Worst-Case Execution Time Analysis for Processors showing Timing Anomalies *

Technical Report 01/2009

Raimund Kirner, Albrecht Kadlec, Peter Puschner
Institut für Technische Informatik
Technische Universität Wien
Treitlstraße 3/182/1
A-1040 Wien, Austria
{raimund,albrecht,peter}@vmars.tuwien.ac.at

Abstract

The analysis of the worst-case execution time (WCET) requires detailed knowledge of the program behavior. On modern processors the instruction timing heavily depends on the processor state. WCET analysis therefore has to the model processor behavior in detail. This analysis is challenging in case of so called timing anomalies, which violate the continuity properties proportionality and continuity of the timing behavior.

In this paper we present a formal definition of timing anomalies and a formal analysis of their impact on WCET analysis. So far, timing anomalies have been described in the composition of instruction sequences. We call them series timing anomalies. We show that timing anomalies are also a problem when composing the overall state of sub-states of the hardware, which we call parallel timing anomalies. The results show that most types of timing anomalies are an impediment to an efficient timing analysis. Some of them can be handled effectively by the proposed composition methods. Finally, we present different methods to avoid the occurrence of timing anomalies.

This report is not a final treatise on timing anomalies, but provides a concise problem characterization including some hints on how to deal with timing anomalies respectively how to avoid them.

Keywords: Worst-Case Execution Time Analysis, Timing Anomalies, Predictable Timing, Processor-Behavior Analysis, Abstraction

Abbreviations: WCET – worst-case execution time, BCET – best-case execution time, TA – timing anomaly, TRDCS – timing-relevant dynamic computer state

*The research leading to these results has received funding from the Austrian Science Fund (Fonds zur Förderung der wissenschaftlichen Forschung) within the research project “Compiler-Support for Timing Analysis” (COSTA) and the European Community’s Seventh Framework Programme [FP7/2008-2011] under grant agreement no 214373 (ArtistDesign, http://www.artist-embedded.org/).
1 Introduction

The knowledge of the worst-case execution time (WCET) of software components is a prerequisite for ensuring the timeliness of a real-time system. Since the end of the 1980s significant effort has been spent on research towards the development of WCET analysis tools.

The two main tasks of WCET analysis tools are the control-flow analysis (also called path analysis [11] or high-level analysis) that determines the (in)feasible paths in a program and the processor-behavior analysis (also known as hardware modeling [11] or low-level analysis) that assesses instruction timing [30].

Within this paper we discuss an open problem of processor-behavior analysis, namely the occurrence of so-called timing anomalies [15, 29, 22]. Timing anomalies are a challenge for WCET analysis, because they violate the continuity properties “proportionality” and “monotony” of program execution time. As instruction timing depends on the concrete hardware and also on the current state of the hardware (cache content, pipeline state, branch predictor, etc.), Section 2.1 analyzes the constituents of the state space and define what is relevant for WCET analysis (the so-called timing-relevant dynamic computer state).

Timing anomalies so far have only been discussed in the context of composing instruction sequences. In Section 3 we motivate that it is also a relevant problem when composing the state space of the computer. We present known instances of timing anomalies in Section 3 and provide compact formal definitions for timing anomalies, which are due to their compact definition suitable for analyzing correctness issues for different composition techniques. We define timing anomalies as timing effects due to changes of the initial state. We reformulate the resource-allocation criterion as a necessary but not sufficient precondition for the occurrence of timing anomalies. In Section 4 we discuss how the instruction timing of two subsequent instruction sequences can be composed and proof the fundamental limitation of this technique due to timing anomalies. Analog, in Section 5 we discuss two approaches of breaking down the state space by subsequent analysis phases and proof which types of parallel timing anomalies can be handled and which type provides fundamental problems for each of them. In Section 6 we discuss practical issues of timing anomalies on WCET analysis and discuss methods of how to avoid anomalous behavior.

2 Worst-Case Execution Time Analysis

WCET analysis is about finding the longest feasible path through a program, where length means execution time [11, 30].

To find the longest path, we first have to determine the set of all feasible program paths, or at least a tight approximation thereof. Control-flow analysis can be used to find (in)feasible paths automatically [7].

After solving the path problem we have to calculate the execution time of each path, which could be done by summing up the execution times of all instructions along that path. But this simplified strategy becomes quickly intractable due to the potentially large number of different paths in a program. Thus, the obvious solution to this problem is to use divide and conquer by local WCET calculations, i.e., if we have an if-else statement we calculate the WCET along the then branch and along the else branch and take the maximum of it. Then we virtually replace the if-else statement with an artificial statement whose execution time is the calculated maximum. Repeating this in a bottom-up manner for all branches in the control-flow graph yields a conservative approximation of the WCET. In principle, this strategy is used by the tree-based [19] and the path-based [8] WCET calculation method.

The concept of local WCET calculation based on divide and conquer has some limitations. First, it is not possible to use global flow information, i.e., flow constraints between control-flow locations from different program scopes. Second, the instruction timing of modern processors with features like caches or pipelines depends on the processor state, which makes it difficult to precisely calculate the WCET of a local instruction sequence without analyzing the preceding program execution. Thus, global WCET
analysis techniques are used for modern processors. For example, the *implicit path-enumeration technique* allows to consider arbitrary linear flow constraints \[14, 20\]. One or more program analysis phases precede the longest path search to calculate the instruction timing \[18, 5\].

### 2.1 Timing-Relevant (System) State

On modern processors with peak-performance improving features like caches or pipelines, the WCET of an instruction sequence depends on the initial hardware state at the time the execution of the instruction sequence is started. Thus, knowing the computer state is important for WCET analysis. In this context a suitable definition of state is given by Mesarovic and Takahara:

*The state enables the determination of a future output solely on the basis of the future input and the state the system is in. In other word, the state enables a decoupling of the past from the present and future. The state embodies all past history of a system. Knowing the state supplants knowledge of the past.*” \[16\]

![Timing-Relevant Computer State (TRCS) vs. Timing-Relevant Computer Configuration (TRCC) and Timing-Relevant Dynamic Computer State (TRDCS)](image)

For WCET analysis it is necessary to analyse the computer system and identify the timing-relevant computer state (TRCS, see Definition 2.1). As shown in Figure 1, the TRCS may not only include the timing-relevant state within the processor on which the code to be analyzed is executed. The computer may also contain other components like external caches or I/O modules that influence the instruction timing.

**Definition 2.1 Timing-Relevant Computer State (TRCS)** *The timing-relevant computer state TRCS consists of those elements of the computer state CS whose values can influence the execution time of at least one instruction executed on that computer. Conversely, the values of the state elements not included in TRCS do not influence the instruction timing.*

The problem with analyzing the TRCS is that the state space of the TRCS can be huge, which makes WCET analysis extremely resource-intensive. To reduce the state space, it is useful to identify those elements of the TRCS that remain constant throughout the execution of the code analyzed for its WCET. We call those TRCS elements as *timing-relevant computer configuration* (TRCC, see Definition 2.2). For example, embedded software that is executed on different target platforms, may be parametrized and configured during the system initialization phase. Identifying the TRCC allows us to reduce the resources needed for the WCET analysis and may thus help us to derive more precise WCET bounds.
Definition 2.2 Timing-Relevant Computer Configuration (TRCC) The timing-relevant computer configuration TRCC of a program scope \( S \) consists of those elements of the timing-relevant computer state TRCS whose values remain constant throughout the execution of the considered program scope \( S \). Note that the TRCS can be modified outside \( S \).

Based on the TRCS and the TRCC we can define the state space that contributes to the complexity of the WCET analysis: the timing-relevant dynamic computer state (TRDCS, see Definition 2.3). Minimizing the TRDCS is a good strategy to reduce the complexity of WCET analysis. This can be also supported by an adequate software design.

Definition 2.3 Timing-Relevant Dynamic Computer State (TRDCS) The timing-relevant dynamic computer state TRDCC consists of those elements of the timing-relevant computer state TRCS that are not part of the timing-relevant computer configuration: \( TRDCS = TRCS \setminus TRCC \).

As said above, the TRDCS relevant for WCET analysis may include the state of other computer components besides the processor state. For example, to predict the WCET of an instruction sequence on a processor with speculative execution, the code memory (set of potentially executed instructions) contributes to the TRDCS of an instruction sequence.

2.2 Notation

The following sections discuss several formal properties on WCET analysis. To keep the definition of these properties short and intuitive we use the following notation:

\[ T(I, s) \] the execution time of an instruction sequence \( I = I_0 \circ I_1 \circ \ldots \circ I_n \) with the initial TRDCS \( s \).

The operator \( \circ \) combines individual instructions or instruction sequences to a combined instruction sequence.

\[ T_{hw_A}(I, a) \] the total latency of processor component \( hw_A \) when executing instruction sequence \( I \) with initial local state \( a \in A \), where \( A \) is the TRDCS state space of processor component \( hw_A \). The total latency of a processor component is the cumulated time where this component performs some data processing. That part of the execution of \( I \) where \( hw_A \) is inactive does not contribute to its total latency.

For example, the total latency of a data cache when executing an instruction sequence \( I \) is the cumulated time the data cache needs to exchange data with the main memory and with the processor registers (including waiting time to get data ready). The TRDCS of the hardware model can be optimized for the instruction sequences of interest. In the data cache, for example, all addresses that will not be used by the instruction sequences of interest can be subsumed as “not part of the TRDCS”.

\[ T_{max}(I, S) = \max\{T(I, s) \mid s \in S\} \] the WCET of instruction sequence \( I \) where \( S \) is the set of potential initial states for execution of \( I \).

\[ \Delta(I, s, s') = T(I, s') - T(I, s) \] the difference of execution time of instruction sequence \( I \) for different initial states \( s \) and \( s' \).

\[ \Delta_{hw_A}(I, a, a') = T_{hw_A}(I, a') - T_{hw_A}(I, a) \] the difference of total latency of processor component \( hw_A \) when executed from different local initial states \( a \) and \( a' \) with \( a, a' \in A \).

\( \mathbb{N}_I \) the set of possible initial states of an instruction sequence \( I \), i.e., states that a program analysis classified as reachable at the beginning of \( I \).
The set of only those initial states \( s \in \mathbb{IN}_I \) of an instruction sequence \( I \) where the execution time \( T(I, s) \) is maximal:

\[
\mathbb{IN}_{I,max} = \{ s \mid s \in \mathbb{IN}_I \wedge \forall s' \in \mathbb{IN}_I. T(I, s') \leq T(I, s) \}
\]

The concept of total latency (\( T_{hw_a}(I, a) \)) is used to discuss the state partitioning as described in Section 2.3.2 and Section 5.

2.3 Fighting Complexity of Analysis

Analyzing the instruction timing on modern processors may impose resource requirements that are too high to be acceptable in practice. There are basically two problems: 1) the number of different paths may be too large to calculate \( T(I, s) \) for each execution path \( I \), 2) the TRDCS state space \( S \) may be too large to calculate \( T(I, s) \) for each state \( s \in S \).

To weaken or avoid the first problem, the standard approach is to calculate fixpoints over nodes of the control-flow structure (e.g., CFG), as described in Section 2.3.1. To weaken or avoid the second problem, the timing analysis can be done hierarchically over partitions of the TRDCS, as described in Section 2.3.2.

2.3.1 Series Decomposition

In real-size programs the enumeration and analysis of all execution paths is intractable, as the number of paths typically grows exponentially with program size. The common technique to avoid this problem is to use a collecting semantics (describes the reachable states for each program point) instead of the trace semantics (additionally describes the sequence in which the states appear).

The typical approach to calculate the collecting semantics is to calculate the set of reachable states at each node of the control-flow structure. This requires to calculate a fixpoint in case programs contain loops. For WCET analysis, the reachable TRDCS is computed for each control-flow node.

As the set of reachable states at each control-flow node can be quite large when calculating the fixpoint solution, a further approximation is needed.

A straight-forward approach for such an approximation is the series decomposition, as shown in Figure 2. The figure shows an example with an instruction sequence \( I = M \circ N \) and four potential input states \( s_1 \ldots s_4 \). Starting in state \( s_i \) on executing instruction sequence \( M \) only, the output state \( s'_i \) is produced. Completing the execution with \( s'_i \) as input state on instruction sequence \( N \) produces the output state \( s''_i \). The individual execution times \( T(M, s_i) \), \( T(N, s'_i) \) for instruction sequence \( M \) and \( N \) are drawn as black horizontal bars. The overall execution times \( T(M \circ N, s_i) \) of the complete instruction sequence \( I \) are drawn as rectangles.
The idea of *series decomposition* is to calculate the longest execution time for each control-flow node. But instead of calculating $k$ output states for $k$ input states of the control-flow node, a state approximation for the output state is calculated, which is then forwarded as input state to the next node. A concrete technique including a discussion of correctness is presented in Section 4.

### 2.3.2 Parallel Decomposition

The idea of parallel decomposition is to calculate the WCET $T_{\text{max}}(I, S)$ of an instruction sequence $I = I_0 \circ I_1 \circ \ldots \circ I_n$ in two steps. Before this calculation, the TRDCS $S$ is partitioned into $S = A \cup B$, where $A$ is the state space of a processor component $hw_A$ and $B$ is the state space of other components $hw_B$ in the processor. For example, the hardware component $hw_A$ may be the instruction cache and the state fraction $B$ may cover the pipeline and the other processor components. For the timing function $T(I, s)$ of an instruction sequence $I$ given in Figure 3, the corresponding timing function with the partitioned state space $A$ and $B$ is shown in Figure 4. The state spaces of $hw_A$ and $hw_B$ may also overlap ($A \cap B \neq \emptyset$).

---

**Figure 3. Timing of Non-Partitioned TRDCS**

**Figure 4. Timing of Partitioned TRDCS**

In the first step, the timing of processor component $hw_A$ is analyzed and one state $a \in A$ is chosen (details describing the choice of $a$ will follow below). Based on this result, the overall processor timing is analyzed in the second step by searching the state space $B$ while using the result for state $a \in A$ to model the timing of $hw_A$. 
The challenge is to find a composable timing calculation method that can be used to calculate safe WCET bounds for the target processor of interest. Concrete calculation methods are discussed in Section 5.

3 Timing Anomalies

The name timing anomalies is used to describe system behavior where relaxing some constraints leads to an increase of the system timing. This is typically caused due to a greedy scheduler that cannot foresee the future impact of its local decisions. With respect to WCET analysis, for example, such a constraint may require the execution of two instruction sequences to finish within a given deadline. Decreasing the execution time of the first instruction sequence relaxes the constraint for the second instruction sequence to finish within the deadline, which can lead to timing anomalies.

3.1 Related Work on Timing Anomalies

Program execution time is not the first field where timing anomalies have been observed. For example, Graham described this effect for task scheduling [6]. He has shown that a greedy task scheduler can produce a longer schedule if the scheduling constraints are weakened, e.g., by using shorter tasks, less dependencies, or more processors.

Lundqvist and Stenström first described timing anomalies in the context of WCET analysis [15]. Their definition of timing anomalies is semi-formal. They have shown an example where a change from a cache hit to a cache miss of the first instruction of an instruction sequence on a processor with out-of-order pipeline and instruction cache can result in a decrease of the total execution-time. However, it has been shown that it is rather challenging to understand the potential triggers of timing anomalies. For example, Lundqvist and Stenström believed that it requires an out-of-order pipeline to trigger timing anomalies [15], which later turned out as too specific, see below.

Schneider developed an integrated WCET analysis method, i.e., he integrated response-time analysis with WCET analysis. He did this on a PowerPCC 755, where he demonstrated that timing anomalies occur on real processors [25]. Besides, he has also demonstrated the occurrence of the so-called domino effect, i.e., different states at the header of a loop do never converge during execution of the loop. Domino effects do not necessarily cause timing anomalies. In the concrete example shown by Schneider it did result in a strong timing anomaly, as he showed that a delay caused in the loop header results in a constant delay in each loop iteration, resulting in a total delay that is at least a linear function of the loop bound. Berg has shown an example of a domino effect that results in a weak timing anomaly with a constant execution-time change each loop iteration [2].

Wenzel et al. have analyzed different patterns of processor architectures to gain knowledge about the possible triggers of timing anomalies [28, 29]. They have shown that timing anomalies can occur even on processors with in-order execution. Further, Wenzel et al. provide a necessary precondition for the potential occurrence of timing anomalies, the resource allocation criterion. However, the concrete formulation of the criterion was a bit too restrictive as it covers only cases in which exactly one instruction changes its timing. This criterion needs to be generalized to cover timing anomalies caused by speculative execution [22] or certain cache replacement policies like pseudo-round robin [27].

Reineke et al. for the first time provide a formal definition of timing anomalies in the context of program execution time. Their definition of timing anomalies is based on a transition system as processor model where a timing anomaly occurs if the WCET path within a local scope is not part of the WCET path of a surrounding scope [22]. This was an important step towards improving the understanding of timing anomalies. However, this formalization of timing anomalies is rather complex, making it clumsy to use
as a tool for exploring safeness properties of WCET analyses. Further, the authors enumerated three different sources of timing anomalies without claiming this to be a complete list: speculation, scheduling, and cache effects. Reineke et al. only discuss the type of timing anomaly that Eisinger et al. call a *strong timing anomaly*: the case where a local increase of execution time leads to a global decrease [3]. The other case, where a local increase of execution time leads to an even larger global increase - called *weak timing anomaly* - is not treated in the context of WCET analysis because it does not hinder an efficient search of the worst-case. The definition of a timing anomaly given by Reineke et al. is specific to WCET analysis, while the original definition given by Lundqvist includes also processor behavior that is not a challenge for WCET analysis.

Kadlec et al. are working on code generation techniques to avoid the potential occurrence of timing anomalies. They sketched a first approach based on NOP-insertion [10]. Further results are necessary to judge the total potential of that approach.

The research described above discusses timing anomalies only in the context of serial decomposition of WCET analysis. Kirner et al. for the first time describe a class of timing anomalies not considered so far: timing anomalies in the context of WCET analysis using parallel decomposition. A timing anomaly in case of parallel decomposition is defined as the situation where the worst-case initial state of a hardware/processor sub-component is not part of the worst-case initial state of the total hardware/processor [12]. So far the authors formalized this new type of timing anomalies without discussing it in the context of the previously mentioned timing anomalies based on local changes within an instruction sequence.

### 3.2 Series Timing Anomalies

In this section we present a formal definition of the timing anomalies that were originally discussed by Lundqvist in the context of WCET analysis [15]. We call these timing anomalies “series timing anomalies” because they are challenging for the series decomposition described in Section 2.3.1.

The series timing anomalies are defined in Definition 3.1. In contrast to the original semi-formal definition given by Lundqvist, we define the timing anomalies over the instruction timing with changes of the initial state TRDCS. Using the state is natural since the state serves as an interface between the timing of two successive instruction sequences.

**Definition 3.1 (Series Timing Anomalies)** Given $T(I, s)$ models the timing of an instruction sequence $I$ with initial state $s$, the timing behavior $T(I, s)$ of an instruction sequence $I$ that is composed of two non-empty instruction sequences $M$ and $N$ ($I = M \circ N$) is called a series timing anomaly, iff at least one of the following two properties holds:

**TA-S-I** “Strong Series TA”, “Series Inversion”

$$\exists s, s' \in \mathbb{N}_M . \quad \Delta(M, s, s') > 0 \land \Delta(I, s, s') < 0$$

**TA-S-A** “Weak Series TA”, “Series Amplification”

$$\exists s, s' \in \mathbb{N}_M . \quad 0 < \Delta(M, s, s') < \Delta(I, s, s')$$

The timing anomaly **TA-S-I** of Definition 3.1 is also called “series inversion” as the local change has an inverse tendency globally. Reineke et al. called it also “strong timing anomaly” [22] as it causes problem for efficient series decomposition.

The other timing anomaly **TA-S-A** is also called “series amplification” or “weak timing anomaly”. In case of the **TA-S-A** it can happen that a local change may have an even larger change globally - this is why it is also called “series amplification”. The **TA-S-A** is also called “weak (series) timing anomaly” because it does not provide problems for the series decomposition of WCET analysis.
The definition of a timing anomaly given in Definition 3.1 follows the original definition by Lundqvist. Definition 3.2 gives a more strict definition that is equivalent to the definition given by Reineke et al., which describes exactly the case that causes problems for efficient WCET analysis using series decomposition.

**Definition 3.2 (Worst-Case Series Timing Anomalies)** The timing behavior \( T(I, s) \) of an instruction sequence \( I \) that is composed of two non-empty instruction sequences \( M \) and \( N \) \( (I = M \circ N) \) is called a worst-case series timing anomaly, iff at least one of the following two properties holds:

**TAW-S-I “Worst-Case Series Inversion”**
\[
\exists s \in \mathbb{IN}_M, \forall s' \in \mathbb{IN}_{M,\max}. \quad \Delta(I_1, s, s') > 0 \land \Delta(I_G, s, s') < 0
\]

**TAW-S-A “Worst-Case Series Amplification”**
\[
\exists s \in \mathbb{IN}_M, \forall s' \in \mathbb{IN}_{M,\max}. \quad 0 < \Delta(I_1, s, s') < \Delta(I_G, s, s')
\]

Note that the definition given in Definition 3.2 is almost identical with the definition given in Definition 3.1, with the small difference that is uses \( \forall s' \in \mathbb{IN}_{M,\max} \) instead of \( \exists s' \in \mathbb{IN}_M \). The worst-case timing anomalies are more specific than the others, which is, besides the specific elements to compare, due to the \( \forall \) quantifier instead the \( \exists \) quantifier. However, this difference is only significant for a concrete program. Given that a program \( P_1 \) including an instruction sequence \( I \) has the set of worst-case initial states \( \mathbb{IN}_{I,\max}^1 \), there might exist a program \( P_2 \) including the same instruction sequence \( I \) where the set of initial states \( \mathbb{IN}_2^1 \) is equal to \( \mathbb{IN}_{I,\max}^1 \) of program \( P_1 \). In such a case the possibility of timing anomalies as defined in Definition 3.1 implies that also a timing anomaly as defined in Definition 3.2 is possible.

### 3.2.1 Visualization of Series Timing Anomalies

Examples for both types of series timing anomalies are given in Figure 5. As described in Section 2.3.1, we again have two subsequent instruction sequences \( A \) and \( B \) for which we want to derive the overall WCET. On the left wide there is the instruction timing of \( A \) given as bold line and the overall timing of \( A \circ B \) given as rectangle.

In the upper two rows (initial states \( s_3 \) and \( s_4 \)) we see the case where a rather small change in \( A \) results in a higher overall change of instruction timing. This is an example for the series timing anomaly **TA-S-A** (amplification).

**Figure 5. Examples for Both Types of Series Timing Anomalies**
In the upper two rows (initial states $s_1$ and $s_2$) we see the case where a change in $A$ results in a change in the inverse direction for the overall timing of $A\circ B$. This is an example for the series timing anomaly TA-S-I (inversion).

### 3.3 Parallel Timing Anomalies

Recently it was found that timing anomalies do not only occur between the timing of two subsequent instruction sequences, but also between the total latency of processor components and the total execution time [12]. We call these timing anomalies “parallel timing anomalies” because they are challenging for the parallel decomposition described in Section 2.3.2.

Parallel timing anomalies are formally defined by Definition 3.3. With parallel timing anomalies we discuss the changes of one instruction sequence, in contrast to the two instruction sequences used for series timing anomalies. What is new with parallel timing anomalies that we split the state TRDCS into two fractions $A$ and $B$. The timing anomalies are defined over the total latency $T_{hw_A}(I, s)$ and the total execution time $T(I, s)$ of an instruction sequence $I$. Total latency is explained in Section 2.2.

**Definition 3.3 (Parallel Timing Anomalies)** Given a partitioned TRDCS $S = A \cup B$ with the timing behavior (total latency) of hardware component $hw_A$ modeled as $T_{hw_A}(I, a)$, the timing behavior $T(I, \langle a, b \rangle)$ of an instruction sequence $I$ on a processor is called a parallel timing anomaly, iff at least one of the following two properties holds:

- **TA-P-I** “Parallel Inversion” (negation of Equation 17):
  \[ \exists a, a' \in A, b \in B. \quad \Delta_{hw_A}(I, a, a') > 0 \land \Delta(I, \langle a, b \rangle, \langle a', b \rangle) < 0 \]

- **TA-P-A** “Parallel Amplification” (negation of Equation 13):
  \[ \exists a, a' \in A, \exists b \in B. \quad 0 < \Delta_{hw_A}(I, a, a') < \Delta(I, \langle a, b \rangle, \langle a', b \rangle) \]

The parallel timing anomaly TA-P-I states that a change of the total latency of instruction sequence $I$ for the processor component $hw_A$ results in a change in the opposite direction for the execution time over the state $\langle a, b \rangle \in A \times B$. Due to this behavior TA-P-I is also called “parallel inversion”.

Analog to the series amplification, the parallel timing anomaly TA-P-A states that a change of the total latency of instruction sequence $I$ for the processor component $hw_A$ results in a larger change in the same direction for the execution time over the state $\langle a, b \rangle \in A \cup B$. Thus, the parallel timing anomaly TA-P-A is also called “parallel amplification”.

In case of parallel timing anomalies, both TA-P-I and TA-P-A can be considered to be “strong (parallel) timing anomalies”, since both potentially invalidate the parallel decomposition described in Section 2.3.2. However, as described in Section 5, not all occurrences of TA-P-I and of TA-P-A are challenging.

Analogously to series timing anomalies, there exists also a more strict definition for parallel timing anomalies than that given in Definition 3.3. This more strict definition of parallel timing anomalies is given in Definition 3.4. These timing anomalies are called worst-case parallel timing anomalies, since they describe exactly the cases that cause problems for efficient WCET analysis with parallel decomposition.

**Definition 3.4 (Worst-Case Parallel Timing Anomalies)** Given a partitioned TRDCS $S = A \cup B$ with the timing behavior (total latency) of hardware component $hw_A$ modeled as $T_{hw_A}(I, a)$, the timing behavior $T(I, \langle a, b \rangle)$ of an instruction sequence $I$ on a processor is called a worst-case parallel timing anomaly, iff at least one of the following two properties holds:
TAW-P-I “Worst-Case Parallel Inversion”
\[
\exists a \in A, b \in B, \forall a' \in A_{\text{max}}, \forall b' \in B_{A, \text{max}}(a').
\]
\[
\Delta_{hw_A}(I, a, a') > 0 \land \Delta(I, \langle a, b \rangle, \langle a', b' \rangle) < 0
\]

TAW-P-A “Worst-Case Parallel Amplification”
\[
\exists a \in A, b \in B, \forall a' \in A_{\text{min}}, \forall b' \in B_{A, \text{max}}(a').
\]
\[
0 < \Delta_{hw_A}(I, a', a) < \Delta(I, \langle a', b' \rangle, \langle a, b \rangle)
\]

with
\[
A_{\text{min}} = \{ a \in A | \forall a' \in A. T_{hw_A}(I, a') \geq T_{hw_A}(I, a) \}
\]
\[
A_{\text{max}} = \{ a \in A | \forall a' \in A. T_{hw_A}(I, a') \leq T_{hw_A}(I, a) \}
\]
\[
B_{A, \text{max}}(a) = \{ b \in B | \forall b' \in B. T(I, \langle a, b \rangle) \geq T(I, \langle a, b' \rangle) \}
\]

Note that the definition given in Definition 3.4 is almost identical with the definition given in Definition 3.3, with the small difference that it uses \( \forall a' \in A_{\text{max}}, \forall b' \in B_{A, \text{max}}(a') \) respectively \( \forall a' \in A_{\text{min}}, \forall b' \in B_{A, \text{max}}(a') \) instead of \( \exists a \in A \). The worst-case timing anomalies are more specific than the others, which is, besides the specific elements to compare, due to the \( \forall \) quantifier instead the \( \exists \) quantifier. Analogous to series timing anomalies, the generic form of timing anomalies given in Definition 3.3 can imply for a specific program (with the right set of reachable states \( \mathbb{N}_I \)) the occurrence of the worst-case timing anomalies given in Definition 3.4.

3.3.1 Visualization of Parallel Timing Anomalies

To visualize parallel timing anomalies we assume that the TRDCS is partitioned into \( A \) and \( B \) as explained in Section 2.3.2. The total latency of instruction sequence \( I \) on hardware component \( hw_A \) is assumed to be as shown in Figure 6.a. It is easier to identify the occurrence of parallel timing anomalies if the total latencies for the different states \( a_i \) are in a (decreasing) order. To get a decreasing order we relabel the states \( a_i \) into states \( a_i' \) as shown in Figure 6.b. We have to compare decreasing total latency with the overall execution time to find occurrences of parallel timing anomalies.

![Figure 6. Total Latency of Processor Component \( hw_A \) for Instruction Sequence \( I \)](image-url)
Figure 7.a shows how the execution time $T(I, \langle a'_i, b \rangle)$ has to look like in case there are no parallel timing anomalies: for all $b \in B$ the execution times are decreasing like the total latency does. The change of execution time is not larger than the change of the total latency. However, the only exception are those cases where the total latency does not change. Whatever the change of the execution time is, as long as the total latency does not change, it is not considered to be a timing anomaly. As described in Section 5, such cases can be handled by doing the parallel composition for multiple total latencies of $hw_A$.

Figure 7.b shows an example of timing anomaly $TA-P-I$ (parallel inversion): the change from state $a'_3$ to state $a'_4$ where the execution time increases while the total latency decreases. Note that between state $a'_2$ and state $a'_3$ there is no timing anomaly, though the execution time also increases. This is not a timing anomaly because in this case the total latency of $hw_A$ does not change.

Figure 7.c shows examples of timing anomaly $TA-P-A$: between states $a'_0$ and $a'_1$ and between states $a'_3$ and $a'_4$. In those cases the execution time decreases more than the total latency of $hw_A$ does.

Of course, it can also happen that both parallel timing anomalies, $TA-P-I$ and $TA-P-A$, occur. As described in Section 5, such a scenario in general does not invalidate parallel decomposition. But it turns problematic when $TA-P-I$ and $TA-P-A$ do occur for the same $b \in B$. The limitations of parallel
decomposition in case of such a scenario are described in Section 5.3.1. Figure 7.d shows such a scenario: the execution time of state $\langle a'_3, b \rangle$ and state $\langle a'_4, b \rangle$ are not bounded by the changes of the total latency of $hw_A$.

### 3.4 Examples of Timing Anomalies

In the previous section we have shown how a processor behaves in case of timing anomalies. In this section we show examples of concrete hardware patterns that can cause such timing anomalies. It is not fully understood how to determine efficiently whether a hardware exhibits timing anomalies. The following presents known instances of timing anomalies, which might help to identify further sources of timing anomalies.

One of the first known potential sources of timing anomalies is out-of-order execution. Wenzel et al. has constructed simple patterns of hardware architectures and studied whether they may exhibit timing anomalies [28, 29]. Figure 8 shows a simple example of inversion timing anomaly, which has been taken from [29]. The assumed processor has an out-of-order pipeline with two non-overlapping resources. Non-overlapping resources means that there are no instructions that can choose from more than one alternatives during each resource allocation. The bold arrows show data dependencies, which restrict the set of different possible executions through the pipeline. The timing anomalies in this example show up due to the combined effect of data dependencies and the out-of-order execution. Figure 9 shows for the same processor model an example of amplification timing anomaly which has been also taken from [29]. Both examples of timing anomalies can manifest as series timing anomalies or as parallel timing anomalies.

---

**Figure 8. Example of TA-S-I and TA-P-I (out-of-order pipeline + cache + data dependencies)**

**Figure 9. Example of TA-S-A and TA-P-A (out-of-order pipeline + cache + data dependencies)**
Despite original belief, the occurrence of timing anomalies is not restricted to out-of-order execution. Wenzel et al. have constructed a simple processor model with in-order execution that exhibits timing anomalies [28, 29]. The assumed processor has an in-order pipeline two partially overlapping resources. Partially overlapping resources means that there are instructions that can choose from more than more than one alternative during a resource allocation. Figure 10 shows an example of inversion timing anomaly. The timing anomalies in this example show up due to the combined effect of partially overlapping resources and caching. The partial resource overlapping happens for instruction $i_4$, which is the only instruction in the example that can choose from both resources ($hw_A$ and $hw_B$). With the given hardware partitioning we consider this as a relevant example mostly for series timing anomalies. It might be not that important as parallel timing anomalies, because a parallel decomposition into overlapping resources is not that beneficial for a first independent analysis of the total latency of hardware component $hw_A$.

Another interesting cause for timing anomalies is speculation. Reineke et al. have shown an example where speculation can cause timing anomalies [22]. The principle is shown in Figure 11, where the interference of speculation on the instruction cache causes timing anomalies. The vertical line shows the time instance where the condition branch instruction $i_1$ has been evaluated, with the result that the speculation is withdrawn and execution continues with instruction $i_2$.

It is important to mention that even if the processor itself is very simple and time-predictable, its
combination with other components can still cause timing anomalies. For example, Berg has shown that a cache with pseudo-round robin (PRR) replacement can cause timing anomalies [2]. The effect of this example is shown in Figure 12, where a loop consisting of instruction $i_1 \ldots i_7$ is executed multiple times. For two different initial states $s_1$ and $s_2$ the iteration finally reaches stabilized iteration with a constant difference of one cache miss during each iteration. The detailed description of the example can be found at [2]. Interestingly, the given scenario is also an example of so-called domino effects (see Section 3.1). With the concrete hardware partitioning this example is only an instance of “weak series timing anomaly” (TA-S-A), but not a parallel timing anomaly.

From above examples we see that timing anomalies can occur on hardware of quite different complexity. Further research is required to develop efficient checks whether a concrete hardware design exhibits timing anomalies.

### 3.5 A Sufficient Precondition for Timing Anomalies

It would be useful to have a criterion that allows to determine whether a concrete processor can exhibit timing anomalies. Wenzel et al. provided a necessary precondition for the potential occurrence of timing anomalies, the resource allocation criterion [28, 29]. However, this precondition has been shown to miss practical applicability. As noted by Reineke et al., the resource allocation criterion is based on a rather restricted definition of timing anomalies, which does not cover all known examples of timing anomalies [22]. Further, the criterion itself focused on functional units of pipelines. But as demonstrated above, even simple processors without pipelines can exhibit timing anomalies.

However, as the resource allocation criterion does not strongly depend on the initially restrictive assumption, we reformulate the resource allocation criterion in the light of the new findings presented in this article. This reformulation reflects the differentiation between serial and parallel timing anomalies and is based on the initial states of instruction sequences.

To present the new precondition for timing anomalies we first have to introduce some additional symbols:

- $\mathbb{R}$ ... the set of all resources of the target hardware.
- $RA(s, I, R, n)$ ... a function that calculates the concrete instruction $i \in I$ that performs the $n$-th allocation of resource $R$ during the execution of instruction sequence $I$ with initial state $s$. A resource
\( R \) can have at most \(|I|\) allocations, i.e., for each \( n \geq |I| \) the function \( RA(s, I, R, n) \) is always false (\( n \) starting with zero). The function \( RA \) actually describes a partial order \( <_{s,R,I} \) over the order of resource allocations:

\[
\begin{align*}
  i_1 <_{s,R,I} i_2 & \iff \exists n_1, n_2. \ i_1 = RA(s, I, R, n_1) \land \\
  & \land i_2 = RA(s, I, R, n_2) \land n_1 < n_2
\end{align*}
\]  

(1)

To define a precondition for timing anomalies we have to express that there happen resource allocations during runtime. A dynamic resource allocation is expressed by the predicate \( ResAlloc \) given in Equation 2, which states that a change of the initial state at the beginning of instruction sequence \( I \) from \( s_1 \) to \( s_2 \) causes a change from a allocation position of resource \( R_1 \) to another allocation position of the resource \( R_2 \). The allocation position \( n \) represents the \( n \)-th allocation of a resource by instructions during the execution of instruction sequence \( I \).

\[
ResAlloc(s_1, s_2, I, i, R_1, R_2) = \exists n_1, n_2. \ i = RA(s_1, I, R_1, n_1) \land \\
  i = RA(s_2, I, R_2, n_2) \land (n_1 \neq n_2 \lor R_1 \neq R_2)
\]  

(2)

Based on that definition, a necessary but not sufficient precondition for the occurrence of timing anomalies is given in Theorem 3.5.

**Theorem 3.5 Precondition for Timing Anomalies:** When changing the initial state at the beginning of an instruction sequence \( I \) from a state \( s_1 \) to another state \( s_2 \) the following is a necessary but not sufficient precondion for the occurrence of timing anomalies:

\[
\exists I, \exists s_1, s_2 \in S, \exists R_1, R_2 \in R, \exists i \in I. \\
ResAlloc(s_1, s_2, I, i, R_1, R_2)
\]  

(3)

Above condition states that it is possible when the initial state at the beginning of an instruction sequence \( I \) is changed from a state \( s_1 \) to another state \( s_2 \) that the allocation sequence for at least one resource changes. The allocation sequence is a list that represents the temporal order in which the resource was allocated by different instructions.

Corollary 3.6 gives a specialization of the precondition for the occurrence of series timing anomalies.

\[
SCondSerTA(s_1, s_2, I) = \exists C \neq \emptyset, \exists D \neq \emptyset. \\
I = C \circ D \land T(C, s_1) \neq T(C, s_2)
\]  

(4)

**Corollary 3.6 Precondition for Series Timing Anomalies:** The following precondition is necessary but not sufficient for the occurrence of parallel timing anomalies:

\[
\exists I, \exists s_1, s_2 \in S, \exists R_1, R_2 \in R, \exists i \in I. \\
ResAlloc(s_1, s_2, I, i, R_1, R_2) \land \\
SCondSerTA(s_1, s_2, I)
\]  

(5)
Above condition is the combination of the general precondition for timing anomalies given in Theorem 3.5 and Equation 4, which is a subcondition used for the definition of series timing anomalies in Definition 3.1. Thus, the condition is a necessary but not sufficient precondition for the occurrence of series timing anomalies.

Corollary 3.7 gives a specialization of the precondition for the occurrence of parallel timing anomalies.

\[
SCondParTA(s_1, s_2, I) = \exists b \in B. s_1 = \langle a_1, b \rangle \land s_2 = \langle a_2, b \rangle \land T_{hwA}(I, s_1) \neq T_{hwA}(I, s_2)
\]

Corollary 3.7 Precondition for Parallel Timing Anomalies: The following precondition is necessary but not sufficient for the occurrence of parallel timing anomalies:

\[
\exists I, \exists s_1, s_2 \in S, \exists R_1, R_2 \in \mathbb{R}, \forall i \in I.
\ ResAlloc(s_1, s_2, I, i, R_1, R_2) \land SCondParTA(s_1, s_2, I)
\]

Above condition is the combination of the general precondition for timing anomalies given in Theorem 3.5 and Equation 6, which is a subcondition used for the definition of parallel timing anomalies in Definition 3.3. Thus, the condition is a necessary but not sufficient precondition for the occurrence of series timing anomalies.

In this section we reformulated the resource allocation criterion as a necessary but not sufficient precondition for the occurrence of timing anomalies. It is an interesting challenge to find additional concrete architectural patterns of hardware that narrows the criterion so that it becomes possible to check whether a concrete hardware exhibits timing anomalies. From the examples in Section 3.4 we have seen that different hardware properties can cause timing anomalies. For example, out-of-order execution in combination with data dependencies can cause timing anomalies even with resources of non-overlapping capabilities. In case of resources with partially overlapping capabilities even in-order execution can cause timing anomalies. And even a cache itself (on a simple non-pipelined processor) can exhibit timing anomalies, for example with PLRU replacement policy.

4 WCET Analysis with Series-Composition

In this section we describe a concrete WCET analysis method based on series decomposition (see Section 2.3.1) and show why timing anomaly TA-S-I invalidates the method and why timing anomaly TA-S-A does not invalidate it.

We define a method, which we call Series-Composition, to derive the maximum overall instruction timing for two successive instruction sequences \( M \) and \( N \). Equation 8 defines how the instruction timing \( T_{sc}(M \circ N) \) is calculated with Series-Composition. This is actually a prototypical definition, but more elaborated approach may still face the same challenges with timing anomalies.

\[
T_{sc}(M \circ N) = \max_{s \in IN, \max} T(M \circ N, s)
\]

Theorem 4.1 describes the sufficient and necessary condition about the hardware behavior such that Series-Composition \( T_{sc}(M \circ N, s) \) is safe, i.e., that it provides an upper bound for the execution time of instruction sequence \( M \circ N \).
Theorem 4.1 Safeness of Series-Composition: Assuming that the program to be analyzed is decomposed into connected control-flow nodes and the set of possible subpaths (= sequences of control-flow nodes) of the program is denoted by the set $\mathcal{SP}$, then Series-Composition allows to provide a safe WCET bound on processor hardware whose timing characteristics obey the following sufficient and necessary condition: 

\begin{align*}
\forall M \circ N \in \mathcal{SP}, \forall s_1 \in \mathbb{IN}_M, \exists s_2 \in \mathbb{IN}_{M,\text{max}}. & \quad \Delta(M, s_1, s_2) > 0 \rightarrow \\
& \quad \Delta(M \circ N, s_1, s_2) \geq 0
\end{align*}

(9)

Condition 9 states that whenever the state at the beginning of instruction sequence $M$ changes from a state $s_1 \notin \mathbb{IN}_{M,\text{max}}$ to a different state $s_2 \in \mathbb{IN}_{M,\text{max}}$ (where the execution time of instruction-sequence $M$ is maximal and thus the resulting change in the execution time of instruction-sequence $M$ is greater than zero: $\Delta(M, s_1, s_2) > 0$), then for the composed instruction sequence $M \circ N$ the change of the execution time has to be greater or equal to zero ($\Delta(M \circ N, s_1, s_2) \geq 0$).

The correctness condition given in Equation 9 is the negation of $\text{TAW-S-I}$ (see Definition 3.1).

**Corollary 4.2 Sufficient Condition for Series-Composition** The safeness condition of Series-Composition given in Theorem 4.1 is the weakest possible one for a given set of possible input states $\mathbb{IN}_M$. However it may be the case that a stronger condition (more restrictive) is easier to check. The following is a more restrictive sufficient condition for the safeness of Series-Composition:

\begin{align*}
\forall M \circ N \in \mathcal{SP}, \forall s_1, s_2 \in \mathbb{IN}_M. & \quad \Delta(M, s_1, s_2) > 0 \rightarrow \\
& \quad \Delta(M \circ N, s_1, s_2) \geq 0
\end{align*}

(10)

Condition 10 states that whenever the state at the beginning of instruction sequence $M$ changes from a state $s_1$ to a different state $s_2$ and the resulting change in the execution time of instruction-sequence $M$ is greater than zero ($\Delta(M, s_1, s_2) > 0$), then for the composed instruction sequence $M \circ N$ the change of the execution time has to be greater or equal to zero ($\Delta(M \circ N, s_1, s_2) \geq 0$). (proof given in Annex A)

The correctness condition given in Equation 10 is the negation of $\text{TA-S-I}$ (see Definition 3.1).

4.1 Safeness of Series-Composition

The following two theorems state which type of series timing anomaly are a challenge for the correctness of Series-Composition. Their proof is based on the correctness condition given in Equation 10.

**Theorem 4.3 Timing-Composability without $\text{TA-S-I}$:** The absence of timing anomalies of type $\text{TA-S-I}$ on a processor and a given instruction sequence is

a) sufficient for the correctness of Series-Composition,

b) not necessary for the correctness of Series-Composition,

(proof given in Annex A)

**Theorem 4.4 Timing-Composability even with $\text{TA-S-A}$:** The absence of timing anomalies of type $\text{TA-S-A}$ on a processor and a given instruction sequence is not necessary for the correctness of Series-Composition.

(proof given in Annex A)
Table 1. Applicability of Series Composition

<table>
<thead>
<tr>
<th>Composition Technique</th>
<th>Timing Anomaly</th>
</tr>
</thead>
<tbody>
<tr>
<td>SC</td>
<td>TA-S-I</td>
</tr>
<tr>
<td>Full State</td>
<td>TA-S-A</td>
</tr>
</tbody>
</table>

4.2 Summary of Series-Composition

Table 1 summarizes the situation where Series-Composition is safe. This is actually the case if no timing anomaly is present or if only timing anomalies of type TA-S-A are present. Timing anomalies of type TA-S-I invalidate the correctness of Series-Composition. “Full Path” means the examination of the complete instruction sequence at once, which is not a composition method anymore. It is just given for completeness, as it is the most precise and safe method in case of TA-S-I, though overly costly in practice.

5 WCET Analysis with Parallel Composition

In Section 2.3.2 we described the basic idea of reducing analysis complexity by using parallel decomposition of a state TRDCS into two sets A and B. The challenge is to find a composable timing calculation method that can be used to calculate safe WCET bounds for the target processor of interest. In the following we describe two different timing-composition techniques and analyze their correctness in case of parallel timing anomalies.

5.1 Delta-Composition

The first prototypical technique to derive the maximum overall instruction timing of an instruction sequence I based on a decomposition of the TRDCS into two state fractions A and B is called Delta-Composition. The principle of Delta-Composition is given in Figure 13:

1. \( \Delta_{hw,A,max} \), the maximum variability (\( \Delta_{hw,A} \)) of \( T_{hw,A}(I,a) \) is determined:
   \[
   \Delta_{hw,A,max} = \max_{a,a' \in A} |T_{hw,A}(I,a) - T_{hw,A}(I,a')|
   \]

2. The set \( A_{min} \) of local states \( a_{hw,A,min} \in A \) where \( T_{hw,A}(I,a) \) is minimal, is determined:
   \[
   A_{min} = \{ a \in A \mid \forall a' \in A. T_{hw,A}(I,a') \geq T_{hw,A}(I,a) \}.
   \]

3. For each \( a_{hw,A,min} \in A_{min} \) the overall timing function \( T(I, \langle a_{hw,A,min}, b \rangle) \) with fixed local state \( a_{hw,A,min} \) is selected.

4. \( b_{dc,max} \), the partial state \( b \in B \) where \( T(I, \langle a_{hw,A,min}, b \rangle) \) is maximal, is determined:
   \[
   b \in B_{A,max}(a_{hw,A,min}) \text{ with } B_{A,max}(a) = \{ b \in B \mid \forall b' \in B. T(I, \langle a, b \rangle) \geq T(I, \langle a, b' \rangle) \}.
   \]

5. \( \Delta_{hw,A,max} \) is added to \( T(I, \langle a_{hw,A,min}, b_{dc,max} \rangle) \).

Equation 11 shows how the maximum instruction timing is calculated with Delta-Composition (\( T_{dc}(I) \)). Note that the Delta-Composition potentially overestimates the WCET: \( T_{dc}(I) \geq T_{max}(I) \).

\[
T_{dc}(I) = \max_{a \in A_{min}, b \in B} T(I, \langle a, b \rangle) + \Delta_{hw,A,max}
\]
In case there are multiple states $a \in A_{\text{min}}$ of minimal latency then Delta-Composition evaluates each of these minima and takes the overall maxima. The computational cost for $T_{dc}(I)$ is $O((|A| + |A_{\text{min}}| \cdot |B|)) \cdot |I|$. Thus, the more minima $a \in A_{\text{min}}$ exist, the higher is the computational cost of Delta-Composition. In the extreme case of $A = A_{\text{min}}$ the Delta-Composition degrades to searching all states $s \in A \times B$. However, this extreme case of $A = A_{\text{min}}$ rarely seems to be a real problem, because in that case it has no influence on the timing and thus cannot be part of the TRDCS. However the worst case of complexity is the case $|A| - 1 = |A_{\text{min}}|$.

Theorem 5.1 describes the sufficient and necessary condition about the hardware behavior such that Delta-Composition ($T_{dc}(I, s)$) is safe, i.e., that it provides an upper bound for the execution time of an instruction sequence $I$.

**Theorem 5.1 Safeness of Delta-Composition:** Based on above definitions of $A_{\text{min}}$, $B_{A,\text{max}}(a)$, and $\Delta_{hw,A,\text{max}}$, the Delta-Composition allows to provide a safe WCET bound on processor hardware whose timing characteristics obey the following sufficient and necessary condition (proof given in Annex A):

$$\forall a \in A, \forall b \in B, \exists a' \in A_{\text{min}}, \exists b' \in B_{A,\text{max}}(a').$$

$$\Delta_{hw,A}(I, a', a) > 0 \rightarrow \Delta(I, \langle a', b' \rangle, \langle a, b \rangle) \leq \Delta_{hw,A,\text{max}}$$

(12)

Condition 12 states that there exists at least one local best-case state $a' \in A_{\text{min}}$ ($a_{hw,A,\text{min}}$) such that whenever the state of a hardware component $hw_A$ changes from any state $a \notin A_{\text{min}}$ to this specific best-case state $a'$, the resulting change in the execution time of instruction-sequence $I$ ($\Delta(I, \langle a', b' \rangle, \langle a, b \rangle)$) is not higher than the maximum change that is possible in the total latency of the hardware component $hw_A$ ($\Delta_{hw,A}(I, a', a)$).

The correctness condition given in Equation 12 is the negation of $\text{TAW-P-A}$ (see Definition 3.3).

**Corollary 5.2 Sufficient Condition for Delta-Composition** The safeness condition of Delta-Composition given in Theorem 5.1 is the weakest possible one. However it may be the case that a stronger
condition (more restrictive) is easier to check. The following is a more restrictive sufficient condition for the safeness of Delta-Composition (proof given in Annex A):

\[ \forall a, a' \in A, b \in B. \quad \Delta_{hw}(I, a, a') > 0 \rightarrow \Delta_{hw}(I, a, a') \geq \Delta(I, \langle a, b \rangle, \langle a', b \rangle) \] (13)

The correctness condition given in Equation 13 is the negation of TA-P-A (see Definition 3.3).

5.2 Max-Composition

The second prototypical technique to derive the maximum overall instruction timing of an instruction sequence \( I \) based on a decomposition of the TRDCS into two state fractions \( A \) and \( B \) is called Max-Composition. The principle of Max-Composition is given in Figure 14:

1. The set \( A_{\text{max}} \) of local states \( a_{hw_{A_{\text{max}}}} \in A \) where \( T_{hw_{A}}(I, a) \) is maximal, is determined:
   \[ A_{\text{max}} = \{ a \in A \mid \forall a' \in A. T_{hw_{A}}(I, a') \leq T_{hw_{A}}(I, a) \} \].
2. For each \( a_{hw_{A_{\text{max}}}} \in A_{\text{max}} \) the overall timing function \( T(I, \langle a_{hw_{A_{\text{max}}}}, b \rangle) \) with fixed local state \( a_{hw_{A_{\text{max}}}} \) is selected.
3. \( b_{mc_{\text{max}}} \), the partial state \( b \in B \) where \( T(I, \langle a_{hw_{A_{\text{max}}}}, b \rangle) \) is maximal, is determined:
   \[ b_{\in B_{\text{max}}(a_{hw_{A_{\text{max}}}})} \text{ with } B_{\text{max}}(a) = \{ b \in B \mid \forall b' \in B. T(I, \langle a, b \rangle) \geq T(I, \langle a', b' \rangle) \}. \]

![Figure 14. Max-Composition of TRDCS](image)

Equation 14 shows how the maximum instruction timing is calculated with Max-Composition \( T_{mc}(I) \). Max-Composition provides a precise WCET bound: \( T_{mc}(I) = T_{\max}(I) \).

\[ T_{mc}(I) = \max_{a \in A_{\text{max}}, b \in B} T(I, \langle a, b \rangle) \] (14)
In case there are multiple states \( a \in A_{\text{max}} \) of maximal latency then Max-Composition evaluates each of these maxima and takes the overall maxima. The computational cost for \( T_{mc}(I) \) is \( O((|A| + |A_{\text{max}}| \cdot |B|) \cdot |I|) \). Thus, the more maxima \( a \in A_{\text{max}} \) exist, the higher is the computational cost of Max-Composition. In the extreme case of \( A = A_{\text{max}} \) the Max-Composition degrades to searching all states \( s \in A \times B \). However, this extreme case of \( A = A_{\text{max}} \) rarely seems to be a real problem. Because in the case \( A = A_{\text{max}} \) it is typically the case that the total latency of hardware component \( hw_B \) is independent of the total latency of hardware component \( hw_A \), and hence the state space \( A \) is not part of the TRDCS.

Theorem 5.3 describes the sufficient and necessary condition about the hardware behavior such that Max-Composition \( (T_{mc}(I, s)) \) is safe, i.e., that it provides an upper bound for the execution time of an instruction sequence \( I \).

**Theorem 5.3 Safeness of Max-Composition:**

Based on above definition of \( A_{\text{max}} \) the Max-Composition allows to provide a safe WCET bound on processor hardware whose timing characteristics obey the following sufficient and necessary condition (proof given in Annex A):

\[
\forall a \in A, \forall b \in B, \exists a' \in A_{\text{max}}, \exists b' \in B_{A_{\text{max}}}(a').
\]

\[
\Delta_{hw_A}(I, a, a') > 0 \rightarrow \Delta(I, \langle a, b \rangle, \langle a', b' \rangle) \geq 0
\]  

(15)

Condition 15 states that there exists at least one local worst-case state \( a' \in A_{\text{max}} (a_{hw_A,\text{max}}) \) such that whenever the state of a hardware component \( hw_A \) changes from any state \( a \notin A_{\text{max}} \) to this specific worst-case state \( a' \), then the execution time of instruction sequence \( I \) must not decrease when also changing the state of hardware component \( hw_B \) from any state \( b \in B \) to one of the states \( b' \in B_{A_{\text{max}}}(a') \) \( (\Delta(I, \langle a, b \rangle, \langle a', b' \rangle) \geq 0) \).

The correctness condition given in Equation 15 is the negation of \( \text{TAW-P-I} \) (see Definition 3.3).

**Corollary 5.4 Sufficient Condition for Max-Composition** The safeness condition of Max-Composition given in Theorem 5.3 is the weakest possible one. However it may be the case that a stronger condition (more restrictive) is easier to check. The following is a more restrictive sufficient condition for the safeness of Max-Composition:

\[
\forall a \in A, \forall b \in B, \exists a' \in A_{\text{max}}. \quad \Delta_{hw_A}(I, a, a') > 0 \rightarrow \Delta(I, \langle a, b \rangle, \langle a', b \rangle) \geq 0
\]  

(16)

And the following Condition 17 is a sufficient condition for the safeness of Max-Composition even more restrictive than Condition 16:

\[
\forall a, a' \in A, b \in B. \quad \Delta_{hw_A}(I, a, a') > 0 \rightarrow \Delta(I, \langle a, b \rangle, \langle a', b \rangle) \geq 0
\]  

(17)

(proof given in Annex A)

The correctness condition given in Equation 17 is the negation of \( \text{TA-P-I} \) (see Definition 3.3).

### 5.3 Safeness of Parallel Composition

The following two theorems state which type of parallel timing anomaly are a challenge for the correctness of Delta-Composition and Max-Composition. Their proof is based on the correctness conditions given in Equation 13 and Equation 16.
Theorem 5.5 Timing-Composability without TA-P-I: The absence of timing anomalies of type TA-P-I on a processor and a given instruction sequence is

a) sufficient for the correctness of Max-Composition,

b) not necessary for the correctness of Max-Composition,

c) not sufficient for the correctness of Delta-Composition.

(proof given in Annex A)

Theorem 5.6 Timing-Composability without TA-P-A: The absence of timing anomalies of type TA-P-A on a processor and a given instruction sequence is

a) sufficient for the correctness of Delta-Composition,

b) not necessary for the correctness of Delta-Composition,

c) not sufficient for the correctness of Max-Composition.

(proof given in Annex A)

Corollary 5.7 Concluding from Theorem 5.5 and Theorem 5.6, processor hardware exhibiting timing anomalies of at most one of the types TA-P-I or TA-P-A, without knowing which one it is, can safely be analyzed by applying both, Delta-Composition (Equation 11) and Max-Composition (Equation 14) simultaneously:

\[ T_{dmc}(I) = \max(T_{dc}(I), T_{mc}(I)) \]  

(18)

From Corollary 5.7 it follows that parallel timing anomalies are not a serious problem as long as only one type of them occurs. Note that since the Delta-Composition is not tight, \( T_{dmc}(I) \) also does not have to be tight: \( T_{dmc}(I) \geq T_{\text{max}}(I) \).

5.3.1 Coupled Parallel Timing Anomaly

In the previous section we have shown that parallel composition can be safe if at most one type of parallel timing anomalies is possible.

In the following we analyze in more detail what happens if both types of parallel timing anomalies (TA-P-I and TA-P-A) occur. In this case we can differ between the case where both types of parallel timing anomalies occur only for different states \( b \in B \) and \( b' \in B \) (discussed in Section 5.3.2) and the more severe case where they also occur for the same state \( b \in B \). The latter case is discussed in the following.

A formal definition of the case where both types of timing anomalies TA-P-I and TA-P-A occur for the same state \( b \in B \) is given in Definition 5.8. To simplify its reference, we have named this case as TA-P-C where “C” stands for the coupled (same \( b \in B \)) of both parallel timing anomalies.

Definition 5.8 (TA-P-C: Coupled Parallel Timing Anomaly) Given a partitioned TRDCS \( S = A \cup B \) with the total latency of hardware component \( h_{wA} \) modeled as \( h_{wA}(I, a) \), the timing behavior \( T(I, (a, b)) \) of an instruction sequence \( I \) on a processor is called a coupled parallel timing anomaly, iff the following property holds:

\[ \exists a_1, a_2, a_3, a_4 \in A, \exists b \in B. \]

\[ (\Delta_{hwA}(I, a_1, a_2) > 0 \land \Delta(I, (a_1, b), (a_2, b)) < 0) \land \]

\[ (0 < \Delta_{hwA}(I, a_3, a_4) < \Delta(I, (a_3, b), (a_4, b))) \]  

(19)
Above definition combines the definitions of TA-P-I and TA-P-A given in Definition 3.3. The word “coupled” signals that both types of timing anomalies occur for the same state \( b \in B \).

Theorem 5.9 states that timing anomalies of type TA-P-C can only be bounded by searching the whole state space \( A \cup B \). This is actually an impossibility result for applying efficient parallel composition whenever the occurrence of TA-P-C is possible.

**Theorem 5.9 Non-Composability with Coupled Parallel Timing Anomalies:** On processor hardware exhibiting parallel timing anomalies of type TA-P-C as defined in Definition 5.8 with a state partitioning into two partitions \( A \) and \( B \), there are no safe parallel composition techniques without analyzing the whole combined state space \( A \times B \).

**Proof of Theorem 5.9 (Non-Composability with Coupled Parallel Timing Anomalies TA-P-C):**
The critical aspect leading to the non-composability is that both types of timing anomalies can show up for the same state \( b \in B \). Thus, from Theorem 5.5.c: we get that Delta-Composition is not a safe composition method. And from Theorem 5.6.c: we get that also Max-Composition is not a safe composition method. That there can be no other safe composition method that does not need to analyze the whole combined state space \( A \times B \) follows from Figure 7.d. As shown in the figure, the values of \( T(I,(a,b)) \) can grow to arbitrary height independently of the values of \( T_{hw,A}(I,a) \). As this growth may take place at any point \( (a,b) \in A \times B \), the only way to bound this effect is to search for it over the whole state space \( A \times B \).

### 5.3.2 Exclusive Parallel Timing Anomaly

We call the case where the two types of parallel timing anomalies (TA-P-I and TA-P-A) occur only for different states \( b \in B \) and \( b' \in B \) exclusive parallel timing anomaly, which is formally defined in Definition 5.10. To simplify its reference, we have named this case as TA-P-E where “E” stands for the exclusive occurrence of either TA-P-I or TA-P-A.

**Definition 5.10 (TA-P-E: Exclusive Parallel Timing Anomaly)** Given a partitioned TRDCS \( S = A \cup B \) with the total latency of hardware component \( hw_A \) modeled as \( T_{hw_A}(I,a) \), the timing behavior \( T(I,(a,b)) \) of an instruction sequence \( I \) on a processor is called an exclusive parallel timing anomaly, iff the following property holds:

\[
\exists a_1, a_2, a_3, a_4 \in A, \exists b_1, b_2 \in B. \quad (b_1 \neq b_2) \land
\left( \Delta_{hw_A}(I,a_1,a_2) > 0 \land \Delta(I,(a_1,b_1),(a_2,b_1)) < 0 \right) \land
\left( 0 < \Delta_{hw_A}(I,a_3,a_4) < \Delta(I,(a_3,b_2),(a_4,b_2)) \right)
\]

Above definition allows the occurrence of both, TA-P-I and TA-P-A as given in Definition 3.3. But the word “exclusive” signals that the two types of timing anomalies can only occur for different states \( b_1, b_2 \in B \).

Theorem 5.11 states that timing anomalies of type TA-P-E can efficiently be bounded without having to search the whole state space \( A \times B \). This is the most generic form of occurrence of parallel timing anomalies that can efficiently be bounded. As Theorem 5.9 states, this is not possible in a more generic form.
<table>
<thead>
<tr>
<th>Composition Technique</th>
<th>Timing Anomaly</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>none</td>
</tr>
<tr>
<td>MC</td>
<td>×</td>
</tr>
<tr>
<td>DC</td>
<td>×</td>
</tr>
<tr>
<td>max(MC,DC)</td>
<td>×</td>
</tr>
<tr>
<td>Full State</td>
<td>×</td>
</tr>
</tbody>
</table>

Table 2. Applicability of Parallel Composition

Theorem 5.11 Composability with Exclusive Parallel Timing Anomalies: Processor hardware exhibiting parallel timing anomalies of type TA-P-E as defined in Definition 5.10 with a state partitioning into two partitions A and B, can safely be analyzed by applying both, Delta-Composition (Equation 11) and Max-Composition (Equation 14) simultaneously as described in Equation 18.

Proof of Theorem 5.11 (Composability with Exclusive Parallel Timing Anomalies TA-P-E): As timing anomalies of type TA-P-I and TA-P-A can only occur for different states \( b_1, b_2 \in B \), for any \( b \in B \) either Max-Composition or Delta-Composition is correct. Thus, the composability result of Corollary 5.11 applies also for the occurrence of TA-P-E.

5.4 Summary of Parallel Composition

Table 2 summarizes the situation where Parallel-Composition is safe. Max-Composition (MC) is safe if at most parallel timing anomalies of type TA-P-A are present and Delta-Composition (DC) is safe if at most parallel timing anomalies of type TA-P-I are present. If TA-P-I and TA-P-A can both occur, but only for different states \( b \in B \) and \( b' \in B \) (scenario TA-P-E) then the maximum of Max-Composition and Delta-Composition is a safe upper bound of the execution time. But if TA-P-I and TA-P-A can both occur for the same state \( b \in B \) (scenario TA-P-C) then there is no efficient method that does not rely on examining the combined state space of A and B. The examination of the combined state space (“Full State”) is not a composition method anymore, but is given to show the consequences in case of TA-P-C.

6 Implications on Static WCET Analysis

It was rather easy to upper-bound the instruction timing for processors like the Motoral M68000 processor, where the instruction timing does rarely depend on the processor’s state. Today we are in a situation that even out-of-order pipelining and caches with hardly to predict timing behavior are used in the embedded systems domain [12, 21, 9].

With such modern processors that features caches, complex pipelines, or speculation it becomes increasingly difficult to perform precise and time-efficient WCET analysis. Static WCET analysis with separated cache and pipeline analyzes [5] have been found inadequate for modern processors. Thus, WCET analysis tools with integrated cache and pipeline analysis have been developed [13]. This integrated cache and pipeline analysis already hits the complexity wall of today’s processors. And the results of this paper also confirm that processors showing series timing anomalies of type TA-S-I (Theorem 5.9) or parallel timing anomalies of type TA-P-C (Theorem 4.3) are hard to analyze for their timing behavior.
6.1 Time-Predictable Processors

The complexity of WCET analysis can be reduced by using processors with an improved predictability. The most desired solution would be a processor with high predictability without sacrificing performance.

Anantaraman et al. proposed the VISA architecture, which is a dual mode processor that has one execution mode that is tuned to average performance and another execution mode that is tuned to time-predictability [1]. The time-predictability of VISA requires software-support. Whenever a checkpoint is reached where the time budget is already consumed, the processor is switched per software in a time-predictable mode for which WCET analysis is relatively easy.

Rochange and Sainrat proposed to add features to the processor to delay instruction fetch until it is sure that no late timing effects [4] can occur [23]. Mohan et al. proposed to add features to a processor that allows to read or restore the processor state. With that feature they constructed a WCET analysis that is safe even in case of timing anomalies [17]. Reineke et al. favor LRU-cache replacement policy as being more predictable than pseudo LRU, pseudo round robin, or FIFO [21]. Schöberl discussed several ways to improve the predictability of future processors [26].

However, it is currently still an open question of how to decide which hardware components in combination can enable timing anomalies. The resource allocation criterion described in Section 3.5 provides a sufficient but not necessary criterion for the occurrence of timing anomalies.

6.2 Software Solutions to Improve WCET Analyzability

Working on more predictable hardware is one way to improve WCET-analyzability. Another way that is also applicable for modern hardware is the generation of more predictable code patterns.

Lundqvist and Stenström proposed to insert instructions to sync the pipeline explicitly such that a WCET analysis tool can easily identify the locations where no timing-relevant state-propagation of the pipeline is possible [15]. Rochange and Sainrat proposed to use code padding with NOP-instructions to fill any prefetch window inside the processor [24]. Kadlec et al. work on different code modification techniques to disable potential timing anomalies, having just preliminary results available so fare [10].

The software solutions discussed so far have some potential to improve predictability of today’s processors. However, in the long term it would be desired to have processors being constructed with time-predictability kept in mind.

7 Conclusion and Outlook

The most challenging problem of WCET analysis is the high complexity of today’s processors. Features like caches and pipelines create a huge state space. Even worse, effects like timing anomalies can make it impossible to construct an efficient processor behavior analysis that does not need to search the whole state space for the whole program at once.

In this paper we presented a compact formal definition of timing anomalies and showed that timing anomalies do not only occur when composing instruction sequences, but also when composing the overall state from components. We call the first class of timing anomalies series timing anomalies, and the second class parallel timing anomalies We have introduced the Series-Composition to show to what extend series timing anomalies are a challenge. Further, we introduced the Delta-Composition and Max-Composition to show to what extend parallel timing anomalies are challenging. We have given formal proofs in the appendix to all essential possibility and impossibility results described in the paper.

The main result is that not all types of timing anomalies are a principle problem for efficient processor behavior analysis. Only the types TA-S-I and TA-S-C provide inherent problems. Future work is
needed on identifying the concrete types of timing anomalies that might occur for a concrete processor implementation.

Acknowledgments

We would like to thank Michael Zolda for useful advice on first-order logic.

References


A Proofs

This section provides the proofs for the theorems of the previous chapters. Their study may not be necessary to understand the technical contribution of the paper. However, they are provided to argue about the correctness and to provide a deeper understanding on how we use the formalism to reason about the temporal behavior of components in an abstract way.

The validity of the following first-order logic formulas is used as auxiliary arguments in certain proofs:

\[(\forall x. (P(x) \rightarrow Q(x)) \land \forall x. P(x)) \rightarrow \forall x.Q(x)\]  

(21)

\[(\forall x, \forall y. P(x, y) \leftrightarrow \forall x, \forall y. (P(x, y) \land P(y, x)))\]  

(22)

\[(\exists x, \exists y. P(x, y) \leftrightarrow \exists x, \exists y. (P(x, y) \lor P(y, x)))\]  

(23)

A.1 Proofs for Series Composition

Proof of Theorem 4.1 (Safety of Series-Composition):

a) Showing that the condition is sufficient:

The correctness condition of Theorem 4.1 can be written as

\[\forall M \circ N \in \mathcal{SP}, \forall s_1 \in \mathbb{IN}_M, \exists s_2 \in \mathbb{IN}_{M,max}. T(M, s_1) < T(M, s_2) \rightarrow T(M \circ N, s_1) \leq T(M \circ N, s_2)\]

where \(\mathbb{IN}_{M,max}\) is defined as:

\[\mathbb{IN}_{M,max} = \{s \mid s \in \mathbb{IN}_M \land \forall s' \in \mathbb{IN}_M. T(M, s') \leq T(M, s)\}\]

The Series-Composition is calculated as follows:

\[T_{sc}(M \circ N) = \max_{s \in \mathbb{IN}_{M,max}} T(M \circ N, s)\]

There are two cases for the correctness condition:

- \(s_1 \notin \mathbb{IN}_{M,max}\): here the correctness condition implies that there exists at least one state \(s_2 \in \mathbb{IN}_{M,max}\) which results in a higher execution time for \(M \circ N\) as \(s_1\) will do. Since \(T_{sc}\) computes the maximum of \(T(M \circ N, s)\) for any \(s \in \mathbb{IN}_{M,max}\), it will thus have an upper time bound on the execution time caused by \(s \notin \mathbb{IN}_{M,max}\).

- \(s_1 \in \mathbb{IN}_{M,max}\): here the correctness condition does not imply anything on the execution time of \(T(M \circ N, s_1)\) compared to other states \(s_2 \in \mathbb{IN}_{M,max}\). However, in this case the calculation of \(T_{sc}\) provides an upper time bound since it computes the maximum of \(T(M \circ N, s)\) for any \(s \in \mathbb{IN}_{M,max}\).

This proves that the correctness condition is sufficient.

b) Showing that the condition is necessary:

We use proof by contradiction to show that the correctness condition is necessary. Thus, we assume that the negated condition holds:

\[\exists M \circ N \in \mathcal{SP}, \exists s_1 \in \mathbb{IN}_M, \forall s_2 \in \mathbb{IN}_{M,max}. \neg(T(M, s_1) < T(M, s_2) \rightarrow T(M \circ N, s_1) \leq T(M \circ N, s_2))\]
which can be rewritten as

\[ \exists M \circ N \in \mathbb{SP}, \exists s_1 \in \mathbb{IN}_M, \forall s_2 \in \mathbb{IN}_{M,max}, \ T(M, s_1) < T(M, s_2) \wedge T(M \circ N, s_1) > T(M \circ N, s_2) \]

and further rewritten as

\[ \exists M \circ N \in \mathbb{SP}, \exists s_1 \notin \mathbb{IN}_{M,max}, \forall s_2 \in \mathbb{IN}_{M,max}, \ T(M \circ N, s_1) > T(M \circ N, s_2) \]

Since this states that there exists a \( s_1 \notin \mathbb{IN}_{M,max} \) resulting in a higher execution time on \( M \circ N \) than all states \( s_2 \in \mathbb{IN}_{M,max} \) will, the Series-Composition would be unsafe, since \( T_{sc} \) only considers states \( s \notin \mathbb{IN}_{M,max} \). Thus the correctness condition is necessary.

**Proof of Corollary 4.2 (Sufficient Condition for Series-Composition):**

Showing that Condition 10 implies Condition 9:

Condition 10 is defined as follows:

\[ \forall M \circ N \in \mathbb{SP}, \forall s_1 \in \mathbb{IN}_M, \forall s_2 \in \mathbb{IN}_M, \Delta(M, s_1, s_2) > 0 \rightarrow \Delta(M \circ N, s_1, s_2) \geq 0 \]

Condition 9 is exactly the same definition, except that instead of \( \forall s_2 \in \mathbb{IN}_M \) it is written \( \exists s_2 \in \mathbb{IN}_{M,max} \), where \( \mathbb{IN}_{M,max} \) is a subset of \( \mathbb{IN}_M \), which means that Condition 10 is stronger than Condition 9. Thus, Condition 9 is implied by Condition 10.

We use the following abbreviation of logical formulas to explain the different proofs steps for Theorem 4.3 and Theorem 4.4:

**SC**...safeness condition of Series-Composition (Theorem 4.1)

**Proof of Theorem 4.3 (Series Timing-Composability without TA-S-I):**

a) Is sufficient for the correctness of Series-Composition:

i.e., \( \neg TA-S-I \rightarrow SC \)

The formula \( \neg TA-S-I \) is equivalent to the sufficient correctness Condition 10 of Series-Composition. And using Corollary 4.2 we get that Condition 10 implies the safeness condition \( SC \) of Theorem 4.1. Thus, it has been shown that \( \neg TA-S-I \rightarrow SC \).

b) Is not necessary for the correctness of Series-Composition:

i.e., \( \neg(SC \rightarrow \neg TA-S-I) \)

Since the formula \( \neg TA-S-I \) is equivalent to Equation 10 we get from Corollary 4.2 that \( \neg TA-S-I \rightarrow SC \) holds but not \( \neg TA-S-I \rightarrow SC \) because Condition 10 is a stronger condition than \( SC \). Thus, it follows that \( \neg(SC \rightarrow \neg TA-S-I) \).

**Proof of Theorem 4.4 (Series Timing-Composability in presence of TA-S-A):**

Absence of \( TA-S-A \) is not necessary for the correctness of Series-Composition:

i.e., \( \neg(SC \rightarrow \neg TA-S-A) \), which is equivalent to \( (SC \wedge TA-S-A) \)

The formula \( SC \) is defined as:

\[ \forall M \circ N \in \mathbb{SP}, \forall s_1 \in \mathbb{IN}_M, \exists s_2 \in \mathbb{IN}_{M,max}, \Delta(M, s_1, s_2) > 0 \rightarrow \Delta(M \circ N, s_1, s_2) \geq 0 \]
where $\mathbb{N}_{M,\text{max}}$ is a subset of $\mathbb{N}_M$. The condition $\text{SC}$ can be rewritten as

$$\forall M \circ N \in \text{SP}, \ \forall s_1 \in \mathbb{N}_M, \exists s_2 \in \mathbb{N}_{M,\text{max}}. \ \Delta(M, s_1, s_2) \leq 0 \lor \Delta(M \circ N, s_1, s_2) \geq 0$$

The formula $\text{TA-S-A}$ is defined as:

$$\exists s, s' \in S. \ 0 < \Delta(M, s, s') < \Delta(M \circ N, s, s')$$

The formula $\neg \text{TA-S-A}$ requires that the absolute value of $\Delta(M \circ N, s_1, s_2)$ is larger than $\Delta(M, s_1, s_2)$. Since the formula $\text{SC}$ does not impose any constraint on their absolute values, both formulas can be true simultaneously, i.e., $(\text{SC} \land \neg \text{TA-S-A})$ can be fulfilled. Thus, the absence of $\text{TA-S-A}$ is not necessary for the correctness of Series-Composition.

\[ \square \]

### A.2 Proofs for Parallel Composition

**Proof of Theorem 5.1 (Safety of Delta-Composition):**

a) Showing that the condition is sufficient:

The correctness condition of Theorem 5.1 can be written as

$$\forall a \in A, \forall b \in B, \exists a' \in A_{\text{min}}, \exists b' \in B_{A,\text{max}}(a').$$

$$\Delta_{\text{hw}, A}(I, a', a) \leq 0 \lor \Delta(I, \langle a', b' \rangle, \langle a, b \rangle) \leq \Delta_{\text{hw}, A, \text{max}}$$

which can be rewritten as

$$\forall a \in A, b \in B, \exists a' \in A_{\text{min}}, \exists b' \in B_{A,\text{max}}(a').$$

$$T(I, a) \leq T(I, a') \lor T(I, \langle a, b \rangle) \leq (T(I, \langle a', b' \rangle) + \Delta_{\text{hw}, A, \text{max}})$$

Assuming the term $T(I, a) \leq T(I, a')$ is TRUE, it follows that $a \in A_{\text{min}}$. Delta-Composition is safe for the case $a \in A_{\text{min}}$ because it searches the maximum over all $a \in A_{\text{min}}$.

Assuming that $T(I, \langle a, b \rangle) \leq (T(I, \langle a', b' \rangle) + \Delta_{\text{hw}, A, \text{max}})$ is TRUE, the Delta-Composition is also safe, because Delta-Composition is computed as $\max_{a' \in A_{\text{min}}, b' \in B_{A,\text{max}}(a')}\Delta(I, \langle a', b' \rangle) + \Delta_{\text{hw}, A, \text{max}}$. Thus in this case it follows that all execution times are upper bounded by the Delta-Composition. To conclude, both cases imply that Delta-Composition is correct, which proves that the correctness condition is sufficient.

b) Showing that the condition is necessary:

We use proof by contradiction to show that the correctness condition is necessary. Thus, we assume that the negated condition holds:

$$\neg( \forall a \in A, \forall b \in B, \exists a' \in A_{\text{min}}, \exists b' \in B_{A,\text{max}}(a').$$

$$\Delta_{\text{hw}, A}(I, a', a) > 0 \rightarrow \Delta(I, \langle a', b' \rangle, \langle a, b \rangle) > \Delta_{\text{hw}, A, \text{max}})$$

which can be written as

$$\exists a \in A, b \in B, \forall a' \in A_{\text{min}}, \forall b' \in B_{A,\text{max}}(a').$$

$$\Delta_{\text{hw}, A}(I, a', a) > 0 \land \Delta(I, \langle a', b' \rangle, \langle a, b \rangle) > \Delta_{\text{hw}, A, \text{max}}$$

32
which, since \( \Delta_{hw,A}(I, a', a) > 0 \) implies \( a \not\in A_{min} \), can be rewritten as

\[
\exists a \not\in A_{min}, \exists b \in B, \forall a' \in A_{min}, \forall b' \in B_{A_{max}}(a').
\]

\[
T(I, (a, b)) > (T(I, (a', b')) + \Delta_{hw,A,max})
\]

As \( \max_{a' \in A_{min}, b' \in B_{A_{max}}(a')} (T(I, (a', b')) + \Delta_{hw,A,max}) \) is how the Delta-Computation is computed, it follows that there exist higher execution times and thus the Delta-Computation would not be safe. Thus the correctness condition is necessary.

**Proof of Corollary 5.2 (Sufficient Condition for Delta-Composition):**

Showing that Condition 13 implies Condition 12:

Condition 13 is defined as follows:

\[
\forall a \in A, \forall a' \in A, \forall b \in B. \quad \Delta_{hw,A}(I, a, a') > 0 \rightarrow \Delta_{hw,A}(I, a, a') \geq \Delta(I, (a, b), (a', b))
\]

from which the following specialization with \( a' \in A_{min} \) can be derived:

\[
\forall a \in A, \forall b \in B, \exists a' \in A_{min}. \quad \Delta_{hw,A}(I, a', a) > 0 \rightarrow \Delta_{hw,A}(I, a', a) \geq \Delta(I, (a', b), (a, b))
\]

As the definition of \( \Delta_{hw,A,max} \) implies that \( \Delta_{hw,A,max} \geq \Delta_{hw,A}(I, a', a) \), above condition implies the following condition

\[
\forall a \in A, \forall b \in B, \exists a' \in A_{min}. \quad \Delta_{hw,A}(I, a', a) > 0 \rightarrow \Delta(I, (a', b), (a, b)) \leq \Delta_{hw,A,max}
\]

which can be rewritten as

\[
\forall a \in A, \forall b \in B, \exists a' \in A_{min}. \quad \Delta_{hw,A}(I, a', a) > 0 \rightarrow T(I, (a, b)) \leq T(I, (a', b)) + \Delta_{hw,A,max}
\]

Since for any \( b' \in B_{A_{max}}(a') \) it holds that \( T(I, (a', b)) \leq T(I, (a', b')) \), above condition implies the following condition

\[
\forall a \in A, \forall b \in B, \exists a' \in A_{min}, \exists b' \in B_{A_{max}}(a'). \quad \Delta_{hw,A}(I, a', a) > 0 \rightarrow T(I, (a, b)) \leq T(I, (a', b')) + \Delta_{hw,A,max}
\]

which can be rewritten as

\[
\forall a \in A, \forall b \in B, \exists a' \in A_{min}, \exists b' \in B_{A_{max}}(a'). \quad \Delta_{hw,A}(I, a', a) > 0 \rightarrow \Delta(I, (a', b'), (a, b)) \leq \Delta_{hw,A,max}
\]

which is exactly the definition of Condition 12.

**Proof of Theorem 5.3 (Safety of Max-Composition):**

a) Showing that the condition is sufficient:
The correctness condition of Theorem 5.3 can be written as
\[ \forall a \in A, \forall b \in B, \exists a' \in A_{max}, \exists b' \in B_{A_{max}}(a'). \]
\[ \Delta_{hw,A}(I, a, a') \leq 0 \lor \Delta(I, (a, b), (a', b')) \geq 0 \]
which can be rewritten as
\[ \forall a \in A, b \in B, \exists a' \in A_{max}, \exists b' \in B_{A_{max}}(a'). \]
\[ T(I, a') \leq T(I, a) \lor T(I, (a', b')) \geq (T(I, (a, b))) \]
Assuming the term \( T(I, a') \leq T(I, a) \) is True, it follows that \( a \in A_{max} \). Max-Composition is safe for the case \( a \in A_{max} \) because it searches the maximum over all \( a \in A_{max} \).
Assuming that \( T(I, (a', b')) \geq (T(I, (a, b))) \) is True, the Max-Composition is also safe, because the Max-Composition is computed as follows: \( \max_{a' \in A_{max}, b' \in B_{A_{max}}(a')} (T(I, (a', b'))) \). Thus in this case it follows that all execution times are upper bounded by the Max-Composition. To conclude, both cases imply that Max-Composition is correct, which proves that the correctness condition is sufficient.

b) Showing that the condition is necessary:
We use proof by contradiction to show that the correctness condition is necessary. Thus, we assume that the negated condition holds:
\[ \neg( \forall a \in A, \forall b \in B, \exists a' \in A_{max}, \exists b' \in B_{A_{max}}(a'). \]
\[ \Delta_{hw,A}(I, a, a') > 0 \rightarrow \Delta(I, (a, b), (a', b')) \geq 0 ) \]
which can be written as
\[ \exists a \in A, \exists b \in B, \forall a' \in A_{max}, \forall b' \in B_{A_{max}}(a'). \]
\[ \Delta_{hw,A}(I, a, a') > 0 \land \Delta(I, (a, b), (a', b')) < 0 \]
which, since \( \Delta_{hw,A}(I, a, a') > 0 \) implies \( a \notin A_{max} \), can be rewritten as
\[ \exists a \notin A_{max}, \exists b \in B, \forall a' \in A_{max}, \forall b' \in B_{A_{max}}(a'). \]
\[ T(I, (a', b')) > T(I, (a, b)) \]
As \( \max_{a' \in A_{max}, b' \in B_{A_{max}}(a')} (T(I, (a', b'))) \) is how the Max-Computation is computed, it follows that there exist higher execution times and thus the Max-Computation would not be safe. Thus the correctness condition is necessary.

Proof of Corollary 5.4 (Sufficient Condition for Max-Composition):
a) Showing that Condition 17 implies Condition 16:
Condition 17 is defined as follows:
\[ \forall a \in A, \forall a' \in A, \forall b \in B. \]
\[ \Delta_{hw,A}(I, a, a') > 0 \rightarrow \Delta(I, (a, b), (a', b')) \geq 0 \]

34
from which the following specialization with \( a' \in A_{\text{max}} \) can be derived:

\[
\forall a \in A, \forall b \in B, \exists a' \in A_{\text{max}}. \quad \Delta_{\text{hw}, A}(I, a, a') > 0 \implies \Delta(I, (a, b), (a', b)) \geq 0
\]

which is exactly the definition of Condition 16.

b) Showing that the sufficient correctness Condition 16 of Max-Composition implies the sufficient and necessary correctness Condition 15:

Condition 16 is defined as follows:

\[
\forall a \in A, \forall b \in B, \exists a' \in A_{\text{max}}. \quad \Delta_{\text{hw}, A}(I, a, a') > 0 \implies \Delta(I, (a, b), (a', b)) \geq 0
\]

Since for any \( b' \in B_{A,\text{max}}(a') \) it holds that \( T(I, (a', b)) \leq T(I, (a', b')) \), above condition implies the following condition

\[
\forall a \in A, \forall b \in B, \exists a' \in A_{\text{max}}, \exists b' \in B_{A,\text{max}}(a'). \quad \Delta_{\text{hw}, A}(I, a, a') > 0 \implies T(I, (a', b')) \geq T(I, (a, b))
\]

which can be rewritten as

\[
\forall a \in A, \forall b \in B, \exists a' \in A_{\text{max}}, \exists b' \in B_{A,\text{max}}(a'). \quad \Delta_{\text{hw}, A}(I, a, a') > 0 \implies \Delta(I, (a, b), (a', b')) \geq 0
\]

which is exactly the definition of Condition 15.

We use the following abbreviations of logical formulas to explain the different proofs steps for Theorem 5.5 and Theorem 5.6:

**DC**... correctness condition of Delta-Composition (Theorem 5.1)

**MC**... correctness condition of Max-Composition (Theorem 5.3)

**Proof of Theorem 5.5 (Parallel Timing-Composability without TA-P-I):**

a) Is sufficient for the correctness of Max-Composition:

i.e., \( \neg \text{TA-P-I} \rightarrow \text{MC} \)

The formula \( \neg \text{TA-P-I} \) is equivalent to the sufficient correctness Condition 17 of Max-Composition. And using Corollary 5.4 we get that Equation 17 implies the safeness condition MC of Theorem 5.3. Thus, it has been shown that \( \neg \text{TA-P-I} \rightarrow \text{MC} \).

b) Is not necessary for the correctness of Max-Composition:

i.e., \( \neg (\text{MC} \rightarrow \neg \text{TA-P-I}) \)

Since the formula \( \neg \text{TA-P-I} \) is equivalent to Equation 17 we get from Corollary 5.4 that \( \neg \text{TA-P-I} \rightarrow \text{MC} \) holds but not \( \neg \text{TA-P-I} \leftrightarrow \text{MC} \) because Equation 17 is a stronger condition than MC. Thus, it follows that \( \neg (\text{MC} \rightarrow \neg \text{TA-P-I}) \).

c) Is not sufficient for the correctness of Delta-Composition:

i.e., \( \neg (\neg \text{TA-P-I} \rightarrow \text{DC}) \)

We have to show that there exists a model (of hardware and software) such that \( \neg (\neg \text{TA-P-I} \rightarrow \text{DC}) \) is
satisfied. The formula \( \models \neg(\neg TA-P-I \rightarrow DC) \) can be rewritten as \( \models (\neg TA-P-I \land \neg DC) \) with \( \neg TA-P-I \) (using Equation 22):

\[
\exists a, a' \in A, b \in B. \quad \Delta_{hw_A}(I, a, a') = 0 \lor \Delta(I, \langle a, b \rangle, \langle a', b \rangle) = 0 \lor (\Delta_{hw_A}(I, a, a') > 0 \land \Delta(I, \langle a, b \rangle, \langle a', b \rangle) > 0) \lor (\Delta_{hw_A}(I, a, a') < 0 \land \Delta(I, \langle a, b \rangle, \langle a', b \rangle) < 0)
\]

and \( \neg DC \):

\[
\exists a \in A, \exists b \in B, \forall a' \in A_{min}, \forall b' \in B_{A,max}(a'). \\
\Delta_{hw_A}(I, a', a) > 0 \land \Delta(I, \langle a', b' \rangle, \langle a, b \rangle) > \Delta_{hw_A,max}
\]

To demonstrate that \( \models (\neg TA-P-I \land \neg DC) \) can be satisfied we assume a model of hardware and software (characterized by the functions \( \Delta(I, s, s') \) and \( \Delta_{hw_A}(I, a, a') \)) with the following property:

\[
\forall a, a' \in A, b \in B. \Delta(I, \langle a, b \rangle, \langle a', b \rangle) = 2 \cdot \Delta_{hw_A}(I, a, a')
\]

With this model both formulas, \( \neg TA-P-I \) and \( \neg DC \), are satisfied. Thus, the absence of \( TA-P-I \) is not sufficient for the correctness of Delta-Composition.

**Proof of Theorem 5.6 (Parallel Timing-Composability without TA-P-A):**

a) Is sufficient for the correctness of Delta-Composition:

i.e., \( TA-P-A \rightarrow DC \)

The formula \( \neg TA-P-A \) is equivalent to the sufficient correctness Condition 13 for Delta-Composition. And using Corollary 5.2 we get that Condition 13 implies the safeness condition \( DC \) of Theorem 5.1. Thus, it has been shown that \( \neg TA-P-A \rightarrow DC \).

b) Is not necessary for the correctness of Delta-Composition:

i.e., \( (DC \rightarrow \neg TA-P-A) \)

Since the formula \( \neg TA-P-A \) is equivalent to Equation 13 we get from Corollary 5.2 that \( \neg TA-P-A \rightarrow DC \) holds but not \( \neg TA-P-A \leftrightarrow DC \) because Equation 13 is a stronger condition than \( DC \). Thus, it follows that \( \neg (DC \rightarrow \neg TA-P-A) \).

c) Is not sufficient for the correctness of Max-Composition:

i.e., \( (\neg TA-P-A \rightarrow MC) \)

We have to show that there exists a model (of hardware and software) such that \( \neg (TA-P-A \rightarrow MC) \) is satisfied. The formula \( \models \neg (\neg TA-P-A \rightarrow MC) \) can be rewritten as \( \models (\neg TA-P-A \land \neg MC) \) with \( \neg TA-P-A \) (using Equation 22):

\[
\exists a, a' \in A, b \in B. \quad (\Delta_{hw_A}(I, a, a') = 0) \lor \\
(\Delta(I, \langle a, b \rangle, \langle a', b \rangle) = \Delta_{hw_A}(I, a, a')) \lor \\
(0 > \Delta_{hw_A}(I, a, a') \land \Delta(I, \langle a, b \rangle, \langle a', b \rangle) \geq \Delta_{hw_A}(I, a, a')) \lor \\
(0 < \Delta_{hw_A}(I, a, a') \land \Delta(I, \langle a, b \rangle, \langle a', b \rangle) \leq \Delta_{hw_A}(I, a, a'))
\]

and \( \neg MC \):

\[
\exists a \in A, \exists b \in B, \forall a' \in A_{max}, \forall b' \in B_{A,max}(a'). \\
\Delta_{hw_A}(I, a, a') > 0 \land \Delta(I, \langle a, b \rangle, \langle a', b' \rangle) < 0
\]
To demonstrate that $\models (\neg \text{TA-P-A} \land \neg \text{MC})$ can be satisfied we assume a model of hardware and software (characterized by the functions $\Delta(I, s, s')$ and $\Delta_{\text{hw}}(I, a, a')$) with the following property:

$$\forall a, a' \in A, b \in B. \Delta(I, \langle a, b \rangle, \langle a', b \rangle) = -\Delta_{\text{hw}}(I, a, a')$$

With this model both formulas, $\neg \text{TA-P-A}$ and $\neg \text{MC}$, are satisfied. Thus, the absence of $\text{TA-P-A}$ is not sufficient for the correctness of Max-Composition. \qed