University of Hertfordshire UH 25 Research Archive

Citation for published version:
M. Zolda, and R. Kirner, "Calculating WCET estimates from timed traces", *Real Time Systems,* Vol. 52(1), September 2015.

DOI:
10.1007/s11241-015-9240-1

Document Version:
This is the Published Version.

Copyright and Reuse:
© The Author(s) 2015. This article is published with open access at Springerlink.com.
This is an open-access article distributed under the terms of the Creative Commons Attribution License (CC BY). The use, distribution or reproduction in other forums is permitted, provided the original author(s) or licensor are credited and that the original publication in this journal is cited, in accordance with accepted academic practice. No use, distribution or reproduction is permitted which does not comply with these terms.

Enquiries
If you believe this document infringes copyright, please contact the Research & Scholarly Communications Team at rsc@herts.ac.uk

Real-Time Syst (2016) 52:38–87
DOI 10.1007/s11241-015-9240-1

Calculating WCET estimates from timed traces

Michael Zolda[1] · **Raimund Kirner**[1]

Published online: 1 September 2015
© The Author(s) 2015. This article is published with open access at Springerlink.com

Abstract Real-time systems engineers face a daunting duty: they must ensure that each task in their system can always meet its deadline. To analyse schedulability they must know the worst-case execution time (WCET) of each task. However, determining exact WCETs is practically infeasible in cost-constrained industrial settings involving real-life code and COTS hardware. Static analysis tools that could yield sufficiently tight WCET bounds are often unavailable. As a result, interest in portable analysis approaches like measurement-based timing analysis is growing. We present an approach based on integer linear programming (ILP) for calculating a WCET estimate from a given database of timed execution traces. Unlike previous work, our method specifically aims at reducing overestimation, by means of an automatic classification of code executions into scenarios with differing worst-case behaviour. To ease the integration into existing analysis tool chains, our method is based on the implicit path enumeration technique. It can thus reuse flow facts from other analysis tools and produces ILP problems that can be solved by off-the-shelf solvers.

Keywords WCET · Worst-case execution time · IPET · Implicit path-enumeration technique · ILP · Integer linear programming · Timing analysis · Measurement · Context-sensitive · Scenarios

✉ Michael Zolda
michael@zolda.eu

Raimund Kirner
r.kirner@herts.ac.uk

[1] University of Hertfordshire, Hatfield AL10 9AB, UK

1 Introduction

Modern software usually adopts a modular design with multiple interacting computational processes executing concurrently inside the system. These processes compete for shared system resources like memory space, processing time, etc. In real-time processor scheduling the demands of each process for processing time are specified by a set of task parameters. Given all task parameters of the system, real-time systems engineers can easily check if a given scheduling policy (Liu and Layland 1973; Dertouzos and Aloysius 1989; Xu and Parnas 1990; Neil et al. 1991) can guarantee to arbitrate all resource demands in a timely manner during system operation.

We are concerned with the *worst-case execution time* (WCET)—a crucial task parameter prescribing the maximal amount of time the corresponding process requires a processor to complete.

We consider that processes are implemented in software. The problem we need to solve then is: How can the prescribed WCET of a task (a mathematical abstraction used in schedulability analysis) be linked to the actual behavior of executable code on particular hardware? Specifically, how can we determine the WCET of a piece of code on a given microprocessor?

Measurement-based timing analysis (MBTA) (Bernat et al. 2002, 2003; Kirner et al. 2005;Wenzel et al. 2009; Stattelmann and Martin 2010) proposes the calculation of WCET estimates from a database of timed execution traces of code runs on the target hardware. Furthermore, many analysis tool chains rely on the *Implicit Path Enumeration Technique* (IPET) (Li and Malik 1995; Puschner and Schedl 1997; Li and Malik 1997) to calculate a global WCET estimate from WCET estimates of individual code constituents.

WCET estimates obtained by MBTA are generally not guaranteed *upper WCET bounds*, and care must be taken when they are used to approximate execution times. For example, the schedulability tests of many classical hard real time scheduling algorithms admit the approximation of execution times by upper bounds, whereas the use of mere estimates could result in false positives. WCET estimates are more useful in various other scenarios, like: soft real time systems, mixed-criticality systems, load balancing, the fast evaluation of design options, and others.

We present an IPET-based approach for calculating a WCET estimate from a given database of timed execution traces (Sect. 3). Extending standard IPET, our method can reuse flow facts from other flow analysis tools. First, however, we discuss related work (Sect. 2). We conclude with an experimental evaluation of a proof-of-concept implementation (Sect. 4). In Appendix 1 we present proofs for all theorems in the main part of the paper.

2 Related work

WCET analysis is usually performed in two stages: A low-level stage where local estimates for individual code constituents—individual instructions, instruction sequences, basic blocks, etc.—are determined, and a high-level stage, where local estimates are combined into a global estimate for the complete code.

Early methods for high-level analysis were based on abstract syntax trees (Puschner and Koza 1989; Puschner and Schedl 1993; Puschner 1998; Shaw 1989; Park and Shaw 1991), where leaf nodes corresponded to elementary statements (e.g. assignments) and inner nodes corresponded to structured control constructs for sequences, selects, and iterations (Dijkstra 1970). Determining the local estimates for elementary statements from the execution times of individual machine instructions was considered easy, so low-level analysis was not considered further.

Later it became apparent that the jitter in the execution time of individual instructions that is caused by certain performance-enhancing hardware features, e.g., caches, instruction pipelines, and branch predictors, must be considered in order to obtain practically usable WCET estimates. Consequently, various solutions for the low-level analysis of individual path fragments in the presence of caches, pipelines, and branch predictors were proposed (Li and Malik 1999; Lundqvist and Stenström 1998, 1999; Stappert and Altenbernd 2000; Theiling and Ferdinand 1998; Schneider and Ferdinand 1999; Ferdinand et al. 1999; Colin and Puaut 2000). In contrast to these model-based approach, measurement-based timing analysis (MBTA) proposed the use of timed execution traces of individual code constituents obtained runs of the code on the target hardware (Bernat et al. 2002, 2003; Kirner et al. 2005; Wenzel et al. 2009; Stattelmann and Martin 2010).

On the high-level stage, the early tree-based approaches for high-level analysis were soon found too inflexible: Firstly, tree-based approaches work only for structured programs, but not for programs that contain unstructured jumps. Although unstructured code is sometimes dismissed as bad programming discipline, there exist cases that justify the use of unstructured jumps, e.g., state machines. Moreover, language features like exceptions contain implicit unstructured jumps. Secondly, the tree-based approaches did not support the exclusion of infeasible execution paths. Experimentation with various solutions eventually led to the wide adoption of the Implicit Path Enumeration Technique (IPET) (Li and Malik 1995; Puschner and Schedl 1997; Li and Malik 1997) for the high-level stage. We recap IPET in Sect. 3.1.

IPET uses a fixed WCET estimate for each code constituent. In Sect. 3.2 we argue that this is a major limitation that can lead to overly pessimistic estimates by combining mutually exclusive execution scenarios. Researchers have been aware of this and have proposed various remedies:

Li et al. analyse the control-flow between code mapping to shared cache lines. This analysis affords linear constraints on the number of cache misses, modeled by an extra variables. They have shown their approach for direct-mapped (Li and Malik 1995, 1999) and set-associative caches (Li and Malik 1996). In comparison our approach is not limited to modeling specific hardware, but relies on the observed system behavior. Still our approach is capable of handling cache-induced jitter, as demonstrated by our experimental evaluation (cf. Sect. 4).

Ottoson and Sjödin (1997) introduce separate variables for pairs of adjacent program constituent to model pipelines. Our concept of *clips* (cf. Sect. 3.3) allows a more general separation of contexts, subsuming Ottoson's and Sjödin's case.

Engblom and Ermedahl (1999) use instruction sequences. Since instruction sequences are one of many possible choices for code constituents, our approach can reuse this idea. Theiling and Ferdinand (1998) have proposed *virtual inlining/virtual*

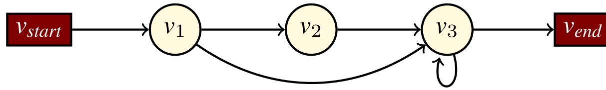


Fig. 1 Visualization of the CFG from Example 1

unrolling (VIVU), allowing the separation of certain forms of execution scenarios: Firstly, they can separate the first iteration of loops from subsequent ones. However, our approach provides a more general way of choosing contexts via clips. Secondly, they can distinguish procedure invocations that originate from different call sites. Although our approach currently works only on individual procedures, it could be extended to integrate the idea of call contexts.

3 Calculating WCET estimates

3.1 The implicit path-enumeration technique

The *Implicit Path Enumeration Technique* (IPET) (Li and Malik 1995, 1997; Puschner and Schedl 1997) is a widely-used method for calculating an upper WCET bound for a piece of code from upper WCET bounds of the code's constituents.

Standard IPET is based on integer linear programming: The execution count of each code constituent is expressed as a non-negative integer variable, and the possible control flow between code constituents is overapproximated by linear constraints. The cost of each code constituent is its local upper WCET *bound*, and the objective function is the cost-weighted sum of execution counts of all code constituents. Maximizing the objective function yields a WCET bound for the complete code. If *estimates* rather than bounds are used as costs, the methods yields a WCET estimate for the code.

Definition 1 (*Control-flow graph*) A *control-flow graph* G of a program $\mathcal{P}$ is a quadruple $(V, E, v_{start}, v_{end})$ consisting of a set of *nodes* V, a set of *edges* $E \subseteq V \times V$, a *start node* $v_{start} \in V$, and an *end node* $v_{end} \in V$, such that all other nodes are reachable from v_{start}, and v_{end} is reachable from all other nodes, i.e., if E^+ designates the transitive closure of E, $\{(v_{start}, v) \mid v \in V \setminus \{v_{start}\}\} \subseteq E^+$ and $\{(v, v_{end}) \mid v \in V \setminus \{v_{end}\}\} \subseteq E^+$. Moreover, v_{start} is unreachable from other nodes and no node is reachable from v_{end}, i.e., $\{(v, v_{start}), (v_{end}, v) \mid v \in V\} \cap E^+ = \emptyset$.

Example 1 Figure 1 illustrates the CFG $\mathcal{G} = (V, E, v_{start}, v_{end})$ with nodes

$$V = \{v_{start}, v_{end}, v_1, v_2, v_3\} \text{ and edges}$$
$$E = \{(v_{start}, v_1), (v_1, v_2), (v_1, v_3), (v_2, v_3), (v_3, v_3), (v_3, v_{end})\}.$$

Each node in a CFG $\mathcal{G} = (V, E, v_{start}, v_{end})$ of a program $\mathcal{P}$ is identified with some atomically executable code constituent. Different choices, like instructions, instruction sequences, arbitrary code blocks, etc., affords a lot a flexibility. The start node v_{start} and the end node v_{end} mark the entry and exit points of the code and thus correspond to empty code.

Following the modeling presented in Li and Malik (1997), we use an integer *node variable* f_v for each node $v \in V$, designating the execution count of v during a single run of $\mathcal{P}$ and an integer *edge variable* $f_{(v,w)}$ for each edge $(v, w) \in E$, designating the number of control transfers from node v to node w. The solution space is chosen to overapproximate all feasible runs: For any feasible run $\pi = v_1, \ldots v_n$ of $\mathcal{P}$, the valuation

$$\{f_v \mapsto c_v \mid v \in V\} \cup \{f_e \mapsto c_e \mid e \in E\}$$

must be a solution, where c_v is the number of occurrences of node v in π, and where c_e is the number of occurrences of edge e in π, i.e.,

$$c_v = |\{i \mid v_i = v, 1 \leq i \leq n\}| \quad \text{and} \quad c_e = |\{i \mid (v_i, v_{i+1}) = e, 1 \leq i < n\}|.$$

Because numbers of occurrences are cardinals, non-negativity constraints are added

$$f_v \geq 0, \text{ for all } v \in V \quad \text{and} \quad f_e \geq 0, \text{ for all } e \in E.$$

To model a single run of program $\mathcal{P}$, exactly one occurrence of the start node v_{start} and exactly one occurrence of the end node v_{end} is enforced:

$$f_{v_{start}} = 1 \quad \text{and} \quad f_{v_{end}} = 1.$$

Just as in Li and Malik (1995, 1997) and Puschner and Schedl (1997) , we consider sequential programs, i.e., the execution of $v \neq v_{end}$ must be followed by the execution of exactly one of its immediate successor nodes w. Likewise, the execution of any node $v \neq v_{start}$ must always follow the execution of exactly one of its immediate predecessor nodes w. In IPET, this can be expressed by linear constraints called *structural* constraints

$$f_v = \sum_{(w,v) \in E} f_{(w,v)} \quad \text{and} \quad f_v = \sum_{(v,w) \in E} f_{(v,w)}, \quad \text{for all } v \in V.$$

To obtain a bounded problem, addition information about cyclic regions is needed. The notion of a *natural loop* is sufficient for cycles in structured code: A node v is said to *dominate* another node w, if every path from the start node v_{start} to w must pass through v. A natural loop is defined by its *back edge* (w, v), which is an edge, such that its target node v *dominates* its source node w. Constraints for natural loops are typically inequalities that bound the flow through the back edge relative to the total flow $f_{(w_1,v)} + \cdots + f_{(w_n,v)}$ into the *loop header* v, i.e.,

$$f_{(w,v)} \leq b \cdot (f_{(w_1,v)} + \cdots + f_{(w_n,v)}),$$

where b is a positive integer constant, where $\{w_1, \ldots, w_n, w\}$ is the set of immediate predecessors of node v, and where w dominates v.

The objective function of a standard IPET problem is

$$\sum_{v \in V} \widetilde{wcet}_v \cdot f_v,$$

where $\widetilde{wcet}_v$ is the local WCET estimate of node v.

Additional linear constraints can be obtained by program analysis or expert insight into the code (Puschner and Schedl 1997).

Example 2 Reconsider the CFG from Example 1 and assume that the loop can reiterate at most 7 times after entry. Also assume WCET estimates of 50, 20, and 30 microseconds for nodes v_1, v_2, and v_3. We obtain the following IPET problem:

– Non-negativity constraints:

$$f_{v_{start}} \geq 0; \qquad f_{v_1} \geq 0; \qquad f_{v_2} \geq 0; \qquad f_{v_3} \geq 0; \qquad f_{v_{end}} \geq 0.$$

– Structural and single run constraints:

$$\begin{aligned} f_{v_{start}} &= f_{(v_{start},v_1)}; & f_{v_1} &= f_{(v_{start},v_1)}; & f_{v_2} &= f_{(v_1,v_2)}; \\ f_{v_{start}} &= 1; & f_{v_1} &= f_{(v_1,v_2)} + f_{(v_1,v_3)}; & f_{v_2} &= f_{(v_2,v_3)}; \\ f_{v_3} &= f_{(v_3,v_3)} + f_{v_{end}}; & f_{v_{end}} &= (v_3, v_{end}); \\ f_{v_3} &= f_{(v_1,v_3)} + f_{(v_2,v_3)} + f_{(v_3,v_3)}; & f_{v_{end}} &= 1; \end{aligned}$$

– Iteration constraint:

$$f_{(v_3,v_3)} \leq 7 \cdot (f_{(v_1,v_3)} + f_{(v_2,v_3)}).$$

– Objective function:

$$0 \cdot f_{v_{start}} + 50 \cdot f_{v_1} + 20 \cdot f_{v_2} + 30 \cdot f_{v_3} + 0 \cdot f_{v_{end}}.$$

– We may add additional linear constraints, if available. For example, if we understood that the loop is constrained to four iterations for each entry through edge (v_2, v_3), we would add an extra constraint

$$f_{(v_3,v_3)} \leq 7 \cdot f_{(v_1,v_3)} + 3 \cdot f_{(v_2,v_3)}.$$

3.2 Pessimism and monotonicity

IPET overapproximates the global WCET, assuming that local WCET behaviors in two or more different program constituents may coincide, even where such a scenario cannot be exhibited by the system, as illustrated by the example in Fig. 2. This conservative approximation is sometimes called as *pessimism*. Given a suitable metric over

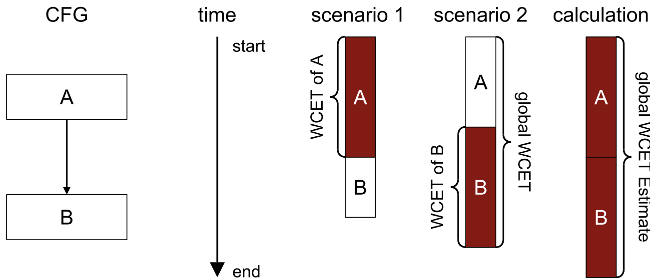


Fig. 2 Example for pessimistic estimate calculation: consider a program fragment consisting of two consecutive constituents A and B. Assume that there are only two execution scenarios. Constituent A exhibits its WCET in the first scenario; constituent B in second scenario. Standard IPET yields the sum of the local WCET estimates as global WCET estimate. This is pessimistic, because a run in which both local WCETs occur does not exist. The global WCET estimate is higher than the actual global WCET, which occurs in Scenario 2

WCET estimates, we can quantify the pessimism of a given method as the distance of its estimate(s) from the actual WCET(s).

Note that this extremely simple example serves just to illustrate the concept of pessimism. The method proposed in this paper requires the existence of different paths between nodes, as in Example 1.

The IPET method is *monotonic* in the local WCET estimates, i.e., the global WCET estimate cannot decrease, unless some local WCET estimate decreases by a positive value δ.

Definition 2 (*Monotonic estimate calculation method*) Let f be a function that takes as arguments local WCET estimates $\widetilde{wcet}_1, \ldots, \widetilde{wcet}_n$ and yields a global WCET estimate $\widetilde{wcet} = f(\widetilde{wcet}_1, \ldots, \widetilde{wcet}_n)$. We say that f *denotes a monotonic estimate calculation method*, iff, for any $\delta_1, \ldots, \delta_n \geq 0$,

$$f(\widetilde{wcet}_1 + \delta_1, \ldots, \widetilde{wcet}_n + \delta_n) \geq f(\widetilde{wcet}_1, \ldots, \widetilde{wcet}_n).$$

3.3 Context-sensitive IPET

Context-sensitive IPET introduces *execution scenarios* to standard IPET, to reduce overestimation. The method reuses all variables, constraints, and extra flow facts from the respective standard IPET problem. For each node $v \in V$ we model a set of execution scenarios

$$E_{v,1}, \ldots, E_{v,n(v)}$$

with separate WCET estimates

$$\widetilde{wcet}_{v,1}, \ldots, \widetilde{wcet}_{v,n(v)}.$$

The execution counts of these scenarios are modeled by *scenario variables*

$$f_{v,1}, \ldots, f_{v,n(v)}.$$

The scenarios of node $v \in V$ are refinements of the unspecific scenario of v in standard IPET. Hence, the estimates $\widetilde{wcet}_{v,1}, \ldots, \widetilde{wcet}_{v,n(v)}$ do not exceed $\widetilde{wcet}_v$.

Requirement 1 (Execution scenario specialization) *The WCET estimate under a specific execution scenario of a node v does not exceed the unspecific WCET estimate $\widetilde{wcet}_v$ of v, i.e.,*

$$\widetilde{wcet}_{v,i} \leq \widetilde{wcet}_v, \quad \textit{for all } 1 \leq i \leq n(v).$$

The execution scenarios $E_{v,1}, \ldots, E_{v,n(v)}$ must classify the executions of v, i.e., each execution of node v during a run of program $\mathcal{P}$ is associated with exactly one execution scenario of v.

Requirement 2 (Execution scenario classification) *For each node $v \in V$, the associated execution scenarios $E_{v,1}, \ldots E_{v,n(v)}$ form a classification of the occurrences of v in any end-to-end path through the CFG.*

Requirement 2 yields new constraints

$$f_v = \sum_{i=1}^{n(v)} f_{v,i}, \quad \text{for all } v \in V.$$

Because number of occurrences are cardinals, we add constraints

$$f_{v,i} \geq 0, \quad \text{for all } v \in V, 1 \leq i \leq n(v).$$

Our objective function is a refinement of the objective function of the corresponding standard IPET problem, obtained by replacing each summand

$$\widetilde{wcet}_v \cdot f_v \qquad \text{by the weighted sum} \qquad \sum_{i=1}^{n(v)} \widetilde{wcet}_{v,i} \cdot f_{v,i},$$

i.e., the new objective function has the form

$$\sum_{v \in V} \sum_{i=1}^{n(v)} \widetilde{wcet}_{v,i} \cdot f_{v,i}.$$

Theorem 1 *The global WCET estimate obtained from a context-sensitive IPET problem never exceeds the global WCET estimate obtained from the respective standard IPET problem.*

A proof of this and all further theorems can be found in Appendix 1 of this paper.

Having several different execution scenarios is only useful, if we can establish additional constraints that involve these scenarios. We therefore add the following requirement:

Requirement 3 (Constraints) *The classification of trace occurrences of a node* $v \in V$ *into scenarios* $E_{v,1}, \ldots E_{v,n(v)}$ *admits additional constraints over the corresponding scenario variables* $f_{v,1}, \ldots, f_{v,n(v)}$.

Example 3 Reconsider the IPET problem from Example 2. Assume that the WCET of node v_3 varies depending on which of the three node v_1, v_2, and v_3 were executed just before node v_3, e.g., due to the caching of accesses to instruction and data memory. Assume that the WCET estimates of node v_3 right after executing nodes v_1, v_2, and v_3 are 30, 10, and 5 microseconds. We introduce different execution scenarios $E_{v_3,1}$, $E_{v_3,2}$, and $E_{v_3,3}$ for node v_3 and add the following constraints to the original IPET problem:

$$f_{v_3} = f_{v_3,1} + f_{v_3,2} + f_{v_3,3}.$$

The objective function of our context-sensitive IPET problem is:

$$0 \cdot f_{v_{start}} + 50 \cdot f_{v_1} + 20 \cdot f_{v_2} + 30 \cdot f_{v_3,1} + 10 \cdot f_{v_3,2} + 5 \cdot f_{v_3,3} + 0 \cdot f_{v_{end}}.$$

So far, the context-sensitive IPET problem does not improve on the standard IPET problem. However, the execution scenario of node v_3 depends on which out of nodes v_1, v_2, or v_3 was executed immediately before, so we add the following constraints:

$$f_{v_3,1} = f_{(v_1,v_3)}; \qquad f_{v_3,2} = f_{(v_2,v_3)}; \qquad f_{v_3,3} = f_{(v_3,v_3)}.$$

The total number of occurrences of edges (v_1, v_3) and (v_2, v_3) in a single run must be 1, due to the structural constraints of the IPET problem. These constraints tighten the WCET estimate by enforcing a closer upper bound on $f_{v_3,3}$.

Concrete methods with different properties can be created by instantiating context-sensitive IPET, which is a generic method. In the following sections we develop one such instantiation:

- In Sect. 3.4, we introduce the notion of a *context* of a node, to instantiate the notion of an execution scenario, and two associated refinement operators.
- In Sect. 3.5, we show how to infer context-specific linear constraints over the number of times the given node may appear on any end-to-end CFG paths, such that Requirement 3 is fulfilled.
- In Sect. 3.6, we show how to infer WCET estimates for individual contexts from a database of timed execution traces, such that Requirement 1 is fulfilled.
- In Sect. 3.7, we present an algorithm for obtaining contexts for a given node, such that Requirements 2 and 3 are fulfilled.
- In Sect.3.8, we put everything together, instantiating context-sensitive IPET.

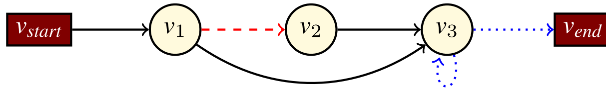


Fig. 3 Visualization of the clip $\mathcal{S}_0$ from Examples 4. The single entry edge of $\mathcal{S}_0$ are *dashed*, and its two exit edges are *dotted*. The are two paths in $\mathcal{S}_0$

3.4 Contexts

We consider the CFG $\mathcal{G}_\mathcal{P} = (V, E, v_{start}, v_{end})$ of some fixed program $\mathcal{P}$. We write $\mathcal{U}$ to denote the set of *end-to-end paths* of $\mathcal{G}_\mathcal{P}$, i.e.,

$$\mathcal{U} = \{v_1 \dots v_n \mid v_1 = v_{start}, v_n = v_{end}, (v_i, v_{i+1}) \in E, 1 \leq i < n, n \in \mathbb{N}\}.$$

In the following, R^* denote the reflexive, transitive closure of any binary relation R. Hence, E^* denotes the reachability relation in $\mathcal{G}_\mathcal{P}$, and $(E \setminus K)^*$ with $K \subseteq E$ denotes reachability via paths that do not pass through any of the edges in the set K.

We first introduce the notion of a *clip*, which is a specification of a set of CFG paths leading from a specific set of *entry edges* to a specific set of *exit edges*. It allows us to describe sets of paths with similar control flows.

Definition 3 (*Clip*) A *clip* $\mathcal{S}$ is a pair $[\![A, B]\!]$ consisting of a set $A \subseteq E$ of edges called *entry edges*, and a set $B \subseteq E$ of edges called *exit edges*.

Definition 4 (*Paths in a clip*) The set $paths(\mathcal{S})$ of *paths in a clip* $\mathcal{S} = [\![A, B]\!]$ is the set of all CFG paths that start with some entry edge in A, that end with some exit edge in B, and that do not contain any further entry or exit edges, i.e.,

$$\begin{aligned} paths(\mathcal{S}) = \{v_1 \dots v_n \mid\ & (v_1, v_2) \in A, (v_{n-1}, v_n) \in B, \\ & (v_i, v_{i+1}) \in E \setminus (A \cup B), 1 < i < n-1, n \geq 3\}. \end{aligned}$$

Note that by this definition each path in a clip has at least two edges.

Example 4 Reconsider the CFG from Example 1. Consider the clip

$$\mathcal{S}_0 = [\![\{(v_1, v_2)\}, \{(v_3, v_3), (v_3, v_{end})\}]\!].$$

Figure 3 illustrates this clip. The set of paths in $\mathcal{S}_0$ is

$$paths(\mathcal{S}_0) = \{v_1 v_2 v_3 v_3, v_1 v_2 v_3 v_{end}\}.$$

Note that $paths(\mathcal{S}_0)$ does not contain any longer paths, because the back edge of the loop is an exit edge.

A *context* of a node $v \in V$ is a clip $\mathcal{S}$ such that v has at most one *inner occurrence* in any path of $\mathcal{S}$. A context enables us to pinpoint a particular occurrence of a given node. Later we will use contexts to represent individual execution scenarios.

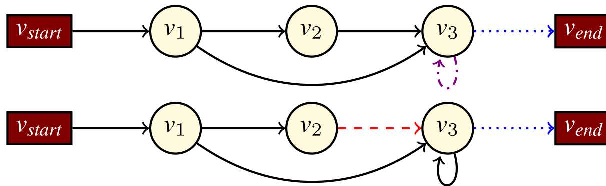


Fig. 4 Visualization of the clips $\mathcal{S}_1$ and $\mathcal{S}_2$ from Example 5. Entry edges are *dashed*, exit edges are *dotted*, and edges that are both entries and exits are *dotdashed*. The clip $\mathcal{S}_1$ is a context of node v_3, because both of its paths contains only a single inner occurrence of v_3. For clip $\mathcal{S}_2$, the back edge (v_3, v_3) is neither an entry nor an exit edge and may therefore occur arbitrarily often in the paths of $\mathcal{S}_2$. For example, $v_2 v_3 v_3 v_{end}$ is a path in $\mathcal{S}_2$, hence $\mathcal{S}_2$ is not a context

By inner occurrence we mean any occurrence except at the first or last node. It is a rather technical condition that is of no conceptual interest, but eliminates that the unwanted special case of v sitting at the beginning of an entry edge or at the end of an exit edge.

Definition 5 (*Context*) A *context* $\mathcal{C}$ *of a node* $v \in V$ is a clip, such that any path $v_1 \dots v_n$ in $\mathcal{C}$ contains at most one inner occurrence of v, i.e.,

$$v_1 \dots v_n \in paths(\mathcal{C}), v_i = v_j = v, 1 < i < n, 1 < j < n \implies i = j.$$

Example 5 Reconsider the CFG from Example 1. The clip $\mathcal{S}_1 = [\![A_1, B_1]\!]$ with

$$A_1 = \{(v_3, v_3)\}; \quad B_1 = \{(v_3, v_3), (v_3, v_{end})\}$$

is a context of node v_3, because none of its paths $paths(\mathcal{S}_1) = \{v_3 v_3 v_3, v_3 v_3 v_{end}\}$ contains more than one inner occurrence of v_3. The clip $\mathcal{S}_2 = [\![A_2, B_2]\!]$ with

$$A_2 = \{(v_2, v_3)\}; \quad B_2 = \{(v_3, v_{end})\}$$

is not a context, because the path $v_2 v_3 v_3 v_{end} \in paths(\mathcal{S}_2)$ has two inner occurrences of v_3. Figure 4 illustrates this example.

The following operators allow us measure the length of paths and to concatenate paths and edges:

Definition 6 (*Simple path operators*) The *length* $|v_1 \dots v_n|$ *of a path* $v_1 \dots v_n$ *consisting of nodes* $v_i \in V$ *for* $1 \leq i \leq n$ is defined as

$$|v_1 \dots v_n| = n.$$

The *concatenation* $v_1 \dots v_n \circ w_1 \dots w_m$ *of a path* $v_1 \dots v_n$ *consisting of nodes* $v_i \in V$ *for* $1 \leq i \leq n$ *with a path* $w_1 \dots w_m$ *consisting of nodes* $w_i \in V$ *for* $1 \leq i \leq m$ is defined as

$$v_1 \dots v_n \circ w_1 \dots w_m = v_1 \dots v_n w_1 \dots w_m.$$

The *concatenation* $v_1 \ldots v_n \circ (x, y)$ *of a path* $v_1 \ldots v_n$ *consisting of nodes* $v_i \in V$ *for* $1 \leq i \leq n$ *with an edge* $(x, y) \in E$ is defined as

$$v_1 \ldots v_n \circ (x, y) = v_1 \ldots v_n xy.$$

The *concatenation* $(x, y) \circ v_1 \ldots v_n$ *of an edge* $(x, y) \in E$ *with a path* $v_1 \ldots v_n$ *consisting of nodes* $v_i \in V$ *for* $1 \leq i \leq n$ is defined as

$$(x, y) \circ v_1 \ldots v_n = xyv_1 \ldots v_n.$$

We use contexts as execution scenarios. Requirement 2 asks for a classification scheme, i.e., all situations must be covered and scenarios must not overlap. Hence, we define appropriate notions of coverage and disjointness for contexts: *Flow coverage* catches the idea of capturing all possible control flows that a given node can be involved in; *divergence* captures the idea of contexts representing disjoint scenarios.

Definition 7 (*Flow coverage*) A set of paths X *covers* a node $v \in V$, iff, for all paths ρ, σ with $\rho \circ v \circ \sigma \in \mathcal{U}$ (recall that $\mathcal{U}$ is the set of all end-to-end paths in the CFG), there are subpaths ρ_1, σ_2 and non-empty subpaths ρ_2, σ_1, with $\rho_1 \circ \rho_2 = \rho$ and $\sigma_1 \circ \sigma_2 = \sigma$, such that $\rho_2 \circ v \circ \sigma_1 \in X$.

Informally speaking, this means that every occurrence of the node v in some end-to-end path is located inside a subpath that is in X.

Example 6 Reconsider the CFG from Example 1. We have

$$\mathcal{U} = \left\{v_{start} v_1 \pi v_{end}, v_{start} v_1 v_2 \pi v_{end} \mid \pi \in \{v_3\}^+\right\}.$$

Consider the two clips (cf. Fig. 5)

$$\begin{aligned} \mathcal{S}_3 &= [\![\{(v_1, v_2), (v_3, v_3)\}, \{(v_3, v_3), (v_3, v_{end})\}]\!], \text{ and} \\ \mathcal{S}_4 &= [\![\{(v_1, v_3), (v_1, v_2), (v_3, v_3)\}, \{(v_3, v_{end})\}]\!]. \end{aligned}$$

The paths of these clips are

$$\begin{aligned} paths(\mathcal{S}_3) &= \{v_1 v_2 v_3 v_3, v_1 v_2 v_3 v_{end}, v_3 v_3 v_3, v_3 v_3 v_{end}\}, \text{ and} \\ paths(\mathcal{S}_4) &= \left\{v_1 \pi v_{end}, v_1 v_2 \pi v_{end} \mid \pi \in \{v_3\}^+\right\}. \end{aligned}$$

The paths of clip $\mathcal{S}_4$ cover node v_3, i.e., every occurrence of v_3 in some end-to-end path is located inside a subpath that is an element of $paths(\mathcal{S}_4)$. The paths of $\mathcal{S}_3$ do not cover v_3, e.g., none of the paths in $paths(\mathcal{S}_3)$ is a subpath of $v_{start} v_1 v_3 v_{end} \in \mathcal{U}$.

Definition 8 (*Divergent paths*) Two paths π and σ are *divergent*, iff π and σ neither overlap on more than a single edge, nor one is a subpath of the other, i.e., *none* of the following conditions apply:

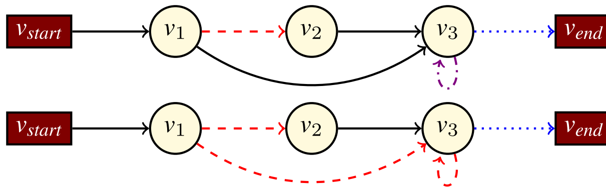


Fig. 5 Visualization of the clips $\mathcal{S}_3$ and $\mathcal{S}_4$ from Example 6. The paths of $\mathcal{S}_4$ cover node v_3, but the paths of $\mathcal{S}_3$ do not

- there exist paths α, β, γ with $\alpha \circ \beta = \pi$, $\beta \circ \gamma = \sigma$ and $|\beta| \geq 2$;
- there exist paths α, β, γ with $\alpha \circ \beta = \sigma$, $\beta \circ \gamma = \pi$ and $|\beta| \geq 2$;
- σ is a subpath of π;
- π is a subpath of σ.

Theorem 2 (Divergence of paths in clip) *Let $\mathcal{S} = [\![A, B]\!]$ be a clip with $\pi, \rho \in paths(\mathcal{S})$ and $\pi \neq \rho$. Then π and ρ are divergent.*

Definition 9 (*Divergent sets*) Two sets of paths X and Y are *divergent*, iff the paths π and σ are divergent, for any choice of $\pi \in X$ and $\sigma \in Y$.

Example 7 Reconsider clip

$$\mathcal{S}_4 = [\![\{(v_1, v_3), (v_1, v_2), (v_3, v_3)\}, \{(v_3, v_{end})\}]\!]$$

from Example 6. The paths of clip $\mathcal{S}_4$ and the paths of clip

$$\mathcal{S}_5 = [\![\{(v_2, v_3)\}, \{(v_3, v_{end})\}]\!]$$

are not divergent, because the path $v_2 v_3 v_{end} \in paths(\mathcal{S}_5)$ is a subpath of the path $v_1 v_2 v_3 v_{end} \in paths(\mathcal{S}_4)$. On the other hand, the paths of clip $\mathcal{S}_4$ and clip

$$\mathcal{S}_6 = [\![\{(v_1, v_3), (v_1, v_2)(v_3, v_3)\}, \{(v_3, v_3)\}]\!]$$

are divergent.

A *simple-history context* is a context that can easily be constructed from a CFG, fulfills flow coverage and divergence, and can serve as a starting point for subsequent refinement through *context splitting* operators.

Definition 10 (*Simple history clip*) The *simple-history clip* of a node $v \in V \setminus \{v_{start}, v_{end}\}$ is the clip

$$\mathcal{S} = [\![(Q \cup B) \cap R, B]\!],$$

where Q is the set of all edges $(v_{start}, q) \in E$ that start with the start node v_{start}, where B is the set of all outgoing edges $(v, b) \in E$ of v, and where R is the set of all edges $(w, r) \in E$, such that there is a path from r to v, i.e.,

$$Q = \{(v_{start}, q) \in E\}; \quad B = \{(v, b) \in E\}; \quad R = \{(w, r) \in E \mid (r, v) \in E^*\}.$$

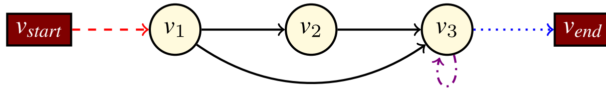


Fig. 6 Visualization of the simple history context $\mathcal{S}$ of node v_3 from Example 8. The paths of $\mathcal{S}$ cover v_3, i.e., every occurrence of node v_3 in some path from v_{start} to v_end is located inside a path of the context $\mathcal{S}$

Theorem 3 (Simple history context) *For any node $v \in V \setminus \{v_{start}, v_{end}\}$, the simple history clip $\mathcal{S} = [\![(Q \cup B) \cap R, B]\!]$ with*

$$Q = \{(v_{start}, q) \in E\}; \quad B = \{(v, b) \in E\}; \quad R = \{(w, r) \in E \mid (r, v) \in E^*\}$$

is a context, and $paths(\mathcal{S})$ covers v.

Example 8 Reconsider the CFG from Example 1. The simple-history context of node v_3 is the clip (cf. Fig. 6)

$$\mathcal{S} = \{(v_{start}, v_1), (v_3, v_3)\}, \{(v_3, v_3), (v_3, v_{end})\}.$$

Contexts can be refined by splitting them into multiple separate "subcontexts". We present two fundamental splitting operators:

- *Vertical context splitting* affords the separation of a chosen *set of subpaths* of an given context.
- *Horizontal context splitting* affords the separation of a chosen *subset of paths* of a given context.

Recursive application of these two splitting operators on an initial covering context—like the simple history context—allow us to obtain a set of fine-grained contexts for any given node $v \in V \setminus \{v_{start}, v_{end}\}$. Importantly, the splitting operators must be designed to preserve coverage. Moreover, the resulting contexts must be divergent, to allow the clean separation of scenarios demanded by Requirement 2. In the following we present our operators and demonstrate that they fulfill these properties.

Definition 11 (*Vertical context splitting*) Let $\mathcal{C} = [\![A, B]\!]$ be a context of node $v \in V$, and let F be the set of all edges that are neither entry nor exit nodes of $\mathcal{C}$, i.e., $F = E \setminus (A \cup B)$. Choose a set X of edges with

$$X \subseteq \left\{(x_1, x_2) \mid (a, u) \in A, (u, x_1) \in F^*, (x_1, x_2) \in F, (x_2, w) \in F^*, (w, b) \in B\right\}.$$

Vertical context splitting of $\mathcal{C}$ along X yields the two clips

$$\mathcal{C}_1 = [\![A, (B \cup X) \cap Y]\!], \text{ and } \mathcal{C}_2 = [\![X, (B \cup X) \cap Z]\!], \text{ where}$$
$$Y = \left\{(u, w) \mid (x, y) \in A, (y, u) \in (F \setminus X)^*, (u, w) \in E\right\}, \text{ and}$$
$$Z = \left\{(u, w) \mid (x, y) \in X, (y, u) \in (F \setminus X)^*, (u, w) \in E\right\}.$$

The following theorem establishes that the resulting clips are contexts:

Theorem 4 (Vertical context splitting) *Let $\mathcal{C}_1$ and $\mathcal{C}_2$ be the clips produced by splitting a context $\mathcal{C} = [\![A, B]\!]$ of some node $v \in V$ vertically along a set of edges X, i.e.,*

$$
\begin{aligned}
&\mathcal{C}_1 = [\![A, (B \cup X) \cap Y]\!], \text{ and } \mathcal{C}_2 = [\![X, (B \cup X) \cap Z]\!], \text{ where} \\
&Y = \left\{(u, w) \mid (x, y) \in A, (y, u) \in (F \setminus X)^*, (u, w) \in E\right\}, \\
&Z = \left\{(u, w) \mid (x, y) \in X, (y, u) \in (F \setminus X)^*, (u, w) \in E\right\}, \text{ and} \\
&F = E \setminus (A \cup B).
\end{aligned}
$$

Then all of the following assertions hold:

1. *$\mathcal{C}_1 = [\![A, (B \cup X) \cap Y]\!]$ and $\mathcal{C}_2 = [\![X, (B \cup X) \cap Z]\!]$ are contexts of v;*
2. *if $paths(\mathcal{C})$ covers v, then $paths(\mathcal{C}_1) \cup paths(\mathcal{C}_2)$ covers v;*
3. *$paths(\mathcal{C}_1)$ and $paths(\mathcal{C}_2)$ are divergent.*

Example 9 Reconsider the simple-history context

$$\mathcal{C}_{v_3} = \mathcal{S} = \{(v_{start}, v_1), (v_3, v_3)\}, \{(v_3, v_3), (v_3, v_{end})\}$$

of node v_3 from Example 8. There is a path from node $v_1 \in A$ to node v_1 that contains only edges in $E \setminus \{(v_{start}, v_1), (v_3, v_3), (v_3, v_{end})\}$, and there is a path from node v_2 to node v_3 that contains only edges from $E \setminus \{(v_{start}, v_1), (v_3, v_3), (v_3, v_{end})\}$, so we may choose

$$X = \{(v_1, v_2)\}.$$

Hence, we have

$$
\begin{aligned}
E \setminus (A \cup B \cup X) &= \{(v_1, v_3), (v_2, v_3)\}; \\
Y &= \{(v_1, v_2), (v_1, v_3), (v_3, v_3), (v_3, v_{end})\}; \\
Z &= \{(v_2, v_3), (v_3, v_3), (v_3, v_{end})\}.
\end{aligned}
$$

Therefore, we obtain subcontexts (cf. Fig. 7)

$$
\begin{aligned}
\mathcal{C}_{v_3,1.0} &= [\![A, (A \cup B \cup X) \cap Y]\!] \\
&= [\![\{(v_{start}, v_1), (v_3, v_3)\}, \{(v_1, v_2), (v_3, v_3), (v_3, v_{end})\}]\!]; \\
\mathcal{C}_{v_3,2.0} &= [\![X, (A \cup B \cup X) \cap Z]\!] = [\![\{(v_1, v_2)\}, \{(v_3, v_3), (v_3, v_{end})\}]\!].
\end{aligned}
$$

Definition 12 (*Horizontal context splitting*) Let $\mathcal{C} = [\![A, B]\!]$ be a context of node $v \in V$, and let $\mathcal{D}$ be a partition of A. For each set $D \in \mathcal{D}$, let Z_D be the set of all edges $(u, w) \in E$ that are reachable from some edge in D without crossing any entry or exit edge, i.e., there exists some edge $(d, x) \in D$ with a (possibly empty) path from node x to node u that contains only edges in $E \setminus (A \cup B)$.

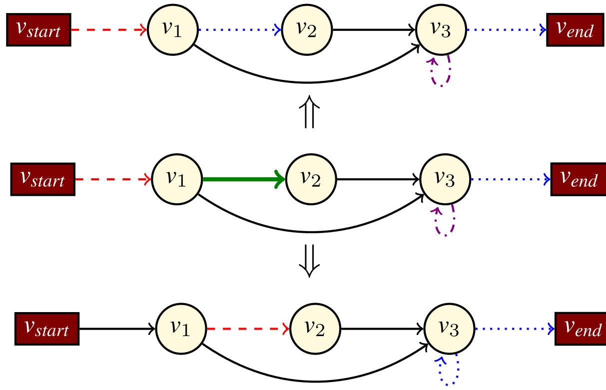


Fig. 7 Visualization of vertical context splitting from Example 9. The *thick edge* is the single split edge (v_1, v_2)

Horizontal context splitting of $\mathcal{C}$ *along* $\mathcal{D}$ yields the set of contexts $\{\mathcal{C}_D \mid D \in \mathcal{D}\}$ of v, where

$$\mathcal{C}_D = [\![D, B \cap Z_D]\!].$$

The fact that $\mathcal{C}_D$ is a context of v, for any $D \in \mathcal{D}$, is shown in the following theorem:

Theorem 5 (Horizontal context splitting) *Let* $\mathcal{C} = [\![A, B]\!]$ *be a context of node* $v \in V$*, and let* $\mathcal{D}$ *be a partition of* A*. For each set* $D \in \mathcal{D}$*, let* Z_D *be the set of all edges* $(u, w) \in E$*, such that there exists some edge* $(d, x) \in D$ *with a (possibly empty) path from node* x *to node* u *that contains only edges in* $E \setminus (A \cup B)$*. Then* $\mathcal{C}_D = [\![D, B \cap Z_D]\!]$ *is a context of* v*. Moreover, the sets* $paths(\mathcal{C}_{D_1})$ *and* $paths(\mathcal{C}_{D_2})$*, are divergent, for any sets* $D_1, D_2 \in \mathcal{D}$ *with* $D_1 \neq D_2$*. Furthermore, if* $W \cup paths(\mathcal{C})$ *covers node* v*, for any set of paths* W*, then* $W \cup \bigcup_{D \in \mathcal{D}} paths(\mathcal{C}_D)$ *covers node* v*.*

Example 10 Reconsider context

$$\mathcal{C}_{v_3,1.0} = [\![\{(v_{start}, v_1), (v_3, v_3)\}, \{(v_1, v_2), (v_3, v_3), (v_3, v_{end})\}]\!]$$

from Example 9. Choose the following partition of A:

$$\mathcal{D} = \{\{(v_{start}, v_1)\}, \{(v_3, v_3)\}\}.$$

We obtain the two new contexts of v_3 (cf. Fig. 8):

$$\mathcal{C}_{v_3,1.1} = [\![\{(v_{start}, v_1)\}, \{(v_1, v_2), (v_3, v_3), (v_3, v_{end})\}]\!];$$
$$\mathcal{C}_{v_3,1.2} = [\![\{(v_3, v_3)\}, \{(v_3, v_3), (v_3, v_{end})\}]\!].$$

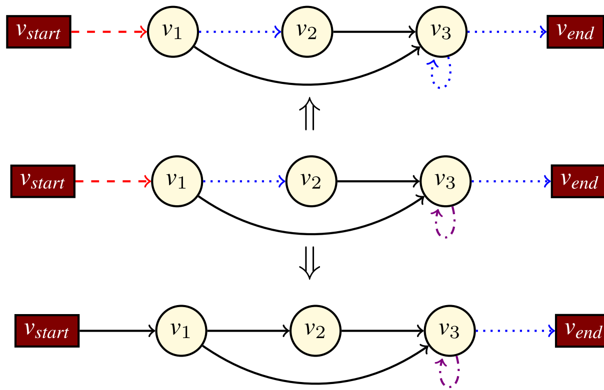


Fig. 8 Visualization of horizontal context splitting from Example 10

3.5 Context constraints

We first formalise the notion of *occurrence*. We start with rather straightforward definitions for occurrences of nodes and edges in a path.

Definition 13 (*Occurrences of a node*) The *set* $occ(v, \pi)$ *of occurrences of a node* $v \in V$ *in a path* π is defined as

$$occ(v, \pi) = \{(\rho, \sigma) \mid \rho \circ v \circ \sigma = \pi\}.$$

Example 11 Consider path $\pi = v_{start} v_1 v_3 v_3 v_3 v_{end}$. We have

$$occ(v_3, \pi) = \{(v_{start} v_1, v_3 v_3 v_{end}), (v_{start} v_1 v_3, v_3 v_{end}), (v_{start} v_1 v_3 v_3, v_{end})\}.$$

Theorem 6 *Let* $v \in V$ *be a node, and let* X *be a set of paths. Then*

$$\left| \bigcup_{\pi \in X} occ(v, \pi) \right| = \sum_{\pi \in X} \left| occ(v, \pi) \right|.$$

Definition 14 (*Occurrences of an edge*) The *set* $occ(e, \pi)$ *of occurrences of an edge* $e \in E$ *in a path* π is defined as

$$occ(e, \pi) = \{(\rho, \sigma) \mid \rho \circ e \circ \sigma = \pi\}.$$

Example 12 Reconsider path $\pi = v_{start} v_1 v_3 v_3 v_3 v_{end}$ from Example 11. We have

$$occ((v_3, v_3), \pi) = \{(v_{start} v_1, v_3 v_{end}), (v_{start} v_1 v_3, v_{end})\}).$$

Theorem 7 *Let $e \in E$ be an edge, and let X be a set of paths. Then*

$$\Big|\bigcup_{\pi \in X} occ(e, \pi)\Big| = \sum_{\pi \in X} \Big|occ(e, \pi)\Big|.$$

The notion of a *covered occurrence* of a node considers inner occurrences in some path of a given clip. Recall that our notion of an *inner occurrence* excludes the border nodes of a path.

Definition 15 (*Covered occurrence of a node*) Let $\mathcal{S}$ be a clip. The set $occ(v, \pi, \mathcal{S})$ *of $\mathcal{S}$-covered occurrences of a node $v \in V$ in a path π* is defined as

$$\begin{aligned} occ(v, \pi, \mathcal{S}) = \{(\rho_1, \rho_2, \sigma_1, \sigma_2) \mid \rho_1 \circ \rho_2 \circ v \circ \sigma_1 \circ \sigma_2 = \pi, \\ \rho_2 \neq \epsilon, \sigma_1 \neq \epsilon, \rho_2 \circ v \circ \sigma_1 \in paths(\mathcal{S})\}. \end{aligned}$$

Example 13 Reconsider the contexts

$$\begin{aligned} \mathcal{C}_{v_3,1.1} &= [\![\{(v_{start}, v_1)\}, \{(v_1, v_2), (v_3, v_3), (v_3, v_{end})\}]\!]; \\ \mathcal{C}_{v_3,1.2} &= [\![\{(v_3, v_3)\}, \{(v_3, v_3), (v_3, v_{end})\}]\!] \end{aligned}$$

from Example 10 and the path $\pi = v_{start} v_1 v_3 v_3 v_3 v_{end}$ from Example 12. We have

$$\begin{aligned} occ(v_3, \pi, \mathcal{C}_{v_3,1.1}) &= \{(\epsilon, v_{start} v_1 v_2, v_3, v_3 v_{end})\}; \\ occ(v_3, \pi, \mathcal{C}_{v_3,1.2}) &= \{(v_{start} v_1 v_2, v_3, v_3, v_{end}), (v_{start} v_1 v_2 v_3, v_3, v_{end}, \epsilon)\}. \end{aligned}$$

Theorem 8 *Let $v \in V$ be a node, let X be a set of paths, and let $\mathcal{S}$ be a clip. Then*

$$\Big|\bigcup_{\pi \in X} occ(v, \pi, \mathcal{S})\Big| = \sum_{\pi \in X} \big|occ(v, \pi, \mathcal{S})\big|.$$

The occurrence of a node are related to to the occurrence of its contexts, via linear constraints.

Theorem 9 (Relating nodes to context) *Let $\mathcal{C}_1, \ldots, \mathcal{C}_n$ be pairwise divergent contexts of some node $v \in V$, such that $\bigcup_{1 \leq i \leq n} paths(\mathcal{C}_i)$ covers v. Then the following constraint holds:*

$$|occ(v, \pi)| = \sum_{1 \leq i \leq n} \Big|occ(v, \pi, \mathcal{C}_i)\Big|, \text{ for all } \pi \in \mathcal{U}.$$

Lastly, the occurrences of a context are related to the occurrence of its entry and exit edges, again via linear constraints.

Theorem 10 (Relating contexts to entries and exits) *Let $\mathcal{C} = [\![A, B]\!]$ be a context of some node $v \in V \setminus \{v_{start}, v_{end}\}$ that is neither the start node, nor the end node.*

Let X be the set of all edges $(x, z) \in E$, such that there exists an edge $(a, w) \in A$ with a path from node w to node x that contains only edges in $E \setminus (A \cup B)$, such that

there exists an edge $(u, v) \in E \setminus (A \cup B)$ *with a path from node* x *to node* u *that contains only edges in* $E \setminus (A \cup B)$*, and such there exists no path from node* x *to node* v *that contains node* z *and only edges in* $E \setminus (A \cup B)$*.*

Let Y *be the set of all edges* $(z, y) \in E$*, such that there exists an edge* $(w, b) \in B$ *with a path from node* y *to node* w *that contains only edges in* $E \setminus (A \cup B)$*, such that there exists an edge* $(v, u) \in E \setminus (A \cup B)$ *with a path from node* u *to node* y *that contains only edges in* $E \setminus (A \cup B)$*, and such there exists no path from node* v *to node* y *that contains node* z *and only edges in* $E \setminus (A \cup B)$*.*

Then the following constraints hold for every path $\pi \in paths(\mathcal{C})$*:*

$$|occ(v, \pi, \mathcal{C})| \leq \sum_{a \in A} |occ(a, \pi)| - \sum_{x \in X} |occ(x, \pi)|;$$
$$|occ(v, \pi, \mathcal{C})| \leq \sum_{b \in B} |occ(b, \pi)| - \sum_{y \in Y} |occ(y, \pi)|.$$

Example 14 Reconsider contexts

$$\mathcal{C}_{v_3,1.0} = [\![\{(v_{start}, v_1), (v_3, v_3)\}, \{(v_1, v_2), (v_3, v_3), (v_3, v_{end})\}]\!];$$
$$\mathcal{C}_{v_3,2.0} = [\![\{(v_1, v_2)\}, \{(v_3, v_3), (v_3, v_{end})\}]\!];$$
$$\mathcal{C}_{v_3,1.1} = [\![\{(v_{start}, v_1)\}, \{(v_1, v_2), (v_3, v_3), (v_3, v_{end})\}]\!];$$
$$\mathcal{C}_{v_3,1.2} = [\![\{(v_3, v_3)\}, \{(v_3, v_3), (v_3, v_{end})\}]\!].$$

of node v_3 from Examples 9 and 10.

For context $\mathcal{C}_{v_3,1.0}$ we have $X = \{(v_1, v_2)\}$ and $Y = \emptyset$, yielding

$$|occ(v_3, \pi, \mathcal{C}_{v_3,1.0})| \leq |occ((v_{start}, v_1), \pi)| + |occ((v_3, v_3), \pi)| - |occ((v_1, v_2), \pi)|;$$
$$|occ(v_3, \pi, \mathcal{C}_{v_3,1.0})| \leq |occ((v_3, v_3), \pi)| + |occ((v_3, v_{end}), \pi)|.$$

For context $\mathcal{C}_{v_3,2.0}$, we have $X = \emptyset$ and $Y = \emptyset$, yielding

$$|occ(v_3, \pi, \mathcal{C}_{v_3,2.0})| \leq |occ((v_1, v_2), \pi)|;$$
$$|occ(v_3, \pi, \mathcal{C}_{v_3,2.0})| \leq |occ((v_3, v_3)), \pi)| + |occ((v_3, v_{end}), \pi)|.$$

For context $\mathcal{C}_{v_3,1.1}$ we have $X = \{(v_1, v_2)\}$ and $Y = \emptyset$, yielding

$$|occ(v_3, \pi, \mathcal{C}_{v_3,1.1})| \leq |occ((v_{start}, v_1), \pi)| - |occ((v_1, v_2), \pi)|;$$
$$|occ(v_3, \pi, \mathcal{C}_{v_3,1.1})| \leq |occ((v_3, v_3), \pi)| + |occ((v_3, v_{end}), \pi)|.$$

For context $\mathcal{C}_{v_3,1.2}$ we have $X = \emptyset$ and $Y = \emptyset$, yielding

$$|occ(v_3, \pi, \mathcal{C}_{v_3,1.2})| \leq |occ((v_3, v_3), \pi)|;$$
$$|occ(v_3, \pi, \mathcal{C}_{v_3,1.2})| \leq |occ((v_3, v_3), \pi)| + |occ((v_3, v_{end}), \pi)|.$$

3.6 Timed traces and clips

In this section we describe how execution times of nodes can be obtained from a database of timed execution traces.

A timed trace indicates the execution sequence of nodes during a particular run of the code on the target platform, together with the execution duration for each occurrence of each node in the sequence.

Definition 16 (*Timed trace*) A *timed trace of a program* $\mathcal{P}$ is a finite sequence $\pi = (v_1, t_1) \dots (v_n, t_n)$, where $v_1 \dots v_n$ is a path in the program's CFG $\mathcal{G}_\mathcal{P} = (V, E, v_{start}, v_{end})$, and where $t_1, \dots, t_n$ are the associated observed execution times of $v_1, \dots, v_n$.

The *maximal observed execution time* (MOET) of a node within a timed trace is the maximal execution time that is associated with any occurrence of the node inside the trace. By *inside* a trace we mean anything *between* the first and the last node, but we do not include the border nodes.

Definition 17 (*MOET of node in timed trace*) The *maximal observed execution time* (MOET) $moet_{v,\pi}$ *of a node* $v \in V$ inside a timed trace π is defined as the maximum over all associated execution times of v occurring inside π, i.e.,

$$moet_{v,\pi} = max\,\{t_i \mid \pi = (v_1, t_1) \dots (v_n, t_n), v_i = v, 1 < i < n\}.$$

Note that $moet_{v,\pi}$ is undefined, if there is no occurrence of v in π.

Example 15 Consider the timed traces

$$\pi_1 = (v_{start}, 0)(v_1, 40)(v_3, 20)(v_{end}, 0); \quad \pi_2 = (v_3, 5)(v_3, 4)(v_3, 4)(v_3, 4);$$
$$\pi_3 = (v_{start}, 0)(v_1, 40)(v_3, 25)(v_{end}, 0); \quad \pi_4 = (v_{start}, 0)(v_1, 40)(v_2, 20);$$
$$\pi_5 = (v_{start}, 0)(v_1, 40)(v_3, 30)(v_3, 20)(v_{end}, 0); \quad \pi_6 = (v_3, 5);$$
$$\pi_7 = (v_{start}, 0)(v_1, 45)(v_2, 15)(v_3, 10)(v_{end}, 0).$$

The MOETs of node v_3 are as follows:

$$moet_{v_3,\pi_1} = 20; \quad moet_{v_3,\pi_2} = 4;$$
$$moet_{v_3,\pi_3} = 25; \quad moet_{v_3,\pi_4} \text{ undefined};$$
$$moet_{v_3,\pi_5} = 30; \quad moet_{v_3,\pi_6} \text{ undefined};$$
$$moet_{v_3,\pi_7} = 10.$$

We lift the notion of the MOET of a node in a path to sets of timed traces:

Definition 18 (*MOET of node in set of traces*) The *maximal observed execution time (MOET)* $moet_{v,\mathcal{T}}$ *of a node* $v \in V$ *over a set of timed traces* $\mathcal{T}$ is defined as the

maximum of all maximal observed execution times of $v \in V$ in any timed trace $\pi \in \mathcal{T}$, i.e.,

$$moet_{v,\mathcal{T}} = \max\left\{moet_{v,\pi} \mid \pi \in \mathcal{T}\right\}.$$

Note that $moet_{v,\mathcal{T}}$ is undefined, if none of the timed traces in $\mathcal{T}$ contains an occurrence of v.

Example 16 Reconsider the timed traces $\pi_1, \ldots, n_7$ from Example 15. The MOET of node v_3 over the set $\mathcal{T} = \{\pi_1, \ldots, \pi_7\}$ of timed traces is

$$moet_{v_3,\mathcal{T}} = 30.$$

Definition 19 (*Untimed trace*) The corresponding *untimed trace* $\overline{\pi}$ *of a timed trace* $\pi = (v_1, t_1) \ldots (v_n, t_n)$ is the sequence of nodes occurring in π, i.e.,

$$\overline{\pi} = v_1 \ldots v_n.$$

Example 17 Reconsider the times traces $\pi_1, \ldots, \pi_7$ from Example 15. The corresponding untimed traces are

$$\begin{aligned} \overline{\pi}_1 &= v_{start} v_1 v_3 v_{end}; & \overline{\pi}_2 &= v_3 v_3 v_3 v_3; \\ \overline{\pi}_3 &= v_{start} v_1 v_3 v_{end}; & \overline{\pi}_4 &= v_{start} v_1 v_2; \\ \overline{\pi}_5 &= v_{start} v_1 v_3 v_3 v_{end}; & \overline{\pi}_6 &= v_3; \\ \overline{\pi}_7 &= v_{start} v_1 v_2 v_3 v_{end}. \end{aligned}$$

Definition 20 (*MOET of node in clip*) The *maximal observed execution time (MOET)* $moet_{v,\mathcal{S},\mathcal{T}}$ *of a node* $v \in V$ *in clip* $\mathcal{S}$ *over a set of timed traces* $\mathcal{T}$ is the MOET of v over the set of all timed subtraces in $\mathcal{T}$ with corresponding untimed traces that are paths in $\mathcal{S}$, i.e.,

$$moet_{v,\mathcal{S},\mathcal{T}} = \max\{moet_{v,\pi} \mid \overline{\pi} \in paths(\mathcal{S}), \sigma \circ \pi \circ \rho \in \mathcal{T}\}.$$

Note that $moet_{v,\mathcal{S},\mathcal{T}}$ is undefined, if none of the timed traces in $\mathcal{T}$ contains an occurrence of v, or if none of the timed traces in $\mathcal{T}$ that contain an occurrence of v has a corresponding untimed trace that is a path in clip $\mathcal{S}$.

Example 18 Recall that a context $\mathcal{C}$ of a node v is a clip with a constraint on the number of times that v may occur within the paths of $\mathcal{C}$. So reconsider contexts

$$\begin{aligned} \mathcal{C}_{v_3,1.0} &= [\![\{(v_{start}, v_1), (v_3, v_3)\}, \{(v_1, v_2), (v_3, v_3), (v_3, v_{end})\}]\!]; \\ \mathcal{C}_{v_3,2.0} &= [\![\{(v_1, v_2)\}, \{(v_3, v_3), (v_3, v_{end})\}]\!]; \\ \mathcal{C}_{v_3,1.1} &= [\![\{(v_{start}, v_1)\}, \{(v_1, v_2), (v_3, v_3), (v_3, v_{end})\}]\!]; \\ \mathcal{C}_{v_3,1.2} &= [\![\{(v_3, v_3)\}, \{(v_3, v_3), (v_3, v_{end})\}]\!] \end{aligned}$$

of node v_3 from Examples 9 and 10. Also reconsider the timed traces $\pi_1, \ldots, \pi_7$ from Example 15. The MOETs of node v_3 within these contexts over the set $\mathcal{T} = \{\pi_1, \ldots, \pi_7\}$ of timed traces are

$$moet_{v3,\mathcal{C}_{v_3,1.0},\mathcal{T}} = 30; \qquad moet_{v3,\mathcal{C}_{v_3,2.0},\mathcal{T}} = 10;$$
$$moet_{v3,\mathcal{C}_{v_3,1.1},\mathcal{T}} = 30; \qquad moet_{v3,\mathcal{C}_{v_3,1.2},\mathcal{T}} = 4.$$

Theorem 11 (MOET reduction) *Let $\mathcal{S}$ be a clip, and let $\mathcal{T}$ be a set of timed traces. Then*

$$moet_{v,\mathcal{S},\mathcal{T}} \leq moet_{v,\mathcal{T}}.$$

3.7 Finding contexts for MBTA

In this section we describe an algorithm for obtaining, for any given node $v \in V \setminus \{v_{start}, v_{end}\}$, a set $\{\mathcal{C}_{v,1}, \ldots \mathcal{C}_{v,n(v)}\}$ of contexts of v, with pairwise divergent sets of paths that together cover v. The contexts are constructed in such a way that they are associated with different maximal observed execution times.

To construct a set of contexts for some node $v \in V \setminus \{v_{start}, v_{end}\}$, the algorithm checks the MOET $moet_{v,\mathcal{C},\mathcal{T}}$ of v over the given set $\mathcal{T}$ of timed traces in various candidate contexts $\mathcal{C}$,

We have noted before that $moet_{v,\mathcal{C},\mathcal{T}}$ needs not be defined under all circumstances: $moet_{v,\mathcal{C},\mathcal{T}}$ is undefined, if none of the timed traces in $\mathcal{T}$ contains an occurrence of v, or if none of the timed traces in $\mathcal{T}$ that contain an occurrence of v has a corresponding untimed trace that is a path in context $\mathcal{C}$.

In MBTA the set $\mathcal{T}$ of timed traces is obtained by performing measurements. In that case, $moet_{v,\mathcal{C},\mathcal{T}}$ is undefined, if node v is unreachable, or if none of the paths in $\mathcal{C}$ was exhibited by any measurement.

There are two basic strategies for handling missing measurements:

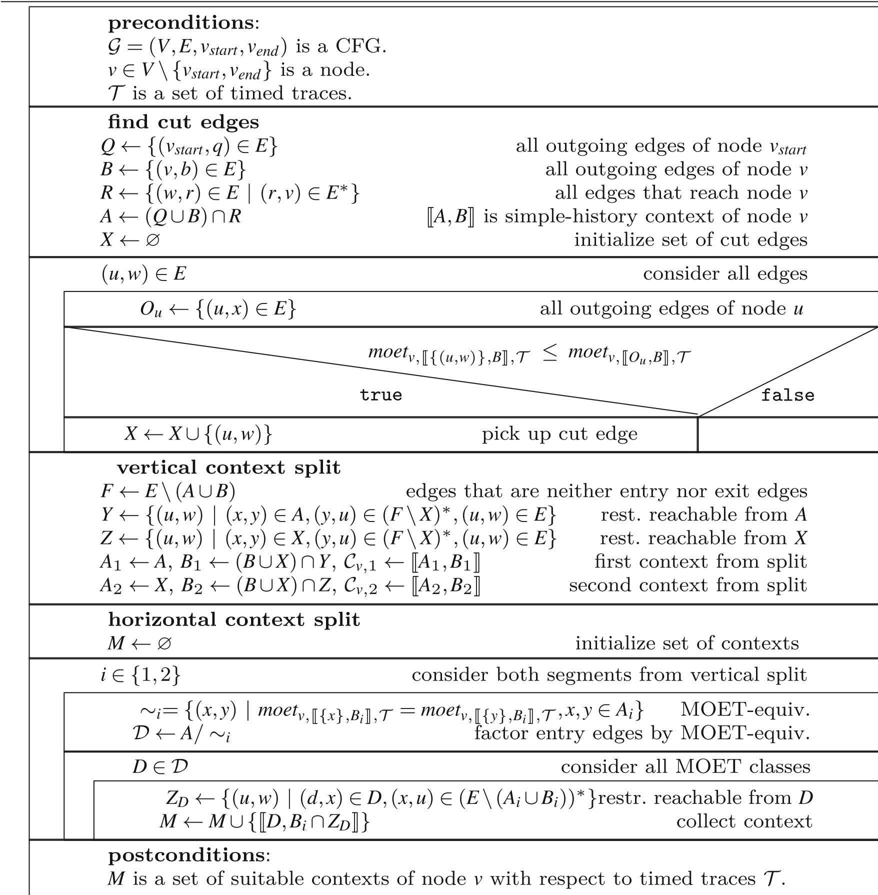
Conservative approach: In this approach, the algorithm by default attributes missing measurements to insufficient coverage of the temporal behavior. It assumes that suitable timed traces can, in principle, be found, and conservatively substitutes the global MOET $moet_{v,\mathcal{T}}$ for $moet_{v,\mathcal{S},\mathcal{T}}$.

Progressive approach: In this approach, the algorithm by default attributes missing measurements to infeasible paths. It assumes that suitable timed traces can, in principle, not be found, substitutes 0 for $moet_{v,\mathcal{S},\mathcal{T}}$, and stipulates that the corresponding clip is infeasible.

However, to simplify the presentation of the algorithm, we just assume that $moet_{v,\mathcal{S},\mathcal{T}}$ is always defined, i.e., there is always at least one matching measurement.

In terms of coverage, this assumption means that the database $\mathcal{T}$ contains, for each node v and each segment $\mathcal{S}$, at least one trace that first passes through any entry edge of $\mathcal{S}$, then passed through v, and eventually passes through any exit edge of $\mathcal{S}$. Between the entry and the exit edge, the path must not pass through any further entry or exit edges.

Algorithm 3.1 Finding a set of contexts (presented in the form of a Nassi–Shneiderman diagram (1973)).



Algorithm 3.1 is a formal description of our algorithm. More informally, our algorithm proceeds as follows:

1. The algorithm initially finds the set Q of all edges $(v_{start}, q) \in E$, the set B of all edges $(v, b) \in E$, and the set R of all edges $(w, r) \in E$, such that there is a path from r to v. Set R can easily be found by performing a backward depth-first search, starting from node v. Sets Q and B can be found from an adjacency list or adjacency matrix of the CFG.
2. Let $A = (Q \cup B) \cap R$. Note that $\mathcal{C} = [\![A, B]\!]$ is the simple-history context of v.
3. For each edge $(u, w) \in E$, the algorithm checks the condition

$$moet_{v,[\![\{(u,w)\},B]\!],\mathcal{T}} \leq moet_{v,[\![O_u,B]\!],\mathcal{T}},$$

where $O_u = \{(u, x) \in E\}$ is the set of all outgoing edges of node u. The condition is a test if context $[\![\{(u, w)\}, B]\!]$ of v—which has edge (u, w) as its only entry edge—provides a lower MOET for node v than context $[\![O_u, B]\!]$ of v—which

has *all* edges starting from node u as entry edges. If this is true, then the context $[\![\{(u, w)\}, B]\!]$ captures a case of executing v with a reduced execution time. Let X be the set of all edges $(u, w) \in E$ for which the condition holds. Consider $[\![X, B]\!]$ as a separate context of v.

4. The next step of the algorithm is a vertical context split: The algorithm finds the set Y of all edges $(u, w) \in E$ such that there exists a path from some edge in A to node u that contains only edges in $E \setminus (A \cup B \cup X)$. It also finds the set Z of all edges (u, w) such that there exists a path from some edge in X to node u that contains only edges in $E \setminus (A \cup B \cup X)$. Let $A_1 = A$, $B_1 = (B \cup X) \cap Y$, $A_2 = X$, and $B_2 = (B \cup X) \cap Z$. Note that $\mathcal{C}_{v,1} = [\![A_1, B_1]\!]$ and $\mathcal{C}_{v,2} = [\![A_2, B_2]\!]$ are contexts with $paths(\mathcal{C}_{v,1}) \cap paths(\mathcal{C}_{v,2}) = \emptyset$ of v that cover node v.
5. The final step of the algorithm is a horizontal split of context $\mathcal{C}_{v,i}$, for $i \in \{1, 2\}$: In this step the algorithm creates a partition $\mathcal{D}_i$ of set A_i by the MOET of node v, i.e., $\mathcal{D}_i = A_i / \sim_i$, where $\sim_i$ is the following equivalence relation:

$$x \sim_i y \quad \text{iff} \quad moet_{v,[\![\{x\},B_i]\!],\mathcal{T}} = moet_{v,[\![\{y\},B_i]\!],\mathcal{T}} \quad \text{for all } x, y \in A_i .$$

   For each set $D \in \mathcal{D}_i$, the algorithm finds the set Z_D of all edges $(u, w) \in E$, such that there exists a path from some edge in D to node u that contains only edges in $E \setminus (A_i \cup B_i)$.
6. The set of contexts produced by the algorithm is

$$M = \left\{[\![D, B_1 \cap Z_D]\!] \mid D \in \mathcal{D}_1\right\} \cup \left\{[\![D, B_2 \cap Z_D]\!] \mid D \in \mathcal{D}_2\right\} .$$

Example 19 Reconsider CFG $\mathcal{G}$ from Example 1. Also, reconsider the timed traces

$$\pi_1 = (v_{start}, 0)(v_1, 40)(v_3, 20)(v_{end}, 0); \qquad \pi_2 = (v_3, 5)(v_3, 4)(v_3, 4)(v_3, 4);$$
$$\pi_3 = (v_{start}, 0)(v_1, 40)(v_3, 25)(v_{end}, 0); \qquad \pi_4 = (v_{start}, 0)(v_1, 40)(v_2, 20);$$
$$\pi_5 = (v_{start}, 0)(v_1, 40)(v_3, 30)(v_3, 20)(v_{end}, 0); \qquad \pi_6 = (v_3, 5);$$
$$\pi_7 = (v_{start}, 0)(v_1, 45)(v_2, 15)(v_3, 10)(v_{end}, 0)$$

from Example 15. We apply Algorithm 3.1 on CFG $\mathcal{G}$, node v_3, and the set of timed traces $\mathcal{T} = \{\pi_1, \ldots, \pi_7\}$:

1. The algorithm initially finds

$$Q = \{(v_{start}, v_1)\}; \quad B = \{(v_3, v_3), (v_3, v_{end})\};$$
$$R = \{(v_{start}, v_1), (v_1, v_2), (v_1, v_3), (v_2, v_3), (v_3, v_3)\} .$$

2. The algorithm sets

$$A = (Q \cup B) \cap R = \{(v_{start}, v_1), (v_3, v_3)\} ,$$

   The clip $\mathcal{C}_{v_3} = [\![A, B]\!]$ is indeed the simple-history context of v_3.

3. The algorithm finds

$$moet_{v_3,[\![\{(v_1,v_2)\},B]\!]} = 10 \leq moet_{v_3,[\![\{(v_1,v_2),(v_1,v_3)\},B]\!]} = 30;$$
$$X = \{(v_1, v_2)\}.$$

4. The Algorithm performs a vertical context split along X:

$$\begin{aligned}
Y &= \{(v_1, v_2), (v_1, v_3), (v_3, v_3), (v_3, v_{end})\};\\
Z &= \{(v_2, v_3), (v_3, v_3), (v_3, v_{end})\};\\
A_1 &= A = \{(v_{start}, v_1), (v_3, v_3)\};\\
B_1 &= (B \cup X) \cap Y = \{(v_1, v_2), (v_3, v_3), (v_3, v_{end})\};\\
A_2 &= X = \{(v_1, v_2)\};\\
B_2 &= (B \cup X) \cap Z = \{(v_3, v_3), (v_3, v_{end})\}.\\
\mathcal{C}_{v,1} &= [\![A_1, B_1]\!]\\
\mathcal{C}_{v,2} &= [\![A_2, B_2]\!]
\end{aligned}$$

5. The algorithm finds

$$\begin{aligned}
moet_{v_3,[\![\{(v_{start},v_1)\},B_1]\!]} = 30 \neq moet_{v_3,[\![\{(v_3,v_3)\},B_2]\!]} &= 4;\\
\mathcal{D}_1 &= \{\{(v_{start}, v_1)\}, \{(v_3, v_3)\}\}\,;\\
\mathcal{D}_2 &= \{\{(v_1, v_2)\}\}\,;\\
Z_{\{(v_{start},v_1)\}} &= \{(v_1, v_2), (v_1, v_3), (v_3, v_3), (v_3, v_{end})\}\,;\\
Z_{\{(v_3,v_3)\}} &= \{(v_3, v_3), (v_3, v_{end})\}\,;\\
Z_{\{(v_1,v_2)\}} &= \{(v_2, v_3), (v_3, v_3), (v_3, v_{end})\}\,;
\end{aligned}$$

6. The algorithm produces the set of contexts

$$\begin{aligned}
M =& \Big\{[\![\{(v_{start}, v_1)\}, B_1 \cap Z_{\{(v_{start},v_1)\}}]\!],\\
&[\![\{(v_3, v_3)\}, B_1 \cap Z_{\{(v_3,v_3)\}}]\!], [\![\{(v_1, v_2)\}, B_2 \cap Z_{\{(v_1,v_2)\}}]\!]\Big\}\\
=& \Big\{[\![\{(v_{start}, v_1)\}, \{(v_1, v_2), (v_3, v_3), (v_3, v_{end})\}]\!],\\
&[\![\{(v_3, v_3)\}, \{(v_3, v_3), (v_3, v_{end})\}]\!], [\![\{(v_1, v_2)\}, \{(v_3, v_3), (v_3, v_{end})\}]\!]\Big\}\,.
\end{aligned}$$

Theorem 12 *Given a CFG $\mathcal{G}_\mathcal{P} = (V, E, v_{start}, v_{end})$, a node $v \in V \setminus \{v_{start}, v_{end}\}$, and a set of timed traces $\mathcal{T}$, Algorithm* 3.1 *returns a set of contexts $M = \{\mathcal{C}_{v,1}, \ldots, \mathcal{C}_{v,n}\}$ of node v, such that $paths(\mathcal{C}_{v,i})$ and $paths(\mathcal{C}_{v,j})$ are divergent, for $1 \leq i \leq n$, $1 \leq j \leq n$, and $i \neq j$, and such that $\bigcup_{1 \leq i \leq n} paths(\mathcal{C}_{v,i})$ covers node v.*

3.8 Instantiating context-sensitive IPET

We are now able to put the results from Sects. 3.4 through 3.7 together, to obtain an instantiation of context-sensitive IPET:

1. We use Algorithm 3.1 to generate a set $\mathcal{Q}_v = \{\mathcal{C}_{v,1}, \ldots \mathcal{C}_{v,n(v)}\}$ of suitable contexts, for every node $v \in V \setminus \{v_{start}, v_{end}\}$. By way of Theorem 12, $paths(\mathcal{C}_{v,i})$ and $paths(\mathcal{C}_{v,j})$ are divergent, for $1 \leq i \leq n(v)$, $1 \leq j \leq n(n)$, and $i \neq j$, and $\bigcup_{1 \leq i \leq n(v)} paths(\mathcal{C}_{v,i})$ covers node v. Therefore, Theorem 9 applies, hence Requirement 2 is met.
2. For each node $v \in V \setminus \{v_{start}, v_{end}\}$, we interpret the individual contexts $\mathcal{C}_{v,1}, \ldots \mathcal{C}_{v,n(v)}$ as individual execution scenarios $E_{v,1}, \ldots E_{v,n(v)}$.
3. We use the MOETs $moet_{v,\mathcal{C}_{v,1},\mathcal{T}}, \ldots, moet_{v,\mathcal{C}_{v,n(v)},\mathcal{T}}$ of each context as WCET estimates $\widetilde{wcet}_{v,1}, \ldots, \widetilde{wcet}_{v,n(v)}$.
4. We use the construction in Theorem 10 to infer ILP constraints over our execution scenario variables. The translation of the linear constraints presented in the theorem is straightforward: For example, the linear constraint

$$|occ(v, \pi, \mathcal{C})| \leq \sum_{e \in A} |occ(e, \pi)| - \sum_{e \in X} |occ(e, \pi)|, \quad \text{for any path } \pi \in paths(\mathcal{C}),$$

translates to a corresponding IPET constraint

$$f_{v,i} \leq \sum_{e \in A} f_e - \sum_{e \in X} f_e.$$

By adding these constraints, we fulfill Requirement 3. Moreover, Requirement 1 is fulfilled as a consequence of Theorem 11.

Example 20 Reconsider CFG $\mathcal{G}$ from Example 1. Example 2 provides an IPET problem for $\mathcal{G}$, constructed for some hypothetical WCET estimates of the individual nodes. We can reuse the constraints from that IPET problem to construct a context-sensitive IPET problem for the timed trace $\mathcal{T} = \{\pi_1, \ldots, \pi_7\}$ from Example 1.

1. The latter example has already illustrated the application of Algorithm 3.1, to obtain a set

$$M = \left\{\mathcal{C}_{v_3,1.1}, \mathcal{C}_{v_3,1.2}, \mathcal{C}_{v_3,2.0}\right\}$$

of suitable contexts for node v_3, where

$$\begin{aligned}
\mathcal{C}_{v_3,1.1} &= [\![\{(v_{start}, v_1)\}, \{(v_1, v_2), (v_3, v_3), (v_3, v_{end})\}]\!];\\
\mathcal{C}_{v_3,1.2} &= [\![\{(v_3, v_3)\}, \{(v_3, v_3), (v_3, v_{end})\}]\!];\\
\mathcal{C}_{v_3,2.0} &= [\![\{(v_1, v_2)\}, \{(v_3, v_3), (v_3, v_{end})\}]\!].
\end{aligned}$$

2. We interpret the individual contexts $\mathcal{C}_{v_3,1.1}, \mathcal{C}_{v_3,1.2}, \mathcal{C}_{v_3,2.0}$ as individual execution scenarios $E_{v_3,1}, E_{v_3,2}, E_{v_3,3}$ with associated variables $f_{v_3,1}, f_{v_3,2}, f_{v_3,3}$.
3. We use the MOETs

$$moet_{v_3,\mathcal{C}_{v_3,1.1},\mathcal{T}} = 30; \quad moet_{v_3,\mathcal{C}_{v_3,1.2},\mathcal{T}} = 4; \quad moet_{v_3,\mathcal{C}_{v_3,2.0},\mathcal{T}} = 10.$$

of contexts $\mathcal{C}_{v_3,1.1}, \mathcal{C}_{v_3,1.2}, \mathcal{C}_{v_3,2.0}$—which we have calculated in Example 15—as WCET estimates $\widetilde{wcet}_{v_3,1}, \widetilde{wcet}_{v_3,2}, \widetilde{wcet}_{v_3,3}$.

4. We use the construction in Theorem 10 to infer the IPET constraints. We have already calculated the linear constraints

$$
\begin{aligned}
&|occ(v_3, \pi, \mathcal{C}_{v_3,1.1})| \leq |occ((v_{start}, v_1), \pi)| - |occ((v_1, v_2), \pi)|;\\
&|occ(v_3, \pi, \mathcal{C}_{v_3,1.1})| \leq |occ((v_3, v_3), \pi)| + |occ((v_3, v_{end}), \pi)|;\\
&|occ(v_3, \pi, \mathcal{C}_{v_3,1.2})| \leq |occ((v_3, v_3), \pi)|;\\
&|occ(v_3, \pi, \mathcal{C}_{v_3,1.2})| \leq |occ((v_3, v_3), \pi)| + |occ((v_3, v_{end}), \pi)|;\\
&|occ(v_3, \pi, \mathcal{C}_{v_3,2.0})| \leq |occ((v_1, v_2), \pi)|;\\
&|occ(v_3, \pi, \mathcal{C}_{v_3,2.0})| \leq |occ((v_3, v_3), \pi)| + |occ((v_3, v_{end}), \pi)|.
\end{aligned}
$$

in Example 14. These translate to corresponding IPET constraints

$$
\begin{aligned}
&f_{v_3,1} \leq f_{(v_{start},v_1)} - f_{(v_1,v_2)}; &&f_{v_3,1} \leq f_{(v_3,v_3)} + f_{(v_3,v_{end})};\\
&f_{v_3,2} \leq f_{(v_3,v_3)}; &&f_{v_3,2} \leq f_{(v_3,v_3)} + f_{(v_3,v_{end})};\\
&f_{v_3,3} \leq f_{(v_1,v_2)}; &&f_{v_3,3} \leq f_{(v_3,v_3)} + f_{(v_3,v_{end})}.
\end{aligned}
$$

4 Experimental evaluation

4.1 The FORTAS high-precision MBTA framework

We have implemented our approach for calculating WCET estimates as a component of the FORTAS framework (Zolda 2012). The FORTAS framework is a prototypical implementation of a portable, high-precision MBTA toolchain.

We call the FORTAS framework portable, because it can easily be adapted for different target architectures, essentially by implementing a driver that can execute the program under analysis on the desired target and collect a timed execution trace. We call the FORTAS framework a high precision analysis framework, because it incorporates a method for reducing the effect of underestimation emerging from insufficient measurement coverage (Bünte et al. 2011) as well as the method for reducing the effect of overestimation presented in this paper.

Figure 9 illustrates how our approach fits into the analysis workflow.

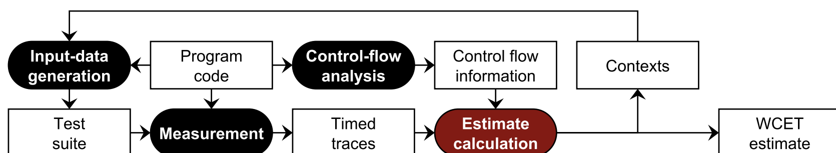


Fig. 9 The FORTAS framework features an adaptive refinement loop, where automatically inferred contexts (cf. Sect. 3.4) guide the generation of additional input vectors for subsequent measurement runs. These subsequent measurement runs yield additional timed traces

The Framework currently provides a backend for the Infineon TriCore TC1796, a fairly complex 32-bit microprocessor targeted at the automotive market that features basic versions of many performance-enhancing features found in modern desktop and server processors, like caching, pipelining, and branch prediction. For example, the TC1796 features a simple static rather than a more sophisticated dynamic branch predictor.

The TriCore TC1796 has a single processing core, yet allows parallel processing of different types of instructions via three parallel instruction pipelines and a separate floating point unit. The processing core includes a special debugging interface, which allows us to non-intrusively capture cycle-accurate timed traces using a *Lauterbach PowerTrace* device.[1]

As processor platform we are using a TriBoard TC179X evaluation board equipped with 4 MiB of Burst Flash memory and 1 MiB of asynchronous SRAM, both connected to the processing core via the processor's *External Bus Unit*. In our experiments, the *Clock Generation Unit* was driven by an external crystal oscillator, producing a CPU clock at 150 MHz, and a system clock at 75 MHz.

Further details on the processor and on our hardware setup are provided in Appendix 2.

As indicated in Fig. 9, the FORTAS framework uses a feedback-driven anytime approach to calculate a sequence of increasingly fine-grained WCET estimates, as new traces are continuously added to the database of timed execution traces.

Our framework employs a hybrid method for generating input data that is based on genetic programming and source-code analysis (Bünte et al. 2011). The trace database is iteratively refined with traces that correspond to the identified execution scenarios by generating according FQL queries (Holzer et al. 2011) that are subsequently fed to the FSHELL test case generator (Holzer et al. 2008) to obtain respective input vectors. After executing the program under analysis with the new input vectors, the resulting timed execution traces are added to the database. We did not make use of any input vectors distributed as part of any benchmarks.

Thanks to the anytime approach, the framework can be used to quickly obtain a rough WCET estimate and more refined estimates later on, all within a single analysis run. Intermediate WCET estimates can be obtained repeatedly whenever desired, until the wanted precision has been obtained.

Early intermediate WCET estimates are useful as a quick approximation of the expected WCET, e.g., to obtain instant feedback on how a certain modification of the program might affect its WCET.

4.2 Benchmarks

We used benchmarks from four different benchmark suites to evaluate our approach:

Industry Study (IS): A benchmark suite derived from the code of an engine controller, provided by an industrial partner.

[1] Lauterbach GmbH, Höhenkirchen-Siegertsbrunn, Germany. PowerTrace

Mälardalen WCET Benchmark Suite (MD): A collection of benchmark programs from different research groups and tool vendors. We selected *bs*, an implementation of binary search over an array of 15 integer elements, and *bsort100*, an implementation of bubble sort over an array of 100 integer elements. For the latter benchmark we reduced the input vector to 10 integer elements.

PapaBench (PB): A benchmark suite originating from UAV software developed within the Paparazzi project (Paparazzi 2012). We chose subproblems *A1*, *A2*, *F1*, and *F2* from the version used in the WCET Tool Challenge 2011 (von Hanxleden et al. 2011).

Java Optimized Processor Benchmark Suite (JOP): A collection of programs that are used for evaluating the Java Optimized Processor (JOP) (Schoeberl 2009). We used the central control function of a C port of the *lift control* benchmark.

Due to technical limitations in some tools on which the FORTAS framework depends—like the TriCore compiler tool chain—some source code transformations must be performed on the benchmarks before analysis. For example, types that are defined by *typedef* must be expanded, *for* loops must be transformed to equivalent *while* loops, and the code must be formatted canonically, e.g., individual statements must occur in separate lines. Although these transformations are rather trivial, there is currently no tool to perform them automatically, which is the reason for our limited range of benchmarks.

4.3 Experiments and results

We used two different memory setups: For the *internal memory setup* we placed the executable code in the PMI's scratchpad RAM (SPRAM), and the program data in the DMI's local data RAM (LDRAM). For the *external memory setup* we placed the executable program code and the program data in the external SRAM, and enabled the ICACHE.

These two setups represent extreme cases for temporal predictability: Memory accesses have a constant penalty of 1 cycle for the internal memory setup, whereas the external memory setup introduces a high access time jitter. Given the hardware description in Appendix 2, sources of jitter can be found in: possible cache misses, if instruction caching is used; mixed single/block transfers over the PLMB/DLMB; possible occupation of the PLMB/DLMB by another bus master, like the PMU/DMU or the LMI; occupation of the external memory-bus by another bus master; jitter in DRAM accesses.

The two extreme cases for temporal predictability are therefore, on the one hand, the use of the separate internal PMI and DMI memories, and, on the other hand, the shared use of external memory for both, program and data, with instruction caching.

For the internal memory setup, we also analyzed the benchmarks using the industrial-strength static WCET analysis tool AIT (Thesing et al. 2003). It was not possible to obtain comparative data for the external memory setup, because AIT does not support such a more complex setup.

We performed the analyses on an Intel Core2 Quad Q9450 CPU running at 2.66 GHz with 8 GiB of DRAM. For each benchmark, we generated at least 100,000 timed

Table 1 Comparison of WCET estimates for internal memory setup

Benchmark	MOET (μs)	FORTAS		AIT	Sens./ std. (%)	Contexts	Nodes
		Sensitive (μs)	Standard (μs)				
engine_control_cs1-Ak.	11.99	21.13	21.37	19.91	99	937	398
lift_control-ctrl_loop	7.39	10.45	11.00	9.91	95	240	119
binary_search-b._s.	1.68	2.23	2.24	2.10	100	15	14
bsort10-BubbleSort	29.34	40.39	42.44	37.60	95	28	15
a1-course_pid_run	0.91	1.35	1.41	1.89	96	26	17
a1-course_run	1.28	1.83	1.89	2.33	97	12	10
a2-atan2	0.85	1.11	1.12	1.67	99	25	15
a2-compute_dist2_t._h.	0.41	0.56	0.56	0.97	100	6	6
a2-nav_home	8.17	11.48	12.13	16.56	95	27	21
a2-navigation_update	8.23	11.78	12.43	16.75	95	4	5
a2-sin	1.77	2.69	2.71	3.88	99	38	29
f1-check_mega128_v._t.	2.81	3.52	3.63	4.18	97	16	11
f1-servo_set	2.21	2.69	2.76	3.57	97	81	43
f2-vector_10	1.07	1.30	1.32	1.14	98	22	18

traces. Table 1 summarizes our analysis results for the internal memory setup. For each benchmark we list:

1. the observed end-to-end MOET;
2. the WCET estimate via FORTAS's context-sensitive IPET;
3. the WCET estimate via FORTAS's standard IPET;
4. the WCET bound via AIT's static timing analysis;
5. the quotient between the two FORTAS WCET estimates;
6. the number of contexts produced by context-sensitive IPET for the final estimate;
7. the number of CFG nodes.

The latter two numbers are indicative of the quality of each benchmark: the number of contexts indicates how many suitable execution scenarios occur. The number of CFG nodes provides an estimate of the size of the analysis problem.

The observed end-to-end MOET is the maximal observed execution time for the entire program. It is our best lower bound for the actual WCET. Indeed, both estimates calculated by the FORTAS framework, as well the upper WCET bound calculated by AIT are consistently higher.

Comparing the estimates of the FORTAS framework with the bounds of AIT, there is no consistent ranking. Assuming that AIT produces safe upper bounds, we can attribute any higher estimates of the FORTAS framework to higher pessimism, and these estimates are also upper bounds. On possible reason for the lower pessimism of AIT could be tighter loop iteration constraints.

Those cases where AIT produces larger numbers than the FORTAS framework could indicate that the latter are less pessimistic, but without knowledge of the actual

Table 2 Comparison of WCET estimates for external memory setup.

Benchmark	MOET (μs)	FORTAS		Sens./std. (%)	Contexts	CFG Nodes
		Sensitive (μs)	Standard (μs)			
engine_control_cs1-Ak.	133.45	161.09	166.25	97	813	398
lift_control-ctrl_loop	64.95	73.59	87.29	84	319	119
binary_search-b._s.	17.76	18.32	22.61	81	17	14
bsort10-BubbleSort	287.89	316.23	392.93	80	32	15
a1-course_pid_run	8.78	9.71	10.83	90	25	17
a1-course_run	11.45	13.33	14.45	92	12	10
a2-atan2	8.29	8.58	8.66	99	24	15
a2-compute_dist2_t._h.	4.02	4.16	4.16	100	6	6
a2-fly_to_xy	9.27	11.87	11.95	99	5	5
a2-nav_home	62.21	76.18	85.81	89	34	21
a2-navigation_update	63.67	77.79	87.41	89	4	5
a2-sin	15.30	17.74	19.61	90	46	29
f1-check_mega128_v._t.	22.00	24.07	26.61	90	18	11
f1-servo_set	17.69	18.74	21.28	88	64	43
f2-vector_10	8.08	8.34	8.59	97	21	18

WCET it is impossible to decide to which degree the FORTAS framework is affected by underestimation. The magnitude of the deviation between the estimates of both tools does not provide an indication about the precision of the analyses performed by the FORTAS framework and aiT.

Comparing the WCET estimates produced by context-sensitive IPET to those produced by standard IPET, we see that the former results are closer to the MOET than the latter. This reduction can be a rough indicator of the achieved reduction in pessimism, illustrating the effect of context-sensitive IPET.

Table 2 summarizes the analysis results for the external memory setup. As mentioned before, aiT does not support this configuration, so we can provide results for the FORTAS framework only.

It can be seen that all MOETs and WCET estimates are considerably higher than for the internal memory setup. This is unsurprising, as dynamic RAM has typically a much higher access latency than static RAM. Moreover, the data path to external memory is much longer than the data path to the SPRAM/LDRAM.

The context-sensitive WCET estimates are close to the respective MOETs. This indicates that context-free IPET can work well in scenarios with a high execution time jitter for individual code constituents. Moreover, the distance between the WCET estimates of context-sensitive IPET and standard IPET is much larger for the external memory setup than for the internal memory setup—in relative and absolute measures. This meets our expectation, because the external memory datapath contains many sources of temporal jitter, whereas the internal memory datapath is virtually free of jitter, leaving less room for reducing pessimism.

Table 3 Mean runtime of `lp_solve` for internal memory setup

Benchmark	Sensitive (ms)	Standard (ms)	Sens./std (%)
engine_control_cs1-Ak.	682.1	539.6	126
lift_control-ctrl_loop	76.4	70.3	109
binary_search-b._s.	1.7	1.2	142
bsort10-BubbleSort	1.8	3.0	60
a1-course_pid_run	1.6	2.0	80
a1-course_run	0.8	0.9	89
a2-atan2	2.0	0.8	250
a2-compute_dist2_t._h.	0.3	0.3	100
a2-fly_to_xy	0.4	0.4	100
a2-nav_home	2.6	1.9	137
a2-navigation_update	0.5	0.6	83
a2-sin	3.3	3.0	110
f1-check_mega128_v._t.	1.3	1.0	130
f1-servo_set	9.5	7.9	120
f2-vector_10	1.8	1.8	100

Concerning the computational cost of context-sensitive IPET, we have to consider two things: The complexity of building the context-sensitive IPET model from the trace database and the cost of solving that model.

The complexity of building the context-sensitive IPET model depends on the concrete instantiation of context-sensitive IPET. Considering the instantiation that we have presented in this paper, Algorithm 3.1 has linear time complexity in the number $|E|$ of CFG edges and linear time complexity in the maximal number of MOET classes.

The cost of solving the resulting context-sensitive IPET models depends on the used LP solver. For our experiments we used `lp_solve` with standard settings. Tables 3 and 4 present a comparison of the mean solving times of the context-sensitive IPET problems and their standard IPET counterparts. It can be seen that the solving times for the two cases are quite similar. In one extreme case the context-sensitive problems took about twice as long to solve as the corresponding standard IPET problems, whereas in the other extreme case the context-sensitive problems took only half as long to solve as the corresponding standard IPET problems. Overall, we observe no indication that the context-sensitive IPET problems are significantly harder or easier compared to standard IPET problems.

Summarizing our experimental data, we observed that context-free IPET works well, especially in scenarios with a high execution-time jitter for individual code constituents.

5 Conclusion

We have presented context-sensitive IPET, an ILP-based approach for calculating a WCET estimate from a given database of timed execution traces.

Table 4 Mean runtime of `lp_solve` for external memory setup

Benchmark	Sensitive (ms)	Standard (ms)	Sens./std (%)
engine_control_cs1-Ak.	626.7	510.3	123
lift_control-ctrl_loop	86.5	73.7	117
binary_search-b._s.	0.8	1.6	50
bsort10-BubbleSort	2.4	2.2	109
a1-course_pid_run	2.2	1.0	220
a1-course_run	0.9	0.9	100
a2-atan2	1.6	2.1	76
a2-compute_dist2_t._h.	0.8	0.4	200
a2-fly_to_xy	0.5	0.4	125
a2-nav_home	2.4	2.4	100
a2-navigation_update	0.3	0.6	50
a2-sin	3.9	3.1	125
f1-check_mega128_v._t.	1.5	1.5	100
f1-servo_set	8.4	6.9	122
f2-vector_10	1.9	1.8	106

Our method is based on standard IPET, a widely used method for calculating an upper WCET bound for a piece of code from upper WCET bounds of its code constituents. It can thus reuse flow facts from other analysis tools and produces ILP problems that can be solved by off-the-shelf solvers. Unlike previous work, our method specifically aims at reducing overestimation, by means of an automatic classification of code executions into scenarios with differing worst-case behaviour.

Context-sensitive IPET is a generic method. To obtain a concrete method, context-sensitive IPET must be instantiated with a concrete notion of an *execution scenario*. We have presented such an instantiation, which is based on the notion of a *context*, which captures control flows within a program. We have also presented an algorithm for producing such contexts based on measured execution times.

We have implemented our method as a component of the FORTAS framework—a prototypical implementation of a portable, high-precision MBTA toolchain—and presented an experimental evaluation of our method. The results of our evaluation indicate that context-sensitive IPET can yield closer WCET estimates than standard IPET.

Our results also indicate that WCET estimates obtained by an MBTA toolchain that harnesses context-sensitive IPET can be comparable to those obtained by toolchains that are based on static analysis. However, it is important to bear in mind that MBTA and static timing analysis have different use-cases and must therefore be understood as complementary approaches.

Acknowledgments This research has been supported by the Austrian Science Fund (Fonds zur Förderung der wissenschaftlichen Forschung) within the research project "Formal Timing Analysis Suite of Real-Time Systems" (FORTAS-RT) under contract P19230-N13, by the EU FP-7 project "Asynchronous and Dynamic

Virtualisation through performance ANalysis to support Concurrency Engineering" (ADVANCE) under contract no. 248828, by ARTEMIS-JU within the FP7 research project "ConstRaint and Application driven Framework for Tailoring Embedded Real-time Systems" (CRAFTERS) under contract no. 295371, and by the EU COST Action IC1202 "Timing Analysis On Code-Level" (TACLe).

Open Access This article is distributed under the terms of the Creative Commons Attribution 4.0 International License (http://creativecommons.org/licenses/by/4.0/), which permits unrestricted use, distribution, and reproduction in any medium, provided you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license, and indicate if changes were made.

Appendix 1: Proofs

Proof of Theorem 1 Let $f_{v,1}, \ldots, f_{v,n(v)}$ be a solution of the context-sensitive IPET problem. The context-sensitive IPET problem contains more constraints, so $f_{v,1}, \ldots, f_{v,n(v)}$ is also a solution of the respective standard IPET problem. Because $f_v = \sum_{i=1}^{n(v)} f_{v,i}$ and $\widetilde{wcet}_{v,i} \leq \widetilde{wcet}_v$ for $1 \leq i \leq n(v)$, we have

$$\sum_{v \in V} \sum_{i=1}^{n(v)} \widetilde{wcet}_{v,i} \cdot f_{v,i} \leq \sum_{v \in V} \sum_{i=1}^{n(v)} \widetilde{wcet}_v \cdot f_{v,i} = \sum_{v \in V} \widetilde{wcet}_v \cdot f_v.$$

□

Proof of Theorem 2 Let $\mathcal{S} = [\![A, B]\!]$ be a clip. Choose any non-divergent paths $\pi \neq \sigma$ with

$$\pi = u_1 \ldots u_n \in paths(\mathcal{S}), \text{ and } \sigma = w_1 \ldots w_m \in paths(\mathcal{S}).$$

Case 1: There exist paths α, β, γ with $\alpha \circ \beta = \pi$, $\beta \circ \gamma = \sigma$ and $|\beta| \geq 2$. Because $\pi \neq \sigma$, we have $|\alpha| \geq 1$ or $|\gamma| \geq 1$ (or both). If $|\alpha| \geq 1$, then π contains some entry edge $(u_i, u_{i+1}) \in A$, where $2 \leq i \leq n-2$. This contradicts the assumption that $\pi \in paths(\mathcal{S})$. If $|\gamma| \geq 1$, then σ contains some exit edge $(w_i, w_{i+1}) \in B$, where $2 \leq i \leq m-2$. This contradicts the assumption that $\sigma \in paths(\mathcal{S})$.

Case 2: There exist paths α, β, γ with $\alpha \circ \beta = \sigma$, $\beta \circ \gamma = \pi$ and $|\beta| \geq 2$. This case is symmetric to Case 1.

Case 3: Path σ is a subpath of π. Since $\sigma \neq \pi$, path σ must be a proper subpath of π, hence π contains some edge $(u_i, u_{i+1}) \in A \cup B$, where $2 \leq i \leq n-2$. This contradicts the assumption that $\pi \in paths(\mathcal{S})$.

Case 4: Path π is a subpath of σ. This case is symmetric to Case 3. □

Proof of Theorem 3 We first show that $\mathcal{S}$ is a context, and then show the coverage property:

1. Consider any path $\pi \in paths(\mathcal{S})$. Any edge (v, b) with source node v is an exit edge of clip $\mathcal{S}$. Therefore, π can contain at most one more node after the first occurrence of v. Hence, there is at most one inner occurrence of v in π.

2. We show that $\mathcal{S}$ covers v. Consider any paths ρ, σ with $\rho \circ v \circ \sigma \in \mathcal{U}$. Note that path ρ starts with the start node v_{start} and that path σ ends with the end node v_{end}, which implies that paths ρ and σ are not empty.
   Case 1: Path ρ contains node v. Then choose paths τ_1, τ_2 with $\tau_1 \circ v \circ \tau_2 = \rho$, such that τ_2 does not contain node v. Path $v \circ \tau_2 \circ v$ starts with an edge in $B \cap R$, path $\tau_2 \circ v$ contains no edge in $Q \cup B$, and path $v \circ \sigma$ starts with an edge $(v, b) \in B$. Hence, path $v \circ \tau_2 \circ v \circ b$ is in $paths(\mathcal{S})$, and there are subpaths ρ_1, σ_2 with $\rho_1 \circ v \circ \tau_2 = \rho$ and $b \circ \sigma_2 = \sigma$.
   Case 2: Path ρ does not contain node v. Then path $\rho \circ v$ starts with an edge in $Q \cap R$, path $\rho \circ v$ contains no edge in B, except for its first edge $(v, b) \in B$, and path $v \circ \sigma$ starts with an edge $(v, b) \in B$. Hence, path $\sigma \circ v \circ b$ is in $paths(\mathcal{S})$, and there are subpaths ρ_1, σ_2 with $\rho_1 \circ \rho = \rho$ and $b \circ \sigma_2 = \sigma$. □

Proof of Theorem 4 1. We show that $\mathcal{C}_1$ and $\mathcal{C}_2$ are contexts of v. For every edge $(x_1, x_2) \in X$, there exists an edge $(a, u) \in A$ with a path from node u to node x_1 that contains only edges in $E \setminus (A \cup B)$ and an edge $(w, b) \in B$ with a path from node x_2 to node w that contains only edges in $E \setminus (A \cup B)$. Therefore, every path in $paths(\llbracket A, X \rrbracket)$ is a subpath of some path in $paths(\llbracket A, B \rrbracket)$, hence the clip $\llbracket A, X \rrbracket$ contains at most one occurrence of v along each path, i.e., it is a context of v.
By a similar argument, every path in $paths(\llbracket X, B \rrbracket)$ is a subpath of some path in $paths(\llbracket A, B \rrbracket)$, hence $\llbracket X, B \rrbracket$ is a context of v.
Lastly, every path in $paths(\llbracket X, X \cap Z \rrbracket)$ is a subpath of some path in $paths(\llbracket A, B \rrbracket)$, hence $\llbracket X, X \rrbracket$ is a context of v.
Hence $\llbracket A, (B \cup X) \cap Y \rrbracket)$ and $\llbracket X, (B \cup X) \cap Z \rrbracket)$ are contexts of v, because

$$\begin{aligned} paths\left(\llbracket A, (B \cup X) \cap Y \rrbracket\right) &\subseteq paths\left(\llbracket A, B \cup X \rrbracket\right) \subseteq paths\left(\llbracket A, B \rrbracket\right) \cup paths\left(\llbracket A, X \rrbracket\right); \\ paths\left(\llbracket X, (B \cup X) \cap Z \rrbracket\right) &\subseteq paths\left(\llbracket X, (B \cap Z) \cup (X \cap Z) \rrbracket\right) \subseteq \\ &\subseteq paths\left(\llbracket X, B \rrbracket\right) \cup paths\left(\llbracket X, X \cap Z \rrbracket\right). \end{aligned}$$

2. We show that $paths(\mathcal{C}_1) \cup paths(\mathcal{C}_2)$ covers v. Choose any paths ρ, σ, with $\rho \circ v \circ \sigma \in \mathcal{U}$. By our initial assumption, $paths(\mathcal{C})$ covers v, i.e., there are subpaths ρ_1, σ_2 and non-empty subpaths ρ_2, σ_1, with $\rho_1 \circ \rho_2 = \rho$ and $\sigma_1 \circ \sigma_2 = \sigma$, such that $\rho_2 \circ v \circ \sigma_1 \in paths(\mathcal{C})$. Moreover, there is some entry edge $(a, u) \in A$, some exit edge $(w, b) \in B$, and paths α, β with $a \circ u \circ \alpha = \rho_2 \circ v$ and $\beta \circ w \circ b = v \circ \sigma_1$. We show by construction, that there are always subpaths ρ'_1, σ'_2 and non-empty subpaths ρ'_2, σ'_1, with $\rho'_1 \circ \rho'_2 = \rho$ and $\sigma'_1 \circ \sigma'_2 = \sigma$, such that $\rho'_2 \circ v \circ \sigma'_1 \in paths(\mathcal{C}_1) \cup paths(\mathcal{C}_2)$.
Case 1: Path $u \circ \alpha \circ v \circ \beta \circ w$ does not contain any edge in X. Then $u \circ \alpha \circ v \circ \beta \circ w$ contains only edges in $E \setminus (A \cup B \cup X)$, therefore edge (w, b) is in $B \cap Y$. Hence, choose $\rho'_2 = \rho_2$ and $\sigma'_1 = \sigma_1$. Path $\rho'_2 \circ v \circ \sigma'_1$ is in $paths(\mathcal{C}_1)$.
Case 2: Path $u \circ \alpha \circ v$ contains some edge in X, but path $v \circ \beta \circ w$ does not. Then there is some edge $(x_1, x_2) \in X$ and paths α_1, α_2, with $\alpha_1 \circ x_1 \circ x_2 \circ \alpha_2 = u \circ \alpha \circ v$, such that path $x_2 \circ \alpha_2$ contains only edges in $E \setminus X$. Path $x_2 \circ \alpha_2 \circ \beta \circ w$ then contains only edges in $E \setminus (A \cup B \cup X)$, therefore edge (w, b) is in $B \cap Z$. Hence, choose $\rho'_2 \circ v = x_1 \circ x_2 \circ \alpha_2$ and $\sigma'_1 = \beta \circ w \circ b$. Path $\rho'_2 \circ v \circ \sigma'_1$ is in $paths(\mathcal{C}_2)$.

Case 3: Path $u \circ \alpha \circ v$ does not contain any edge in X, but path $v \circ \beta \circ w$ does. Then there is some edge $(y_1, y_2) \in X$ and paths β_1, β_2, with $\beta_1 \circ y_1 \circ y_2 \circ \beta_2 = v \circ \beta \circ w$, such that path $\beta_1 \circ y_1$ contains only edges in $E \setminus X$. Path $u \circ \alpha \circ \beta_1 \circ y_1$ then contains only edges in $E \setminus (A \cup B \cup X)$, therefore edge (y_1, y_2) is in $X \cap Y$. Hence, choose $\rho_2' = a \circ u \circ \alpha$ and $\sigma_1 \circ v = \beta_1 \circ y_1 \circ y_2$. Path $\rho_2' \circ v \circ \sigma_1'$ is in $paths(\mathcal{C}_1)$.
Case 4: Path $u \circ \alpha \circ v$ contains some edge in X, and so does path $v \circ \beta \circ w$. Then there is some edge $(x_1, x_2) \in X$ and paths α_1, α_2, with $\alpha_1 \circ x_1 \circ x_2 \circ \alpha_2 = u \circ \alpha \circ v$, such that path $x_2 \circ \alpha_2$ contains only edges in $E \setminus X$. Also, there is some edge $(y_1, y_2) \in X$ and paths β_1, β_2, with $\beta_1 \circ y_1 \circ y_2 \circ \beta_2 = v \circ \beta \circ w$, such that path $\beta_1 \circ y_1$ contains only edges in $E \setminus X$. Now choose α_3 with $\alpha_3 \circ v = x_2 \circ \alpha_2$, and choose β_3 with $v \circ \beta_3 = \beta_2 \circ y_1$. Path $\alpha_3 \circ v \circ \beta_3$ then contains only edges in $E \setminus (A \cup B \cup X)$, therefore edge (y_1, y_2) is in $X \cap Z$. Choose $\rho_2' \circ v = x_1 \circ x_2 \circ \alpha_2$ and $v \circ \sigma_1' = \beta_2 \circ y_1 \circ y_2$. Path $\rho_2' \circ v \circ \sigma_1'$ is in $paths(\mathcal{C}_2)$.

3. We show that $paths(\mathcal{C}_1)$ and $paths(\mathcal{C}_2)$ are divergent.
   Consider the paths

$$\pi = u_1 \dots u_n \in paths(\mathcal{C}_1), \text{ and } \sigma = w_1 \dots w_m \in paths(\mathcal{C}_2).$$

   There are two cases how π and σ may overlap:
   Case 1: π contains the first edge $(w_1, w_2) \in X$ of σ. Since $X \subseteq E \setminus (A \cup B)$ and $(u_1, u_2) \in A$, we have $(u_i, u_{i+1}) \in X$, for some i with $2 \leq i \leq (n-1)$. However, by the definition of Y, that means $(u_{n-1}, u_n) \notin Y$, unless $i = n-1$. Hence, any occurrence of the first edge (w_1, w_2) of π must be on last edge (u_{n-1}, u_n) of σ. Since π is a path of a context, it contains at least two edges—an entry edge and an exit edge. Therefore, π cannot be a subpath of σ, and there are no paths α, β, γ with $\alpha \circ \beta = \pi$, $\beta \circ \gamma = \sigma$, and $|\beta| \geq 2$.
   Case 2: σ contains the first edge $(u_1, u_2) \in A$ of π. Since $X \subseteq E \setminus (A \cup B)$ and $(w_1, w_2) \in X$, we have $(w_i, w_{i+1}) \in A$, for some i with $2 \leq i \leq (m-1)$. However, by the definition of Z, that means $(w_{m-1}, u_m) \notin Z$, unless $i = m-1$. Hence, any occurrence of the first edge (u_1, u_2) of σ must be on last edge (w_{m-1}, w_m) of π. Since σ is a path of a context, it contains at least two edges—an entry edge and an exit edge. Therefore, σ cannot be a subpath of π, and there are no paths α, β, γ with $\alpha \circ \beta = \sigma$, $\beta \circ \gamma = \pi$, and $|\beta| \geq 2$. □

Proof of Theorem 5 First, we show that the clip $\mathcal{C}_D$ is a context of node v, for any $D \in \mathcal{D}$. Next, we show that $W \cup \bigcup_{D \in \mathcal{D}} paths(\mathcal{C}_D)$ covers node v, if $W \cup paths(\mathcal{C})$ covers v, for any set of paths W. Lastly, we show that $paths(\mathcal{C}_{D_1})$ and $paths(\mathcal{C}_{D_2})$ are divergent, for any sets $D_1, D_2 \in \mathcal{D}$ with $D_1 \neq D_2$.

1. We show that clip $\mathcal{C}_D$ is a context of node v, for any $D \in \mathcal{D}$. We have $D \subseteq A$, because $\mathcal{D}$ is a partition of A, and $B \cap Z_D \subseteq B$. Therefore, $paths(\mathcal{C}_D)$ is a subset of $paths(\mathcal{C})$.
   Clip $\mathcal{C}$ is a context of node v, i.e., all paths in $paths(\mathcal{C})$ contain at most one occurrence of v. Since $paths(\mathcal{C}_D)$ is a subset of $paths(\mathcal{C})$, its paths also contain at most one occurrence of v. It follows that $\mathcal{C}_D$ is a context of v.

2. We show that $W \cup \bigcup_{D \in \mathcal{D}} paths(\mathcal{C}_D)$ covers node v, if $W \cup paths(\mathcal{C})$ covers v, for any set of paths W. Choose any paths ρ, σ, with $\rho \circ v \circ \sigma \in \mathcal{U}$. By assumption, $W \cup paths(\mathcal{C})$ covers v, i.e., there are subpaths ρ_1, σ_2 and non-empty subpaths ρ_2, σ_1, with $\rho_1 \circ \rho_2 = \rho$ and $\sigma_1 \circ \sigma_2 = \sigma$, such that $\rho_2 \circ v \circ \sigma_1 \in W \cup paths(\mathcal{C})$. If $\rho_2 \circ v \circ \sigma_1 \in W$, then $\rho_2 \circ v \circ \sigma_1 \in W \cup \bigcup_{D \in \mathcal{D}} paths(\mathcal{C}_D)$, and we are done. Otherwise, path ρ_2 starts with some entry edge $(a, x) \in A$ of context $\mathcal{C}$, and since $\mathcal{D}$ is a partition of A, there is some set $D \in \mathcal{D}$, such that $(a, x) \in D$. Furthermore, path σ_1 ends with some exit edge $(y, b) \in B$ of context $\mathcal{C}$, and there are no further occurrence of any edge from $A \cup B$ in $\rho_2 \circ v \circ \sigma_1$, therefore (y, b) is in $B \cap Z_D$, hence $\rho_2 \circ v \circ \sigma_1 \in paths(\mathcal{C}_D) \subseteq W \cup \bigcup_{D \in \mathcal{D}} paths(\mathcal{C}_D)$.
3. We show that $paths(\mathcal{C}_{D_1})$ and $paths(\mathcal{C}_{D_2})$ are divergent, for any sets $D_1, D_2 \in \mathcal{D}$ with $D_1 \neq D_2$. Choose any path $\pi \in paths(\mathcal{C}_{D_1})$, and any path $\sigma \in paths(\mathcal{C}_{D_2})$. Since $D_1 \cap D_2 = \emptyset$, paths π and σ must have a different entry edge.
Moreover, we have $D_1 \subseteq A$ and $\sigma \in paths(\mathcal{C})$, therefore the entry edge of π can only occur on the last edge of σ, hence π cannot be a subpath of σ, and there are no paths α, β, γ with $\alpha \circ \beta = \pi$, $\beta \circ \gamma = \sigma$, and $|\beta| \geq 2$.
Likewise, we have $D_2 \subseteq A$ and $\pi \in paths(\mathcal{C})$, therefore the entry edge of σ can only occur on the last edge of π, hence σ cannot be a subpath of π, and there are no paths α, β, γ with $\alpha \circ \beta = \sigma$, $\beta \circ \gamma = \pi$, and $|\beta| \geq 2$.
We conclude that $\mathcal{C}_{D_1}$ and $\mathcal{C}_{D_1}$ are divergent. □

Proof of Theorem 6 We have $occ(v, \pi) \cap occ(v, \sigma) = \emptyset$ for any paths π, σ with $\pi \neq \sigma$. From this the theorem follows easily. □

Proof of Theorem 7 We have $occ(e, \pi) \cap occ(e, \sigma) = \emptyset$ for any paths π, σ with $\pi \neq \sigma$. From this the theorem follows easily. □

Proof of Theorem 8 We have $occ(v, \pi, \mathcal{S}) \cap occ(v, \sigma, \mathcal{S}) = \emptyset$ for any paths π, σ with $\pi \neq \sigma$. From this the theorem follows easily. □

Proof of Theorem 9 We separately show the $\leq$ and the $\geq$ part of the equality:

1. Since the contexts $\mathcal{C}_i$ are pairwise divergent, we have

$$\sum_{1 \leq i \leq n} |occ(v, \pi, \mathcal{C}_i)| = \Big| \bigcup_{1 \leq i \leq n} occ(v, \pi, \mathcal{C}_i) \Big|,$$
$$\text{for all } \pi \in \mathcal{U}, \text{ for all } \pi \in \mathcal{U}.$$

Now, consider any path $\pi \in \mathcal{U}$ and any paths ρ, σ, with $\rho \circ v \circ \sigma = \pi$. Since $\bigcup_{1 \leq i \leq n} paths(\mathcal{C}_i)$ covers node v, there is some index i with $1 \leq i \leq n$, subpaths ρ_1, σ_2, and non-empty subpaths ρ_2, σ_1 with $\rho_1 \circ \rho_2 = \rho$ and $\sigma_1 \circ \sigma_2 = \sigma$, such that $\rho_2 \circ v \circ \sigma_1 \in paths(\mathcal{C}_i)$.
Also, consider any paths ρ', σ', with $\rho' \circ v \circ \sigma' = \pi$ and $(\rho, \sigma) \neq (\rho', \sigma')$. Again, since $\bigcup_{1 \leq i \leq n} paths(\mathcal{C}_i)$ covers node v, there is some index i' with $1 \leq i' \leq n$, subpaths ρ'_1, σ'_2, and non-empty subpaths ρ'_2, σ'_1 with $\rho'_1 \circ \rho'_2 = \rho'$ and $\sigma'_1 \circ \sigma'_2 = \sigma'$, such that $\rho'_2 \circ v \circ \sigma'_1 \in paths(\mathcal{C}'_i)$. Since $(\rho, \sigma) \neq (\rho', \sigma')$, we also have $(\rho'_1, \rho'_2, \sigma'_1, \sigma'_2) \neq (\rho_1, \rho_2, \sigma_1, \sigma_2)$.

We see that, for any two different elements (ρ, σ) and (ρ', σ') in $occ(v, \pi)$, we get different elements $(\rho_1, \rho_2, \sigma_1, \sigma_2)$ and $(\rho_1', \rho_2', \sigma_1', \sigma_2')$ in $\bigcup_{1\le i\le n} occ(v, \pi, \mathcal{C}_i)$, hence

$$|occ(v,\pi)| \le |\bigcup_{1\le i\le n} occ(v,\pi,\mathcal{C}_i)| = \sum_{1\le i\le n} |occ(v,\pi,\mathcal{C}_i)|, \text{ for all } \pi \in \mathcal{U}.$$

2. Since the contexts $\mathcal{C}_i$ are pairwise divergent, we have

$$\sum_{1\le i\le n} |occ(v,\pi,\mathcal{C}_i)| = \left|\bigcup_{1\le i\le n} occ(v,\pi,\mathcal{C}_i)\right|,$$
$$\text{for all } \pi \in \mathcal{U}, \text{ for all } \pi \in \mathcal{U}.$$

Now, consider any path $\pi \in \mathcal{U}$. Choose any paths ρ_1, σ_2 and any paths $\rho_2 \ne \epsilon$, $\sigma_1 \ne \epsilon$, such that $\rho_1 \circ \rho_2 = \rho$, such that $\sigma_1 \circ \sigma_2 = \sigma$, and such that $\rho_2 \circ v \circ \sigma_1 \in paths(\mathcal{C}_i)$ with $1 \le i \le n$. Moreover, choose paths ρ_1', σ_2' and any paths $\rho_2' \ne \epsilon$, $\sigma_1' \ne \epsilon$, such that $\rho_1' \circ \rho_2' = \rho'$, such that $\sigma_1' \circ \sigma_2' = \sigma'$, and such that $\rho_2' \circ v \circ \sigma_1' \in paths(\mathcal{C}_i')$ with $1 \le i' \le n$, and such that $(\rho_2, \sigma_1) \ne (\rho_2', \sigma_1')$. There are two cases:

Case 1: $i = j$, i.e., paths $\rho_2 \circ v \circ \sigma_1$ and $\rho_2' \circ v \circ \sigma_1'$ are in the same context. Then they are divergent, by Theorem 2.

Case 2: $i \ne j$, i.e., paths $\rho_2 \circ v \circ \sigma_1$ and $\rho_2' \circ v \circ \sigma_1'$ are in different contexts. Then they are divergent, by the original assumption that all contexts $\mathcal{C}_i$ are pairwise divergent.

In both cases, paths $\rho_2 \circ v \circ \sigma_1$ and $\rho_2' \circ v \circ \sigma_1'$ are divergent. Therefore, we have $(\sigma_1 \circ \sigma_2, \rho_1 \circ \rho_2) \ne (\sigma_1' \circ \sigma_2', \rho_1' \circ \rho_2')$. We see that, for any two different elements (ρ_2, σ_1) and (ρ_2', σ_1') in $\bigcup_{1\le i\le n} occ(v, \pi, \mathcal{C}_i)$, we get different elements $(\rho_1 \circ \rho_2, \sigma_1 \circ \sigma_2)$ and $(\rho_1' \circ \rho_2', \sigma_1' \circ \sigma_2')$ in $occ(v, \pi)$, hence

$$|occ(v,\pi)| \ge \left|\bigcup_{1\le i\le n} occ(v,\pi,\mathcal{C}_i)\right| = \sum_{1\le i\le n} |occ(v,\pi,\mathcal{C}_i)|, \text{ for all } \pi \in \mathcal{U}.$$

□

Proof of Theorem 10 We give a proof for the first inequality. The proof for the second inequality is symmetric.

By the definition of *occ* and by the observation that

$$occ(w,\pi) \cap occ(w',\pi) = \emptyset \text{ for all } w, w' \in V, w \ne w', \pi \in \mathcal{U},$$

we have

$$\sum_{a\in A} |occ(a,\pi)| = |\{(\rho,\sigma) \mid \rho \circ a \circ \sigma = \pi, a \in A\}|;$$
$$\sum_{x\in X} |occ(x,\pi)| = |\{(\rho',\sigma') \mid \rho' \circ x \circ \sigma' = \pi, x \in X\}|.$$

If we consider the definition of X, we see that ρ' contains an occurrence of some edge $a \in A$, such that there is no subsequent occurrence of any edge $y \in A \cup B$ in ρ', i.e., we have

$$\begin{aligned}&\left\{(\rho',\sigma') \mid \rho' \circ x \circ \sigma' = \pi, x \in X\right\}\\ &\quad = \left\{(\rho',\sigma') \mid \rho' \circ x \circ \sigma' = \pi, \rho \circ a \circ \tau = \rho', a \in A,\right.\\ &\qquad x \in X, \nexists(y \in A \cup B, \tau_1, \tau_2) : \tau_1 \circ y \circ \tau_2 = \tau\} .\end{aligned}$$

We are interested in the number of elements in the latter set, not in the elements themselves. This allows us to make use of the following equality:

$$\begin{aligned}&\Big|\{(\rho',\sigma') \;\Big|\; \rho' \circ x \circ \sigma' = \pi, \rho \circ a \circ \tau = \rho', a \in A,\\ &\quad x \in X, \nexists(y \in A \cup B, \tau_1, \tau_2) : \tau_1 \circ y \circ \tau_2 = \tau\}|\\ &\qquad = |\{(\rho,\sigma) \mid \rho \circ a \circ \sigma = \pi, \tau \circ x \circ \sigma' = \sigma, a \in A,\\ &\qquad\quad x \in X, \nexists(y \in A \cup B, \tau_1, \tau_2) : \tau_1 \circ y \circ \tau_2 = \tau\}|.\end{aligned}$$

Since

$$\begin{aligned}&\{(\rho,\sigma) \mid \rho \circ a \circ \sigma = \pi, \tau \circ x \circ \sigma' = \sigma, a \in A,\\ &\quad x \in X, \nexists(y \in A \cup B, \tau_1, \tau_2) : \tau_1 \circ y \circ \tau_2 = \tau\} \subseteq \{(\rho,\sigma) \mid \rho \circ a \circ \sigma = \pi, a \in A\},\end{aligned}$$

we have

$$\begin{aligned}\sum_{a\in A} |occ(a,\pi)| - \sum_{x\in X} |occ(x,\pi)| = \Big|& \{(\rho,\sigma) \mid \rho \circ a \circ \sigma = \pi, a \in A\} \setminus\\ &\left\{(\rho,\sigma) \mid \rho \circ a \circ \sigma = \pi, \tau \circ x \circ \sigma' = \sigma, a \in A,\right.\\ &x \in X, \nexists(y \in A \cup B, \tau_1, \tau_2) : \tau_1 \circ y \circ \tau_2 = \tau\} \Big|\end{aligned}$$

Expansion of the set subtraction yields

$$\begin{aligned}\sum_{a\in A} |occ(a,\pi)| - \sum_{x\in X} |occ(x,\pi)| = \Big|&\{(\rho,\sigma) \mid \rho \circ a \circ \sigma = \pi, a \in A,\\ &\nexists(x \in X, y \in A \cup B, \tau_1, \tau_2, \sigma') :\\ &\tau_1 \circ y \circ \tau_2 \circ x \circ \sigma' = \sigma\}\Big|.\end{aligned}$$

Now, choose any element from $occ(v, \pi, [\![A, B]\!])$, i.e., choose any paths ρ_1, σ_2, and any paths $\rho_2 \neq \epsilon, \sigma_1 \neq \epsilon$, with $\rho_1 \circ \rho_2 \circ v \circ \sigma_1 \circ \sigma_2 = \pi$, and where $\rho_2 \circ v \circ \sigma_1 \in paths([\![A, B]\!])$. By the definition of a context, path ρ_2 must start with an entry edge $a \in A$, must end with an exit edge $b \in B$, and cannot contain any further occurrences of any edge in $A \cup B$. Moreover, path $\rho_2 \circ v \circ \sigma_1$ cannot contain any edge $x \in X$.

Therefore, $(\rho_1 \circ \rho_2, \sigma_1 \circ \sigma_2)$ is an element of

$$\{(\rho, \sigma) \mid \rho \circ a \circ \sigma = \pi, a \in A, \\ \nexists (x \in X, y \in A \cup B, \tau_1, \tau_2, \sigma') : \tau_1 \circ y \circ \tau_2 \circ x \circ \sigma' = \sigma\Big\}.$$

Next, choose another element from $occ(v, \pi, [\![A, B]\!])$, i.e., choose paths ρ_1', σ_2', and any paths $\rho_2' \neq \epsilon, \sigma_1' \neq \epsilon$, with $\rho_1' \circ \rho_2' \circ v \circ \sigma_1' \circ \sigma_2' = \pi$, and where $\rho_2' \circ v \circ \sigma_1' \in paths([\![A, B]\!])$, with $(\rho_1', \rho_2', \sigma_1', \sigma_2') \neq (\rho_1, \rho_2, \sigma_1, \sigma_2)$. By the definition of a context, path ρ_2' must start with an entry edge $a \in A$, must end with an exit edge $b \in B$, and cannot contain any further occurrences of any edge in $A \cup B$. Moreover, path $\rho_2' \circ v \circ \sigma_1'$ cannot contain any edge $x \in X$. Therefore, $(\rho_1' \circ \rho_2', \sigma_1' \circ \sigma_2')$ is an element of

$$\{(\rho, \sigma) \mid \rho \circ a \circ \sigma = \pi, a \in A, \\ \nexists (x \in X, y \in A \cup B, \tau_1, \tau_2, \sigma') : \tau_1 \circ y \circ \tau_2 \circ x \circ \sigma' = \sigma\Big\}.$$

By Theorem 2, the paths $\rho_2' \circ v \circ \sigma_1'$ and $\rho_2 \circ v \circ \sigma_1$ are divergent, and therefore we have $(\rho_1 \circ \rho_2, \sigma_1 \circ \sigma_2) \neq (\rho_1' \circ \rho_2', \sigma_1' \circ \sigma_2')$. We see that, for any two different elements $(\rho_1, \rho_2, \sigma_1, \sigma_2)$ and $(\rho_1', \rho_2', \sigma_1', \sigma_2')$ from $occ(v, \pi, \mathcal{C})$, we get different elements $(\rho_1 \circ \rho_2, \sigma_1 \circ \sigma_2)$ and $(\rho_1' \circ \rho_2', \sigma_1' \circ \sigma_2')$ in

$$\{(\rho, \sigma) \mid \rho \circ a \circ \sigma = \pi, a \in A, \\ \nexists (x \in X, y \in A \cup B, \tau_1, \tau_2, \sigma') : \tau_1 \circ y \circ \tau_2 \circ x \circ \sigma' = \sigma\Big\},$$

hence

$$|occ(v, \pi, \mathcal{C})| \leq \sum_{a \in A} |occ(a, \pi)| - \sum_{x \in X} |occ(x, \pi)|.$$

□

Proof of Theorem 11 It is easy to see that $\{moet_{v,\pi} \mid \overline{\pi} \in paths(\mathcal{S}), \sigma \circ \pi \circ \rho \in \mathcal{T}\} \subseteq \{moet_{v,\pi} \mid \pi \in \mathcal{T}\}$. Hence the property follows immediately. □

Proof of Theorem 12 The algorithm starts by constructing the simple-history context $[\![A, B]\!]$ of node v. By Theorem 3, $paths([\![A, B]\!])$ covers node v. Next, it obtains contexts $[\![A_1, B_1]\!]$ and $[\![A_1, B_1]\!]$ of node v, by performing a vertical split of context $[\![A, B]\!]$. By Theorem 4, $paths([\![A_1, B_1]\!]) \cup paths([\![A_2, B_2]\!])$ covers node v, and $[\![A_1, B_1]\!]$ and $[\![A_2, B_2]\!]$ are divergent. The algorithm then performs a horizontal split of contexts $[\![A_1, B_1]\!]$ and $[\![A_2, B_2]\!]$, thus obtaining contexts

$$M_1 = \left\{[\![D, B_1 \cap Z_D]\!] \mid D \in \mathcal{D}_1\right\}, \quad \text{and} \quad M_2 = \left\{[\![D, B_2 \cap Z_D]\!] \mid D \in \mathcal{D}_2\right\}.$$

By Theorem 5,

$$paths\left([\![A_1, B_1]\!]\right) \cup \bigcup_{D \in \mathcal{D}_2} paths\left([\![D, B_2 \cap Z_D]\!]\right)$$

covers node v. Again, by Theorem 5,

$$\bigcup_{D \in \mathcal{D}_1} paths\left(\llbracket D, B_1 \cap Z_D \rrbracket\right) \cup \bigcup_{D \in \mathcal{D}_2} paths\left(\llbracket D, B_1 \cap Z_D \rrbracket\right) = \bigcup_{D \in \mathcal{D}_1 \cup \mathcal{D}_2} paths\left(\llbracket D, B_1 \cap Z_D \rrbracket\right)$$

covers node v.

By Theorem 5, $paths(\llbracket D, B_1 \cap Z_D \rrbracket)$ and $paths(\llbracket D', B_1 \cap Z_{D'} \rrbracket)$ are divergent, for $D, D' \in \mathcal{D}_1$, with $D \neq D'$. Also, $paths(\llbracket D, B_2 \cap Z_D \rrbracket)$ and $paths(\llbracket D', B_2 \cap Z_{D'} \rrbracket)$ are divergent, for $D, D' \in \mathcal{D}_2$, with $D \neq D'$.

Now choose any $D \in \mathcal{D}_1$ and $D' \in \mathcal{D}_2$. Since the contexts $\llbracket A_1, B_1 \rrbracket$ and $\llbracket A_2, B_2 \rrbracket$ are divergent, and since

$$paths\left(\llbracket D, B_1 \cap Z_D \rrbracket\right) \subseteq \llbracket A_1, B_1 \rrbracket \text{ and } paths\left(\llbracket D', B_2 \cap Z_{D'} \rrbracket\right) \subseteq \llbracket A_2, B_2 \rrbracket,$$

it follows that $paths(\llbracket D, B_1 \cap Z_D \rrbracket)$ and $paths(\llbracket D', B_2 \cap Z_{D'} \rrbracket)$ are also divergent. □

Appendix 2: The TriCore TC1796 processor

The TriCore TC1796

The TC1796 is a fairly complex 32-bit microprocessor targeted at the automotive market that offers simple versions of many of the performance-enhancing features found in modern desktop and server processors, like caching, pipelining, and branch prediction. Contrary to popular belief, the TriCore TC1796 has only a single processing core. It, however, features three parallel instruction pipelines that allow parallel processing of different types of instructions, as well as a separate floating point unit.

Here we provide an overview of three features that we consider particularly relevant for WCET analysis: the memory subsystem, the instruction pipeline, and branch prediction. More details about the TriCore architecture and the TC1796 microprocessor can be found in the corresponding technical manuals (Infineon Technologies AG 2006, 2007).

Figure 10 provides a high-level view on the structure of the bus systems of the TC1796 processor: The basic design is based on a Harvard architecture, with separate interfaces for program and data memory (PMI, DMI). On the whole, the memory subsystem is fairly complex and allows for an abundance of different memory configurations that can be chosen by the system designer. The system consists of the following components:

Program memory interface (PMI): The PMI (cf. Fig. 11) is directly connected to the CPU and is responsible for all accesses to program memory. It is equipped with 64 KiB of RAM, of which 16 KiB can be used as instruction cache (ICACHE) and of which 48 KiB can be used as scratchpad memory (SPRAM). The ICACHE is a two-way set-associative LRU cache with a line size of 256 bits, and a validity granularity

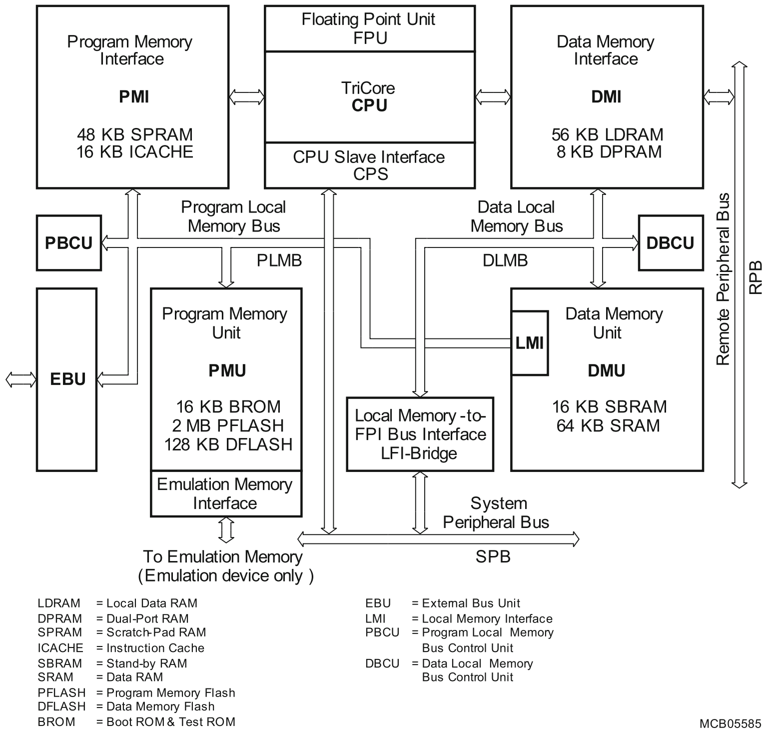


Fig. 10 Block diagram of the bus systems of the TC1796. Illustration taken from Infineon Technologies AG (2007)

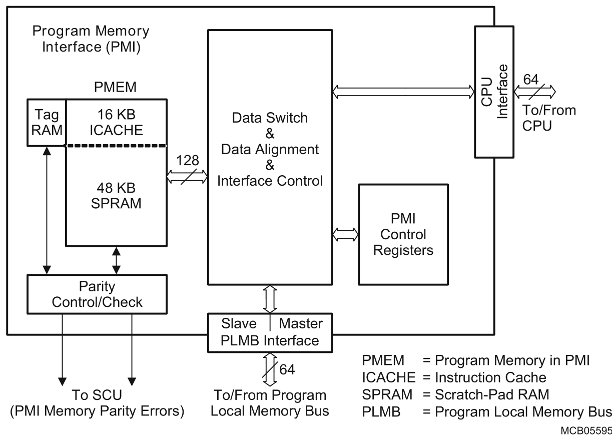


Fig. 11 Block diagram of the TC1796 program memory interface (PMI). Illustration taken from Infineon Technologies AG (2007)

of 4 double-words (64 bit). The ICACHE can be globally invalidated to provide support for cache coherency. The ICACHE can be bypassed, to provide direct access to the program local memory-bus (PLMB). The CPU interface supports unaligned,

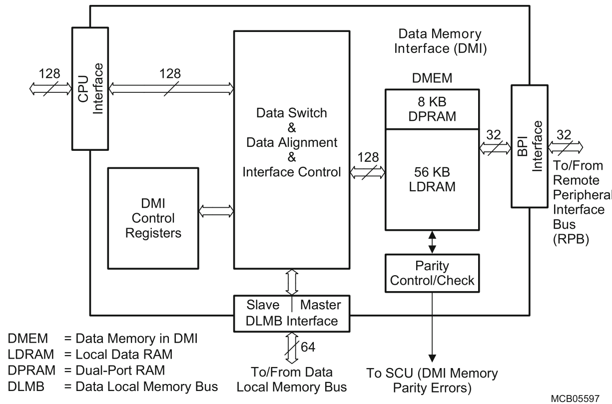


Fig. 12 Block diagram of the TC1796 data-memory interface (DMI). Illustration taken from Infineon Technologies AG (2007)

Bus Cycle	1	2	3	4	5	6	7
Transfer 1	Request/ Grant	Address Cycle	Data Cycle	Data Cycle	Data Cycle	Data Cycle	
Transfer 2		Request/ Grant	Address Cycle				Data Cycle

MCA05629

Fig. 13 Timing of a LMB basic transaction. Illustration taken from Infineon Technologies AG (2007)

Bus Cycle	1	2	3	4	5
Transfer 1	Request / Grant	Address Cycle	Data Cycle		
Transfer 2		Request / Grant	Address Cycle	Data Cycle	
Transfer 3		Request / Grant		Address Cycle	Data Cycle

MCA05628 mod

Fig. 14 Timing of a LMB block transaction. Illustration taken from Infineon Technologies AG (2007)

i.e., 16-bit aligned, accesses with a penalty of one cycle for unaligned accesses that cross cache lines.

Data memory interface (DMI): The DMI (cf. Fig. 12) is directly connected to the CPU and is responsible for all accesses to data memory. It is equipped with 64 KiB of RAM, 8 KiB of which is dual-port RAM (DPRAM) that is accessible from the CPU and from the remote peripheral bus (RPB), and of which 56 KiB is local data memory (LDRAM). The CPU interface supports unaligned, i.e., 16-bit aligned, accesses with a minimum penalty of one cycle for unaligned accesses that cross cache lines. There is a directly accessible interface to the data local memory-bus (DLMB) that provides access to the rest of the system.

Program local memory-bus (PLMB): The DLMB is a synchronous, pipelined bus that connects the DMI to the rest of the data-memory system. The bus protocol supports

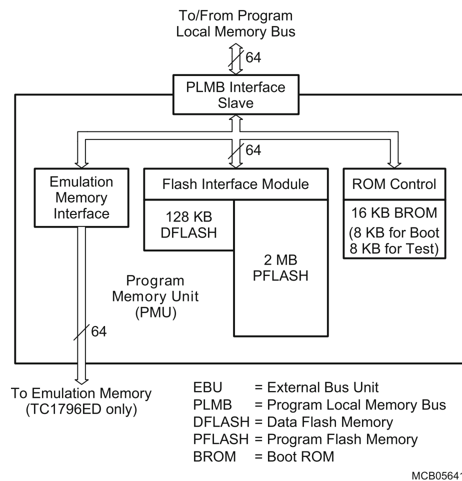


Fig. 15 Block diagram of the TC1796 program memory unit (PMU). Illustration taken from Infineon Technologies AG (2007)

single transfers of 8, 16, 32, and 64 bits (cf. Fig. 13), as well as block transfers of 64 bits (cf. Fig. 14). The PLMB is managed by the program local memory-bus control unit (PBCU), which handles requests from PLMB master devices, which are the PMI and the program memory unit (PMU). Access arbitration takes place in each cycle that precedes a possible address cycle, and is based on the priority of the requesting master device. The PMI has priority over the PMU. Busy slave devices can delay the start of a PLMB transaction.

Data local memory-bus (DLMB): The DLMB is a synchronous, pipelined bus that connects the DMI to the rest of the data-memory system. The bus protocol supports single transfers of 8, 16, 32, and 64 bits (cf. Fig. 13), as well as block transfers of 64 bits (cf. Fig. 14). The DLMB is managed by the data local memory-bus control unit (DBCU), which handles requests from PLMB master devices, which are the DMI and the data-memory unit (DMU). Access arbitration takes place in each cycle that precedes a possible address cycle, and is based on the priority of the requesting master device. The DMI has priority over the DMU. Busy slave devices can delay the start of a DLMB transaction.

Program local memory-bus control unit (PBCU): The PBCU is responsible for managing data transfers on the PLMB.

Data local memory-bus control unit (DBCU): The PBCU is responsible for managing data transfers on the DLMB.

Program memory unit (PMU): The PMU (cf. Fig. 15) is connected to the PLMB. It is equipped with 2MiB of program flash memory (PFLASH), 128 KiB of data flash memory (DFLASH), and 16 KiB of boot ROM (BROM).

Data memory unit (DMU): The DMU (cf. Fig. 16) is connected to the DLMB. It is equipped with 64 KiB of SRAM and 16 KiB of standby memory (SBRAM).

Local memory interface (LMI): The LMI is a part of the DMU. It allows the DMI and the DMU to access the PLMB, thereby enabling data transfers to and from other PLMB devices, like the EBU.

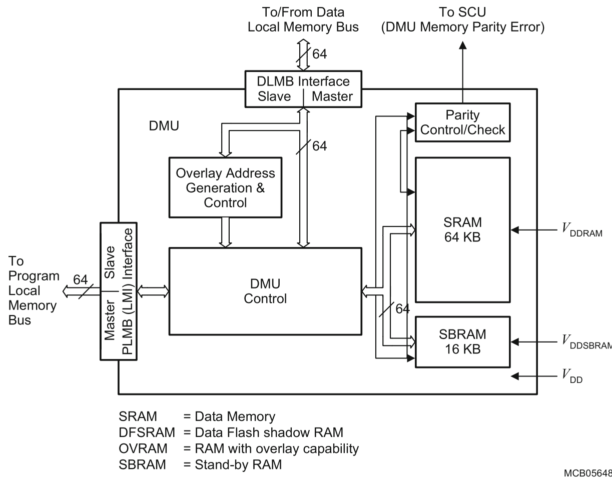


Fig. 16 Block diagram of the TC1796 data-memory unit (DMU). Illustration taken from Infineon Technologies AG (2007)

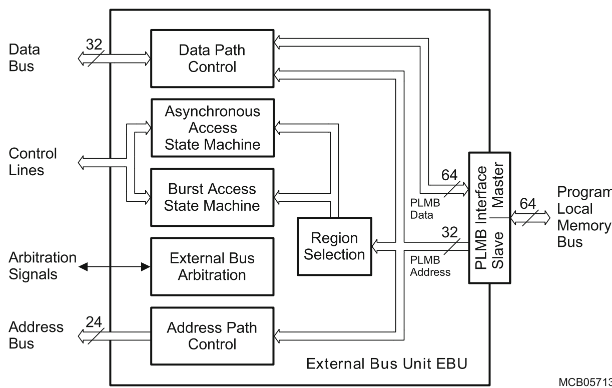


Fig. 17 Block diagram of the TC1796 external bus unit (EBU). Illustration taken from Infineon Technologies AG (2005)

External bus unit (EBU): The EBU (cf. Fig. 17) is connected to the PLMB and serves as an interface to external memory or peripheral units. It supports asynchronous or burst-mode external accesses. The external bus may be shared with other bus masters. Arbitration can be performed either by the EBU, or by an external bus master.

Local Memory to FPI bridge (LFI bridge): The LFI forms a bi-directional bridge between the DLMB and the peripheral FPI bus.

Figure 18 provides a high-level view of the structure of the TC1796 CPU, which consists of the following components:

Instruction fetch unit: The instruction-fetch unit pre-fetches and aligns incoming instructions from the PMI and issues them to the appropriate instruction pipeline.

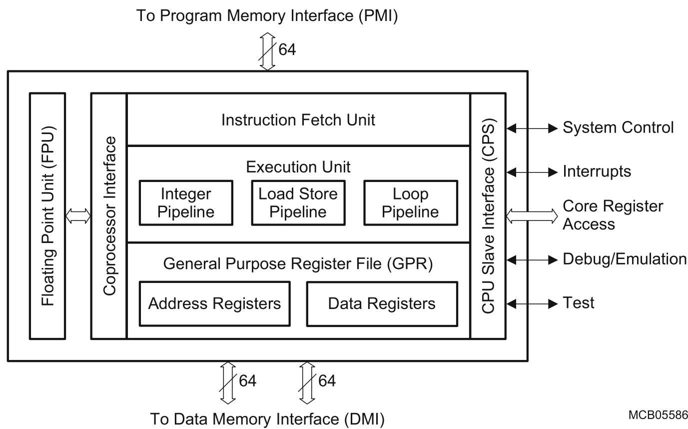


Fig. 18 Block diagram of the TC1796 central processing unit (CPU), indicating the superscalar pipeline design. Illustration taken from Infineon Technologies AG (2005)

Execution unit: The execution unit consists of three parallel pipelines, each of which can process a different type of instructions. The integer pipeline and the load/store pipeline each consist of the following four stages: fetch, decode, execute, and write-back. The loop pipeline consists of the two stages: decode and write-back. The integer pipeline handles data arithmetic instructions, including data conditional jumps. The load/store pipeline handles load/store memory accesses, address arithmetic, unconditional jumps, calls, and context switches. The loop pipeline handles loop instructions, providing zero-overhead loops. The execution unit also maintains the program counter.

General purpose register file (GPR): The GPR provides 16 address registers and 16 data registers.

CPU slave interface (CPS): The CPS provides accesses to the interrupt service requests registers.

Floating point unit (FPU): The FPU is an optional, partially IEEE-754 compatible component for processing floating-point instructions.

Individual instructions may experience a jitter in execution time, due to pipeline stalls. Figure 19 illustrates an example of a pipeline hazard that is resolved by a pipeline stall: In this case, the integer pipeline is processing a multiply-and-accumulate (MAC) instruction, which requires two cycles in the execute stage. At the same time the load/store pipeline is processing a load instruction to the write register of the MAC instruction, which results in a write-after-write hazard.

For conditional branch instructions, the TC1796 uses a simple, static predictor that implements the following rules: Backward and short forward branches (16-bit branches with positive displacement) are predicted taken. Long forward branches are predicted not taken. Table 5 summarizes the cycle penalties for each combination of predicted and actual behavior.

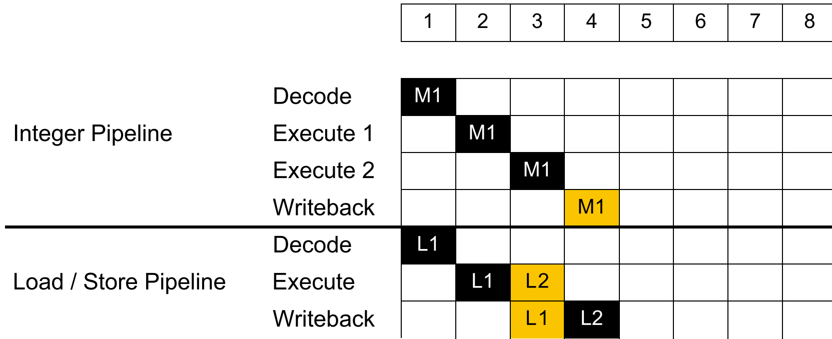


Fig. 19 Example of a pipeline hazard in the TC1796 CPU. Illustration taken from Infineon Technologies AG (2000)

Table 5 Branch penalties of the TC1796 processor, for all combinations of prediction and actual outcome

Prediction	Outcome	Penalty (cycles)
Not taken	Not taken	1
Not taken	Taken	3
Taken	Not taken	3
Taken	Taken	2

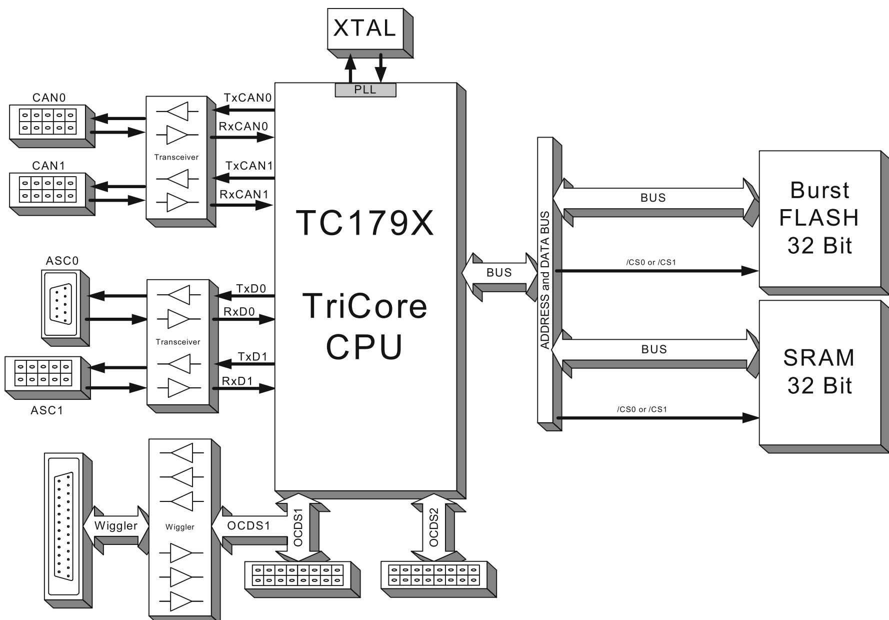


Fig. 20 Block schematics of the TriBoard-T179X, indicating the connection to external memory, as well as the OCDS debugging interface. Illustration taken from Infineon Technologies AG (2005)

The TriBoard TC179X evaluation board

The TriBoard is equipped with 4 MB of Burst Flash memory and 1 MB of asynchronous SRAM, which are both connected to the processing core via the External Bus Unit of the processor, and these are the only devices that are connected to the EBU (cf. Fig. 20). The *Clock Generation Unit*, which is controlled by an external crystal oscillator, produces a clock signal f_{OSC} at 20 MHz. The CPU clock runs at 150 MHz, and the system clock at 75 MHz. More details can be found in the board manual (Infineon Technologies AG 2005).

References

Audsley NC, Burns A, Richardson MF, Wellings AJ (1991) Hard real-time scheduling: the deadline-monotonic approach. In: Halang WA, Ramamritham K (eds) Proceedings of 8th IEEE workshop on real-time operating systems and software (RTOSS'91), pp. 127–132. Pergamon Press

Bernat G, Colin A, Petters S (2002) Wcet analysis of probabilistic hard real-time systems. In: Proceedings of 23rd real-time systems symposium (RTSS'02), pp. 279–288, Austin, Texas, USA

Bernat G, Colin A, Petters S (2003) pwcet: a tool for probabilistic worst-case execution time analysis of real-time systems. In: 3rd international workshop on worst-case execution time analysis (WCET'03), pp. 21–38, Porto, Portugal

Bünte S, Zolda M, Kirner R (2011) Let's get less optimistic in measurement-based timing analysis. In: 6th IEEE international symposium on industrial embedded systems (SIES'11), Los Alamitos, CA. IEEE

Colin A, Puaut I (2000) Worst case execution time analysis for a processor with branch prediction. Real-Time Syst 18(2):249–274

Dijkstra E (1970) Notes on structured programming. Circulated privately

Dertouzos M, Mok AK (1989) Multiprocessor online scheduling of hard-real-time tasks. IEEE Trans Softw Eng 15(12):1497–1506

Engblom J, Ermedahl A (1999) Pipeline timing analysis using a trace-driven simulator. In: Proceedings of 6th international conference on real-time computing systems and applications (RTCSA '99), pp. 88–95, Hong Kong, China

Ferdinand C, Martin F, Wilhelm R, Alt M (1999) Cache behavior prediction by abstract interpretation. Sci Comput Progr 35(2–3):163–189

Holzer A, Schallhart C, Tautschnig M, Veith H (2008) Fshell: systematic test case generation for dynamic analysis and measurement. In: Proceedings of 20th international conference on computer aided verification (CAV'08). LNCS, vol. 5123, pp. 209–213. Springer, Princeton

Holzer A, Schallhart C, Tautschnig M, Veith H (2011) An introduction to test specification in fql. In: Sharon B, Daniel K, Orna R (eds) Proceedings of Haifa Verification Conference (HVC'10), Lecture Notes in Computer Science, vol. 6504, pp. 9–22. Springer, Haifa

Infineon Technologies AG (2000) St.-Martin-Strasse 53, D-81541 München, Germany. TriCore(TM) 1 Pipeline Behaviour and Instruction Execution Timing

Infineon Technologies AG (2005) St.-Martin-Strasse 53, D-81541 München, Germany. TriBoard TC179X Hardware Manual

Infineon Technologies AG (2006) St.-Martin-Strasse 53, D-81541 München, Germany. TriCore 1 32-bit Unified Processor Core

Infineon Technologies AG (2007) St.-Martin-Strasse 53, D-81541 München, Germany. TC1796 32-Bit Single-Chip Microcontroller

Kirner R, Wenzel I, Rieder B, Puschner P (2005) Intelligent systems at the service of mankind. Chapter using measurements as a complement to static worst-case execution time analysis, vol 2. UBooks, Augsburg

Liu CL, Layland JW (1973) Scheduling algorithms for multiprogramming in a hard-real-time environment. J ACM 20(1):46–61

Li Y-TS, Malik S (1997) Performance analysis of embedded software using implicit path enumeration. IEEE Trans Comput-Aided Des Integr Circuits Syst 16(12):1477–1487

Li Y-TS, Malik S, Wolfe A (1995) Efficient microarchitecture modeling and path analysis for real-time software (rtss'95). In: 16th IEEE real-time systems symposium (RTSS'95), pp. 298–307. Pisa

Li Y-TS, Malik S, Wolfe A (1996) Cache modeling for real-time software: beyond direct mapped instruction caches. In: Proceedings of 17th IEEE real-time systems symposium (RTSS'96), pp. 254 –263. IEEE, Washington

Li Y-TS, Malik S, Wolfe A (1999) Performance estimation of embedded software with instruction cache modeling. ACM Trans Des Autom Electron Syst 4(3):257–279

Lundqvist T, Stenström P (1998) Integrating path and timing analysis using instruction-level simulation techniques. In: Proceedings of ACM SIGPLAN workshop on languages, compilers, and tools for embedded systems (LCTES'98), pp. 1–15. Springer, New York

Lundqvist T, Stenström P (1999) An integrated path and timing analysis method based on cycle-level symbolic execution. Real-Time Syst 17(2–3):183–207

Nassi I, Shneiderman B (1973) Flowchart techniques for structured programming. SIGPLAN Not 8(8):12–26

Ottosson G, Sjödin M (1997) Worst case execution time analysis for modern hardware architectures. In: ACM SIGPLAN workshop on languages, compilers and tools for real-time systems, pp. 47–55. ACM, Las Vegas

Paparazzi (2012) The free autopilot. http://paparazzi.enac.fr/, August

Park CY, Shaw A (1991) Experiments with a program timing tool based on source-level timing schema. IEEE Comput 24:48–57

Puschner P, Koza C (1989) Calculating the maximum execution time of real-time programs. Real-Time Syst 1(2):159–176

Puschner P, Schedl A (1993) A tool for the computation of worst case task execution times. In: Proceedings of 5th Euromicro workshop on real-time systems (EURO-RTS'93), pp. 224–229. IEEE, Oulu

Puschner P, Schedl A (1997) Computing maximum task execution time—a graph-based approach. J Real-Time Syst 13(1):67–91

Puschner P (1998) A tool for high-level language analysis of worst-case execution times. In: Proceedings of 10th Euromicro workshop on real-time systems (Euro-Rts'98), pp. 130–137. Berlin

Schoeberl M (2009) JOP reference handbook: building embedded systems with a java processor. CreateSpace

Schneider J, Ferdinand C (1999) Pipeline behavior prediction for superscalar processors by abstract interpretation. SIGPLAN Not 34(7):35–44

Shaw A (1989) Reasoning about time in higher-level language software. IEEE Trans Softw Eng 15(7):875–889

Stappert F, Altenbernd P (2000) Complete worst-case execution time analysis of straight-line hard real-time programs. J Syst Architect 46(4):339–355

Stattelmann S, Martin F (2010) On the use of context information for precise measurement-based execution-time estimation. In: Björn L (ed) Proceedings of 10th international workshop on worst-case execution time (WCET) Analysis (WCET'10). OpenAccess Series in Informatics (OASIcs), vol. 15, pp. 64–76. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, Brussels

Theiling H, Ferdinand C (1998) Combining abstract interpretation and ilp for microarchitecture modelling and program path analysis. In: Proceedings of 19th IEEE real-time systems symposium (RTSS'98), pp. 144–153. Madrid

Thesing S, Souyris J, Heckmann R, Randimbivololona F, Langenbach M, Wilhelm R, Ferdinand C (2003) An abstract interpretation-based timing validation of hard real-time avionics software. In: Proceedings of 2003 international conference on dependable systems and networks (DSN'03). IEEE, San Francisco

von Hanxleden R, Holsti N, Lisper B, Ploedereder E, Wilhelm R, Bonenfant A, Cassé H, Bünte S, Fellger W, Gepperth S, Gustafsson J, Huber B, Islam NM, Kästner D, Kirner R, Kovacs L, Krause F, de Michiel M, Olesen MC, Prantl A, Puffitsch W, Rochange C, Schoeberl M, Wegener S, Zolda M, Zwirchmayr J (2011) Wcet tool challenge 2011: Report. In: Proceedings of 11th international workshop on worst-case execution time (WCET) analysis (WCET'11), Porto, Portugal. The analysis problems for the WCET Tool Challenge can be found at http://www.mrtc.mdh.se/projects/WCC/2011

Wenzel I, Kirner R, Rieder B, Puschner P (2009) Measurement-based timing analysis. In: Proceedings of the 3rd international symposium on leveraging applications of formal methods, verification and validation (ISoLA'08). Communications in computer and information science, vol. 17, pp. 430–444. Springer, Porto Sani

Xu J, Parnas D (1990) Scheduling processes with release times, deadlines, precedence and exclusion relations. IEEE Trans Softw Eng 16(3):360–369

Zolda M (2012) Precise measurement-based worst-case execution time estimation. PhD thesis, Vienna University of Technology, Karlsplatz 13, Vienna

Michael Zolda is a research fellow at the University of Hertfordshire, UK. He received his doctoral degree from Vienna University of Technology in 2012. From 2007 to 2011 he worked on the FWF/DFG Research Project FORTAS-RT on execution time analysis of real-time systems. He has published multiple papers at acclaimed international conferences and workshops. Currently he is working on dependable stream processing systems within the EC/transnational research project CRAFTERS. He is also taking part in the European ICT COST Action TACLe (Timing Analysis on Code-Level).

Raimund Kirner is a Reader in Cyberphysical Systems at the University of Hertfordshire. He has published more than 90 refereed journal and conference papers and received two patents. He received his PhD in 2003 from the TU Vienna and his Habilitation in 2010. His research focus is on embedded computing, parallel computing, and system reliability. He currently works on adequate hardware and software architectures to bridge the gap between the many-core computing and embedded computing. He also published excessively on worst-case execution time analysis and served as PC chair of WDES'06, WCET'08, and SEUS'13. He is the local principal investigator of the Artemis-JU Project CRAFTERS and was local co-investigator of the FP7 Project ADVANCE. Further, he has been the principal investigator of three research projects funded by the Austrian Science Foundation (COSTA, FORTAS, SECCO). He is a member of the IEEE, the ACM, and the IFIP Working Group 10.4 (Embedded Systems).