. According to this scenario, oRANGE is able to specify which of the *then* and the *else* branch is taken. It generates an associated FFX file containing several properties for each *case* (*then* or *else*) including a property *executed* which can be true or false. True

Program	WCET w/o Path suppression	WCET w/ Path suppression	Percentage Gain
an_is	2529	2377	6.01%

Table 1: Benchmark of industrial code sample

here means "may be executed" when false means "is not executed".

Listing 3: FFX Condition

```
 <conditional><condition line="125" source="an_is.c" executed="true"></condition>
    <case cond="1"  executed="false" line="126" source="an_is.c"></case>
    <case cond="0" executed="true" line="128" source="an_is.c"></case>
</conditional>
```

```
125 if (lv_dly_n_sp_is) {
126   tmp_u16 = dri_n_sp_is;
127 } else {
128   tmp_u16 = n_sp_is_1;
129 }
```

Figure 7: Code condition

In order to obtain a reduced WCET, we use the flow information present in the FFX file to perform a path suppression when it does not correspond to the scenario file. The scenario file is a conjonction of facts which can be :

- values for code variables
- ranges for code variables
- conditions between code variables

When oRange is able to set the executed property of a case to *false*, the corresponding block in the CFG should never be executed. That is why we remove the associated branch in the CFG and we only let the path which goes through the other case of the condition. This way, if the most costly case of the condition cannot be executed, the only path remaining will be taken for the calculation of the WCET and this new estimation will hopefully be lower than the previous one. However this new estimation shall only be used in association with its specific scenario.

Table 1 shows the reduction of the WCET using the specific scenario file. The reduction of the WCET obtained may be different for any other scenario given, depending on which of the two cases of each condition is specified as false.

6 Conclusion

Over-approximation in WCET computation introduces pessimism and needs to be reduced. Better knowledge of the program, i.e. application domain (in term of variables), execution mode, environment/architecture support, design process, even compilation are valuable sources of information and can be found and exploited all along the development process.

In this work, we presented a classification, aiming to simplify identification and extraction (manual or automatic) of the program properties useful for WCET computation. We express these properties in FFX, which offers an easy way for experts to state their knowledge on the program.

We described an improvement for OTAWA, which was restricted by its ability of reading FFX, by adding a translation step: FFX into CFG thanks to an automaton. Although, this restriction was specific to our tool, the proposal not only levies this restriction, but also helps to enlarge the usage of FFX by other timing analysis tools. The use of FFX as pivot language has been initiated in [ZBdM+12]. Thus, any FFX file, provided by analysis or expertise, with numerous kind of properties would be readable by OTAWA.

The next step is to establish and validate clear requirement for timing analysis and optimization to be able to optimize but ensure predictive behaviour and soundness of real time application.

Acknowledgements. The authors wish to acknowledge Jakob Zwirchmayer and the anonymous referees for their thorough reviews. This work is supported by the French ANR project W-SEPT.

References

[Abs07] AbsInt Angewandte Informatik GmbH. aiT. http://www.absint.com, 2007.

[BCRS10] Clément Ballabriga, Hugues Cassé, Christine Rochange, and Pascal Sainrat. OTAWA: an Open Toolbox for Adaptive WCET Analysis. In *Proc. of IFIP Workshop on Software Technologies for Future Embedded and Ubiquitous Systems (SEUS)*, Austria, October 2010. Springer.

[CdMB12] Hugues Cassé, Marianne de Michiel, and Armelle Bonenfant. FFX (Flow Fact in XML) format. Technical report RR–2012-5–EN, IRIT, Université Paul Sabatier, Toulouse, 2012.

[EE00] Jakob Engblom and Andreas Ermedahl. Modeling complex flows for worst-case execution time analysis. In *RTSS*, pages 163–174, 2000.

[KKZ13] Jens Knoop, Laura Kovács, and Jakob Zwirchmayr. Wcet squeezing: on-demand feasibility refinement for proven precise wcet-bounds. In *RTNS*, pages 161–170, 2013.

[MBCS08] Marianne De Michiel, Armelle Bonenfant, Hugues Cassé, and Pascal Sainrat. Static Loop Bound Analysis of C Programs Based on Flow Analysis and Abstract Interpretation. In *Proc. 14^{th} IEEE International Conference on Embedded and Real-Time Computing Systems and Applications, (RTCSA 2008)*, Taiwan, 2008.

[PSK08] Adrian Prantl, Markus Schordan, and Jens Knoop. TuBound – A Conceptually New Tool for Worst-Case Execution Time Analysis. In *Proc. 8^{th} International Workshop on Worst-Case Execution Time Analysis (WCET 2008)*, pages 141–148, Prague, Czech Republic, 2008. Österreichische Computer Gesellschaft. ISBN: 978-3-85403-237-3.

[Tid05] Tidorum Ltd. Bound-T. lhttp://www.tidorum.fi/bound-t, 2005.

[ZBdM$^+$12] Jakob Zwirchmayr, Armelle Bonenfant, Marianne de Michiel, Hugues Cassé, Laura Kovacs, and Jens Knoop. FFX: A Portable WCET Annotation Language (regular paper). In *International Conference on Real-Time and Network Systems (RTNS), Pont-à-Mousson, 08/11/2012-09/11/2012*, pages 91–100, http://portal.acm.org/dl.cfm, novembre 2012. ACM DL.