Compositional Timing-Aware Semantics for Synchronous Programming

Citation for published version:
Aguado, J, Mendler, M, Wang, J, Bodin, B & Roop, PS 2018, Compositional Timing-Aware Semantics for Synchronous Programming. in FDL 2017 Forum on specification & Design Languages. Institute of Electrical and Electronics Engineers (IEEE), Forum on Specification & Design Languages (FDL 2017), 18/09/17. DOI: 10.1109/FDL.2017.8303895

Digital Object Identifier (DOI):
10.1109/FDL.2017.8303895

Link:
Link to publication record in Edinburgh Research Explorer

Document Version:
Peer reviewed version

Published In:
FDL 2017 Forum on specification & Design Languages

Compositional Timing-Aware Semantics for Synchronous Programming

Joaquín Aguado*, Michael Mendler*, Jia Jie Wang‡, Bruno Bodin† and Partha Roop‡
* University of Bamberg, Germany. † University of Edinburgh, United Kingdom. ‡ University of Auckland, New Zealand.

***Abstract*—In this paper we propose a WCRT analysis technique for synchronous programs, executed as sequential or multi-threaded code, based on formal power series in min-max-plus algebra. The algebraic model constitutes the first fully declarative timing-aware semantics of synchronous programs with arbitrary hierarchical control-flow structure. Under signal abstraction this model permits efficient compositional WCRT analyses based on structural boxes as the unit of composition. The algebraic model leads to a sound methodology to deal with the state space explosion arising from tick alignment of parallel composition by reduction to the maximum weighted clique problem.**

***Index Terms*—Algebra, Timing, Systems Modeling Language**

I. Introduction

The synchronous paradigm [1] is ideal for designing safety critical systems in aviation, automotive and industrial automation. Synchronous languages offer a simple mechanism, based on a logical global clock, for thread synchronisation. This removes the inter-leavings and associated non-determinism of asynchronous composition, resulting in a framework that is more amenable for static analysis for functional correctness. The issue of timing correctness is at the heart of many real-time safety critical systems and is the topic of our interest.

Timing correctness of synchronous programs is closely intertwined with the *synchrony hypothesis*, which asserts that the synchronous program operates infinitely fast relative to its environment. Practical implementations validate this by ensuring that inputs from the environment never happen at a rate that is faster than the *worst case reaction time* (WCRT) of any synchronous reaction (*tick*). Compared to the problem of worst case execution time (WCET) [2] of sequential programs, WCRT analysis has received much less attention. What is the difference? WCET asks for the worst-case execution time over *all* initial memories of a code that is executed *once*. WCRT analyses a step function that is started in a *fixed* initial memory but *iterated* over *many* clock ticks. The worst-case is taken over all *reachable* memories. Because of the reachability aspect, WCRT produces tighter results than WCET. Despite this difference, there exist combined WCET/WCRT methods for parallel synchronous systems. For instance, the WCET of [3] is developed for parallel multicore applications and improves on precision by incorporating a WCRT for analyzing the synchronization time between cores and [4] investigates the response time of synchronous data flow programs mapped to many-core processor. However, interest in WCRT alone has been growing, with many recent attempts that primarily explore the trade-off between precision and analysis time:

(1) **Maximum thread cost** [5], [6]: These approaches compute the maximum tick lengths for every thread (termed their local ticks) and then the sum of these maximum local ticks to determine the WCRT. These, while being the most efficient, produce large overestimates.

(2) **Implicit path enumeration** [7], [8]: These approaches rely on integer linear programming (ILP) to model the constraints of a control flow graph and are inspired by ILP-based techniques for WCET analysis of sequential programs [2]. Hence, they convert the concurrent control flow of the synchronous program into its sequential equivalent before applying the ILP formulation. They can be used for pruning infeasible paths to obtain precise WCRT yet have a higher complexity (NP hard) compared to the polynomial complexity of the previous approach (1). We call this ILP_s (ILP sequential).

(3) **State exploration** [9], [10]: These approaches work on the concurrent control flow to compute the worst case tick length by examining all possible thread-valid inter-leavings. These approaches compute precise WCRT at the expense of exponential worst case complexity. A recent paper [11] compares model checking [12], reachability [9], and ILP_s [7]. This shows that reachability works best in practice compared to the other techniques for large state space (above 10^6 states).

(4) **Iterative tightening** [11], [13]: Wang et al. [11] noticed that there is a trade-off between approach (1) and the approaches based on path enumeration approach (2) or state exploration approach (3). They developed an iterative refinement approach called ILP_C (ILP concurrent), by the creation of two different ILP models on the concurrent control flow graph. The first is used to compute an over-approximation of the maximum cost of local ticks and uses a second ILP model to check if the over-approximation is infeasible (i.e., ticks do not align during execution). They iteratively refine this until the most precise value is computed. They have shown that ILP_C performs the best among known approaches for large benchmarks. Independently from this, a strategy for iterative tightening has been proposed by Raymond et al. [13] for ILP_s. They employ *flow facts* or *infeasibility properties* (verified as invariants using a model-checker) at the high-level source language (Lustre) to derive low-level path constraints on the scheduled sequential program to guide the ILP solver towards tighter WCRT values.

A unifying approach that would make it possible to integrate these various techniques systematically has recently been proposed based on formal power series in min-max-plus algebra [14]. This paper further investigates these algebraic techniques and makes the following contributions:

- This is the first fully functional and time-aware semantics for SP (Sec. IV). It models arbitrary hierarchical, sequential and parallel program structure as well as signals. Existing modelling techniques only treat the flat parallel composition of sequential automata, e.g., [14], or are fully structural but do not have signals, e.g., [15].
- For signal-abstracted WCRT we present the first fully modular modelling approach, which can be directly implemented to generate a practical WCRT analysis algorithm. Our modelling is based on boxes as the unit of composition (Sec. V). This leads to a methodological improvement of [15]. Further, our modelling fits with the definition of timing compositionality proposed in [16].
- By exploring algebraic properties of the new approach (clock decomposition) we show how to deal with the state space explosion arising from parallel composition, called the tick alignment problem (TAP). We show how to reduce TAP to the maximum weighted clique problem (MWCP) which can be solved using standard algorithms. We present experimental evidence that this results in improved performance (Secs. VI and VII).

II. Timed Concurrent Control Flow Graph

The proposed WCRT approach is applied to PRET-C and its intermediate format *Timed Concurrent Control Flow Graph* (TCCFG) [12]. A TCCFG has the following types of nodes: conventional *start*, *end*, *computation* and *condition* nodes, with additional *abort-start* and *abort-end* nodes for preemption, *fork* and *join* nodes for concurrency, and *EOT* nodes for the pauses (i.e., state boundaries). This TCCFG captures all the information required in the WCRT analysis including the high-level control flow and the timing information back-annotated from the underlying hardware. Fig. 1 shows a TCCFG where each node B is annotated with an execution cost $\mathsf{wcrt}(B)$ in processor clock cycles. In our case, these costs are derived using the technique presented in [17]. The WCRT problem for a TCCFG is to compute the maximal duration of any tick under the operational semantics of PRET-C [12] assuming each node B takes exactly $\mathsf{wcrt}(B)$ units of time to complete. Concurrency is implemented by (statically scheduled) multi-threading as in [12]. This means that the WCRT of a parallel composition is the *sum* of the WCRT of its threads rather than their maximum. The timing of EOT nodes are delays added in the tick in which the EOT is reached. The timing costs of all other nodes count for the tick in which the node is exited.

Table I shows the execution traces of the running example for the first ticks. In these traces, we assume the preemption is false unless stated otherwise in the event column. Threads in a PRET-C program execute in a static order, from left to right in the TCCFG. A thread only switches to the next one when it reaches an EOT node. The tick count advances when all the active threads have reached their respective EOT nodes.

The program execution begins from the start node *B1*, and reaches the abort-start node *B2* which spawns two threads:

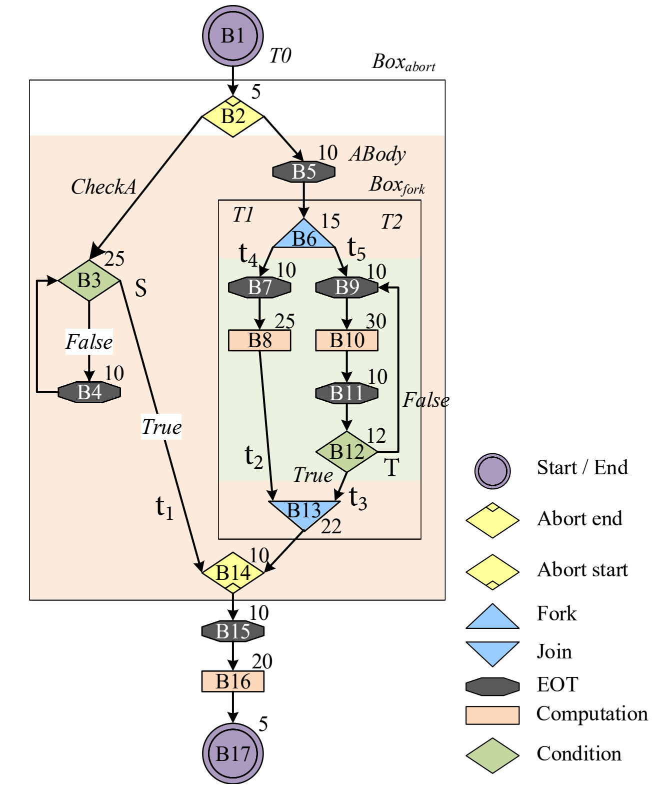

Fig. 1. Timed concurrent control flow graph (TCCFG), adapted from [11].

Tick count	Execution path	Events during that tick
1	*B1* → *B2* → *B3* → *B4* → *B5*	Entering abort
2	*B3* → *B4* → *B6* → *B7* → *B9*	Forking *T1* and *T2*
3	*B3* → *B4* → *B8* → *B10* → *B11*	*T1* terminates
4a	*B3* → *B4* → *B12* → *B13* → *B14* → *B15*	*T2* terminates, joining
4b	*B3* → *B14* → *B15*	Preemption
5	*B16* → *B17*	Program finishes

TABLE I
Tick snapshots of the TCCFG example of Fig. 1.

CheckA for checking the abort condition, and the abort body *ABody*. This is a strong abort since the *CheckA* thread has a higher priority (i.e., to the left of) than *ABody* thread. At the end of *tick 1*, *CheckA* and *ABody* pause at their EOT nodes *B4* and *B5* respectively. In *tick 2*, the execution resumes from these EOT nodes. The thread *ABody* spawns *T1* and *T2* using the fork node *B6* and suspends itself. In *tick 3*, *T1* terminates as it reaches the join node *B13*, and *T2* pauses at the EOT in *B11*. In *tick 4*, we present two scenarios. If preemption does not take place (*tick 4a*), *T2* terminates and activates *B13* and the program pauses at *B15*. If preemption takes place (*tick 4b*), *CheckA* reaches the abort-end node *B14*, preempting all threads in the abort body, and the program pauses at *B15*. In either case, the program finishes in *tick 5* at node *B17*.

Computing the WCRT from a TCCFG is to find the longest possible execution time for a tick. For example, for the TCCFG in Fig. 1, by looking at the six execution traces in Table I, the longest tick is *tick 3*, which has an execution cost of 100.

III. Min-Max-Plus Formal Power Series

The max-plus structure is $(\mathbb{N}_\infty, \oplus, \odot, \mathbb{0}, \mathbb{1})$ where $\mathbb{N}_\infty =_{df} \mathbb{N} \cup \{-\infty, +\infty\}$, $\oplus$ stands for the maximum and $\odot$ for addition on $\mathbb{N}_\infty$. Both operators are commutative, associative and have neutral elements $\mathbb{0} =_{df} -\infty$ and $\mathbb{1} =_{df} 0$, respectively, i.e., $x \oplus \mathbb{0} = x$ and $x \odot \mathbb{1} = x$. The constant $\mathbb{0}$ is absorbing for $\odot$, i.e., $x \odot \mathbb{0} = \mathbb{0} \odot x = \mathbb{0}$. In particular, $-\infty \odot +\infty = -\infty$. Addition $\odot$ distributes over max $\oplus$, i.e., $x \odot (y \oplus z) = x + \mathit{max}(y, z) = \mathit{max}(x + y, x + z) = (x \odot y) \oplus (x \odot y)$. This induces on $\mathbb{N}_\infty$ a (commutative, idempotent) semi-ring. The notation $\odot$ and $\oplus$ highlights the multiplicative and additive nature, respectively, of the operators. As usual, $x \odot y$ is also written $x\, y$. $\mathbb{N}_\infty$ is not only a semi-ring but also a complete lattice with the natural ordering $\leq$. Meet is $x \wedge y = \mathit{min}(x, y)$ and join is $x \vee y = \mathit{max}(x, y) = x \oplus y$. Further, $-\infty$ and $+\infty$ are the minimal and maximal elements. We can get least and greatest solutions of fixed-point equations by taking infinite join $\bigvee$ and meet $\bigwedge$, respectively.

A comprehensive study of the theory of max-plus algebra, and its generalisation the *dioids*, can be found in [18]. The important role of this structure for solving path problems is highlighted e.g. in [19]. What is rarely exploited, however, is the fact that the lattice structure of this algebra also supports logical reasoning, built around the *min* operation. The logical view is natural for our application where the values in $\mathbb{N}_\infty$ represent stabilisation times and measure the presence or absence of signals during a tick. The bottom element $\mathbb{0} = -\infty$ indicates that a signal is *absent*, i.e., is never going to become active. Logically, this corresponds to falsity, usually written $\perp$. A signal with an upper bound stabilisation time of $+\infty$ on the other hand is known to become *present eventually*. This is weak logical truth, written $\top$. All other stabilisation values $d \in \mathbb{N}$ codify *bounded presence* which are forms of truth stronger than $\top$. On these multi-valued forms of truth (aka "presence") the minimum operation $\wedge$ acts like logical conjunction while the maximum $\oplus$ is logical disjunction $\vee$. The behaviour of $\top = +\infty$ and $\perp = -\infty = \mathbb{0}$ with respect to $\wedge$ and $\vee$ follows the classical Boolean truth tables. For synchronous programs, negation is important to model data-dependent branching, priorities and (if needed) preemption. It is defined as $\neg x = \top$ if $x = \perp$ and $\neg x = \perp$ if $x \geq 0$.

A *(max-plus) formal power series*, *fps*, is an ω-sequence

$$A \;=\; \bigoplus_{i\geq 0} a_i X^i \;=\; a_0 \oplus a_1\, X \oplus a_2\, X^2 \oplus a_3\, X^3 \cdots \qquad (1)$$

with $a_i \in \mathbb{N}_\infty$ and where exponentiation is repeated multiplication, i.e., $X^0 = \mathbb{1}$ and $X^{k+1} = X\, X^k = X \odot X^k$. An fps stores an infinite sequence of numbers $a_0, a_1, a_2, a_3, \ldots$ as the scalar coefficients of the base polynomials X^i. An fps A may model the time cost a_i for a thread A to complete each tick i or to reach a given state A. If $a_i = \mathbb{0}$ then this means that thread A is not executed during the tick i and thus not contributing to the tick cost, or that a state A is not reachable during this tick. This contrasts with $a_i = \mathbb{1}$ which means A is executed during tick i but with zero cost, or that the state A is active at the beginning of the tick. If $a_i > \mathbb{0}$ then thread A is executed taking at most a_i time to finish tick i, or state A is reached within a_i-time during the selected tick. We can evaluate A with $X = \mathbb{1}$, written $A[\mathbb{1}]$, and obtain the worst-case time across all ticks. Note that an fps A could also be used to model a signal. Then, $a_i = \mathbb{0}$ is equivalent to the signal being absent in tick i. Otherwise, $a_i = \mathbb{1}$ implies s is present from the beginning of the tick, while $a_i > \mathbb{0}$ would mean that A becomes present during tick i with a maximal delay of a_i.

Let $\mathbb{N}_\infty[X]$ denote the set of fps over $\mathbb{N}_\infty$. For a comprehensive discussion of formal power series in max-plus algebra the reader is referred to [18]. Constants $d \in \mathbb{N}_\infty$ are naturally viewed as scalar fps $d = d \oplus \mathbb{0}X \oplus \mathbb{0}X^2 \oplus \cdots$. If we want d to be repeated indefinitely, we have to write $d^\omega = d \oplus dX \oplus dX^2 \cdots$. For finite state systems the fps will all be ultimately periodic. For compactness of notation we will write, e.g., $A = 0{:}2{:}(1{:}4)^\omega$ for the periodic sequence satisfying $A = 0 \oplus 2X \oplus X^2 B$ and $B = 1 \oplus 4X \oplus X^2 B$.

The operations $\oplus$ and $\odot$ are lifted naturally to power series. If $B = \bigoplus_{i\geq 0} b_i X^i$, then $A \oplus B$ is the tick-wise max $A \oplus B = \bigoplus_{i\geq 0}(a_i \oplus b_i) X^i$ and $A \| B$ the tick-wise lifting of $\odot$ given by $A \| B = \bigoplus_{i\geq 0}(a_i \odot b_i) X^i$. This series $A \| B$ executes A and B synchronously, adding the tick costs to account for the interleaving at the level of the instantaneous transitions (multi-threaded semantics). Sequential composition is (essentially) captured by *convolution* $A \odot B = \bigoplus_{i\geq 0} \bigoplus_{i=i_1+i_2}(a_{i_1} \odot b_{i_2}) X^i$. A special case is *scalar multiplication* (addition in $\mathbb{N}$) $d \odot A = \bigoplus_{i\geq 0} d \odot a_i X^i$. Scalar multiplication $\odot$, parallel composition $\|$ and conjunction $\wedge$ are distributive over $\oplus$. Since we want to express logical conditions on signals we also lift the logical operations to fps. Disjunction $\vee$ is identical to $\oplus$, conjunction $A \wedge B = \bigoplus_{i\geq 0}(a_i \wedge b_i) X^i$ and negation $\neg A = \bigoplus_{i\geq 0} \neg a_i X^i$. Convolution $\odot$, parallel composition $\|$ and conjunction $\wedge$ are distributive over $\oplus$.

IV. Algebraic Modelling of TCCFGs

This section extends the results of [14] to model arbitrary hierarchical sequential-parallel control-flow. We call all primitive elements of a TCCFG, particularly signals and transitions, the *controls*. For instance, the controls in Fig. 1 are $C \in \{B1, B2, \ldots, B17, S, T, t_1, t_2, t_3\}$. The logical behaviour, or *clock*, of C is an fps with coefficients $\perp = -\infty$ or $\top = +\infty$ indicating the ticks in which C is activated, starting at $B1$ in tick 1. We identify a control with its clock, which is obtained from other controls' clocks by recursion backwards along the structure of the TCCFG.

The generic specification method will be clear by applying it to the TCCFG of Fig. 1. Since $B1$ is the start node active only in tick 1, we have $B1 = \top$. The abort node $B2$ is reached instantaneously in the same tick in which $B1$ is active. Thus,

$$B2 = B1 = \top. \qquad (2)$$

The conditional node $B3$ in *CheckA* can be reached from $B2$ in the same tick or from $B4$ with one tick delay. Node $B4$ is activated in the same tick as $B3$ provided signal S is false. If

$B3$ is reached and S is true then t_1 is activated. This gives

$$B3 = B2 \vee XB4 \quad B4 = \neg S \wedge B3 \quad t_1 = S \wedge B3 \quad (3)$$

which completely describes the logical behaviour of *checkA*. In *ABody* the node $B5$ is activated from $B2$ unconditionally and instantaneously and $B6$ is reached one tick after $B5$ but only if the abort transition t_1 is not taken, i.e.,

$$B5 = B2 \qquad B6 = \neg t_1 \wedge XB5. \quad (4)$$

The node $B6$ activates the start nodes of both threads *T1* and *T2* for which we get the recursive equation system

$$B7 = B6 \qquad B9 = B6 \vee (\neg T \wedge B12) \quad (5)$$
$$B12 = \neg t_1 \wedge XB11 \qquad B11 = B10 \quad (6)$$
$$B10 = \neg t_1 \wedge XB9 \qquad B8 = \neg t_1 \wedge XB7 \quad (7)$$
$$t_2 = B8 \qquad t_3 = T \wedge B12. \quad (8)$$

Note that the non-abortion condition $\neg t_1$ needs only to be added to the exits of EOT nodes $B5$, $B7$, $B9$ and $B11$ since the strong abort is checked in *ABody* at the start of each tick. Next, we specify the activation of the join node $B13$. This happens if one of the threads *T1* and *T2* reaches its termination transition and the other has reached it in the same tick or earlier. To express this we need a latching operator on clocks. Define the clock $\mathsf{sync}(C)$ to start with $\perp$ and switch to $\top$ in the first tick in which C becomes $\top$. This leads to the recursive clauses $\mathsf{sync}(\perp \oplus XC) = \perp \oplus X\mathsf{sync}(C)$ and $\mathsf{sync}(\top \oplus XC) = \top^\omega$. The join then is

$$B13 \quad = \quad (\mathsf{sync}(t_2) \wedge t_3) \vee (t_2 \wedge \mathsf{sync}(t_3)). \quad (9)$$

Finally, we complete the algebraic specification of the main thread *T0* with the equations

$$B14 = t_1 \vee B13 \qquad B15 = B14 \quad (10)$$
$$B16 = XB15 \qquad B17 = B16. \quad (11)$$

To sum up, equations (2)–(11) describe the exact logical semantics of all controls of main thread *T0* in Fig. 1. It is timing-ignorant but fully parametric in environment signals. Note that this algebraic specification method is completely uniform and generalises to arbitrary TCCFGs.

To illustrate the fps logical semantics consider the scenario 4b of Tab. I, where signal S becomes present for good in tick 4, i.e., $S = X^3\top^\omega = \perp{:}\perp{:}\perp{:}\top^\omega$. Since $B3 = \top \vee XB4 = \top \vee X(\neg S \wedge B3) = \top \vee X(\neg(\perp{:}\perp{:}\perp{:}\top^\omega) \wedge B3) = \top \vee X(\top{:}\top{:}\top{:}\perp^\omega \wedge B3)$, it follows that $B3 = \top{:}\top{:}\top{:}\top{:}\perp^\omega$ and $t_1 = (\perp{:}\perp{:}\perp{:}\top^\omega) \wedge B3 = \perp{:}\perp{:}\perp{:}\top{:}\perp^\omega$ by (3). This means $B3$ remains active for 4 ticks until in the 4th tick the abort transition t_1 occurs. Hence, $B6 = \neg t_1 \wedge X\top = (\top{:}\top{:}\top{:}\perp{:}\top^\omega) \wedge X\top = \perp{:}\top{:}\perp^\omega$ by (2) and (4). Evaluating (5)–(8) we further derive $t_2 = \neg t_1 \wedge XB6$ which gives

$$\begin{aligned} t_2 &= \top{:}\top{:}\top{:}\perp{:}\top^\omega \wedge \perp{:}\perp{:}\top{:}\perp^\omega = \perp{:}\perp{:}\top{:}\perp^\omega \\ \mathsf{sync}(t_2) &= \perp{:}\perp{:}\top^\omega. \end{aligned}$$

Further, $\neg t_1 \wedge \neg t_1 X = \perp{:}\top{:}\top{:}\perp{:}\perp{:}\top^\omega$ and $X^2B6 = \perp{:}\perp{:}\perp{:}\top{:}\perp^\omega$. Hence, $\neg t_1 \wedge \neg t_1 X \wedge X^2 B6 = \perp^\omega$. Thus,

$$\begin{aligned} B12 &= \neg t_1 \wedge \neg t_1 X \wedge X^2 B9 \\ &= \neg t_1 \wedge \neg t_1 X \wedge X^2(B6 \vee (\neg T \wedge B12)) \\ &= (\neg t_1 \wedge \neg t_1 X \wedge X^2 B6) \\ &\quad \vee (\neg t_1 \wedge \neg t_1 X \wedge X^2 \neg T \wedge X^2 B12) \\ &= \perp^\omega \vee (\perp{:}\top{:}\top{:}\perp{:}\perp{:}\top^\omega \wedge X^2 \neg T \wedge X^2 B12)) \\ &= \perp{:}\top{:}\top{:}\perp{:}\perp{:}\top^\omega \wedge X^2 \neg T \wedge X^2 B12 \end{aligned}$$

from which it follows that $B12 = \perp^\omega$. This is precisely the statement that node $B12$ is never reached in scenario 4b when preemption occurs. It follows from (8) that $t_3 = T \wedge B12 = \perp^\omega$, $\mathsf{sync}(t_3) = \perp^\omega$ and therefore

$$\begin{aligned} B13 &= (\mathsf{sync}(t_2) \wedge t_3) \vee (t_2 \wedge \mathsf{sync}(t_3)) = \perp^\omega \\ B15 &= B14 = t_1 \vee B13 = \perp{:}\perp{:}\perp{:}\top{:}\perp^\omega \\ B17 &= B16 = XB15 = \perp{:}\perp{:}\perp{:}\perp{:}\top{:}\perp^\omega \end{aligned}$$

by (9)–(11). This is precisely scenario 4b of Tab. I.

A full timing-aware semantics can be obtained by adding timing costs into the equations (2)–(11). More specifically, for each control C let $\underline{C}$ be the fps with coefficients in $\mathbb{N}_\infty$ describing the timing cost of reaching C in each tick. The logical clock C is a timing abstraction of $\underline{C}$ obtained[1] as $C = \underline{C} \odot \top$. For instance, to account for the timing costs of $B2$, $B3$ and $B4$ we would refine equation $B3 = B2 \vee XB4$ to become $\underline{B3} = (5 \odot \underline{B2}) \vee XB4$ and $B4 = \neg S \wedge B3$ to $\underline{B4} = 10 \odot (\neg S \wedge 25 \odot B3)$. Such WCRT modelling has been proposed in [14] for purely sequential control-flow, without fork/join and abort constructs, which we model here.

There is however another, more direct way of getting the timing behaviour from the logical clocks (2)–(11). We simply superimpose the timing costs of all nodes in the TCCFG qualified by their logical clock that determine their presence or absence in a given tick. Let $\mathcal{B}(T)$ be the set of all nodes of a thread T and $\mathsf{wcrt}(B) \in \mathbb{N}$ the timing cost associated with $B \in \mathcal{B}(T)$. The parallel composition

$$\mathsf{wcrt}(T) = \parallel \{(\mathsf{wcrt}(B) \text{ when } B) \mid B \in \mathcal{B}(T)\} \quad (12)$$

with d when $B = (d^\omega \wedge B) \vee \mathbb{1}^\omega$ is the fps for the worst-case tick costs contributed by thread T. Observe that the parallel term $\mathsf{wcrt}(B)$ when B is a time series with coefficient $\mathsf{wcrt}(B)$ in all ticks where B is active and coefficient 0 in all other ticks. We obtain the WCRT over all ticks by evaluating (12) at $X = \mathbb{1}$, i.e., by computing $\mathsf{wcrt}(T)[\mathbb{1}]$.

Consider thread $T2$ in Fig. 1 for which equation (12) is

$$\begin{aligned} \mathsf{wcrt}(T2) = 10 \text{ when } B9 \parallel 30 \text{ when } B10 \\ \parallel 10 \text{ when } B11 \parallel 12 \text{ when } B12. \quad (13) \end{aligned}$$

To compute the worst case $\mathsf{wcrt}(T2)[\mathbb{1}]$ we need to know the clocks $B9$, $B10$, $B11$ and $B12$, which depend on the environment signals. Let us assume that signals S and T are

[1] Note that $-\infty \odot \top = -\infty \odot +\infty = -\infty = \perp$ and $d \odot \top = d \odot +\infty = +\infty = \top$ for all $d > -\infty$.

constant absent, $S = T = \bot^\omega$, so no abort takes place and thread $T2$ remains in its cycle forever. Then, from (2)–(11):

$$B9 = (\bot{:}\top)^\omega \qquad B10 = B11 = \bot{:}(\bot{:}\top)^\omega \tag{14}$$

$$B12 = \bot{:}\bot{:}\bot{:}(\top{:}\bot)^\omega \tag{15}$$

This means

$$\begin{aligned}\mathsf{wcrt}(T2) &= (0{:}10)^\omega \parallel 0{:}(0{:}30)^\omega \parallel 0{:}(0{:}10)^\omega \parallel 0{:}0{:}0{:}(12{:}0)^\omega \\ &= 0{:}10{:}(30 \odot 10){:}((10 \odot 12){:}(30 \odot 10))^\omega \\ &= 0{:}10{:}40{:}(22{:}40)^\omega \end{aligned}\tag{16}$$

and thus $\mathsf{wcrt}(T2)[\mathbb{1}] = 0 \oplus 10 \oplus 40 \oplus 22 \oplus 40 = 40$.

In this fashion we can use the evaluation laws of WCRT algebra to calculate the WCRT of arbitrary threads via cost-weighted superposition of logical clocks. In the next section we will present an efficient compositional approach for modelling arbitrary TCCFGs when signals are abstracted.

V. Compositional Modelling of TCCFGs

We follow a similar approach to that of [15] in the sense that signals are abstracted and only the control-flow structure is considered. From this perspective, a program execution is a sequence of ticks, and each tick has two possible outcomes. The program can either pause at an EOT node and resume in the next tick, or reach the end node and terminate. It can be shown that for every TCCFG there is a timing-equivalent *minimal Tick Cost Automaton* (mTCA) as seen in Fig. 2. This automata is formed by a sequence of states, each one with two (cost) weighted transitions: one leading to the next state (pause) and the other leading to the end (exit) state. Since execution sequences are ultimately periodic, an mTCA eventually loops back to one of the previous states.

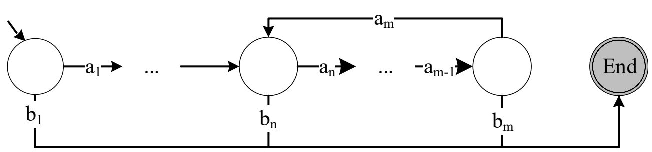

Fig. 2. The general form of a minimal tick cost automaton (mTCA).

The behaviour of an mTCA can be described by the types of control path taken inside the mTCA during a tick. In a *through* path control passes straight through the mTCA from the initial to the end node, i.e., transition b_1 in Fig. 2. In a *sink* path, the mTCA is entered from the initial node and control pauses in a sequential state to wait there for the next tick, i.e., transition a_1 in Fig. 2. A *source* path starts the tick from a sequential state and instantaneously reaches the end state, i.e., any of the transitions b_i with $i \geq 2$ in Fig. 2. Finally, a *internal* path, control starts inside the mTCA and stays there during the current instant, e.g., any of the transitions a_i with $i \geq 2$ in Fig. 2. To reflect this structure in terms of costs of paths, we define an mTCA A as a quadruple $(A_{thr}, A_{snk}, A_{src}, A_{int})$ where $A_{thr}, A_{snk} \in \mathbb{N}_\infty, A_{src}, A_{int} \in \mathbb{N}_\infty[X]$ are the associated costs of the through, sink, source and internal paths of A, respectively. For the mTCA of Fig. 2 we have than $A_{thr} = b_1$, $A_{src} = b_2{:}\cdots{:}b_m$, $A_{snk} = a_1$ and $A_{int} = a_2{:}\cdots{:}(a_n{:}\cdots{:}a_m)^\omega$.

The WCRT of a program is computed from the mTCA of the corresponding TCCFG. The modelling follows the hierarchical structure of the TCCFG and is based on "boxes" as the primitive units of composition. A *box* is a fragment of the TCCFG delimited by two transitions called *entry* and *exit* such that: (i) there is at least one control-flow path from entry to exit and (ii) every control-flow path of the TCCFG intersecting with the box goes from entry to exit. For instance, in Fig. 1, all the controls from transition t_5 (entry) to t_3 (exit) form the box of thread $T2$, and all the controls from transition t_4 (entry) to t_2 (exit) correspond to the box of thread $T1$. The modular translation maps each box into a timing equivalent mTCA. Intuitively this is possible because every box has a single entry and a single exit point relative to which the timing can be measured in the mTCA. For example, for the $box(t_5, t_3)$ of $T2$ the corresponding mTCA is given by $T2_{thr} = \mathbb{0}$, $T2_{src} = (\mathbb{0}{:}12)^\omega$, $T2_{snk} = 10$ and $T2_{int} = (40{:}22)^\omega$.

The idea is that these components describe the worst-case cost generated by the box at each tick when the corresponding path is executed. Since our intention is to obtain a timing equivalent mTCA from the hierarchical structure of a given TCCFG by means of algebraic manipulations, we define the following operations where A and B are mTCAs.

- *Sequential composition* $A;B$ is given by:

$$\begin{aligned}(A;B)_{thr} &= A_{thr} \odot B_{thr} \\ (A;B)_{snk} &= A_{snk} \oplus (A_{thr} \odot B_{snk}) \\ (A;B)_{src} &= (A_{src} \odot B_{thr}) \oplus ((A_{thr} \wedge \mathbb{1}) \odot B_{src}) \\ &\quad \oplus ((A_{src} \wedge \mathbb{1}^\omega) \odot XB_{src}) \\ (A;B)_{int} &= A_{int} \oplus (A_{src} \odot B_{snk}) \\ &\quad \oplus (((A_{thr} \oplus XA_{src}) \wedge \mathbb{1}^\omega) \odot B_{int})\end{aligned}$$

Intuitively, this indicates that the cost of the through path of a sequential composition is the addition of the cost of the through path of both components. The cost of the sink path is that of the sink path of the first component A otherwise it is the cost of the through path of A plus the cost of the sink path of B, which correspond to the two forms of entering from the initial state and pausing inside the sequential composition. A source path cost can be derived in three forms. First, it is the cost of a source path leaving A plus the cost of the through path crossing B. Second, it is the cost of a source path of B provided that A was left from its through path, i.e., if $A_{thr} \wedge \mathbb{1}$ is $\mathbb{1}$. For the third case, we observe that the coefficients of the fps $A_{src} \wedge \mathbb{1}^\omega$ select with $\mathbb{1}$ the ticks when a source path of A can occur and with $\mathbb{0}$ otherwise. The convolution of this fps with XB_{src} lines up the costs of the source paths of B starting from the next instant where the control is transferred from A. The internal path cost of a sequential composition can also occur in three ways. This is the cost of an internal path of A, the cost of a source path leaving A plus the cost of a sink path entering B or the cost of an internal path of B. Notice that in the latter case, the internal paths are obtained from the next tick when control enters B.

• *Parallel composition* $A \| B$ is determined by:

$$\begin{aligned}(A\|B)_{thr} &= A_{thr} \odot B_{thr} \\ (A\|B)_{snk} &= (A_{snk} \odot B_{snk}) \oplus (A_{snk} \odot B_{thr}) \\ &\quad \oplus (A_{thr} \odot B_{snk}) \\ (A\|B)_{src} &= (A_{src} \,\|\, \mathsf{sync}^\star(B_{src})) \oplus (\mathsf{sync}^\star(A_{src}) \,\|\, B_{src}) \\ (A\|B)_{int} &= (A_{int} \,\|\, B_{int}) \oplus (A_{int} \,\|\, \mathsf{sync}^\star(B_{src})) \\ &\quad \oplus (\mathsf{sync}^\star(A_{src}) \,\|\, B_{int})\end{aligned}$$

For parallel composition, the cost of a through path is the addition of the cost of the through paths of both components, calculated as a result of an interleaving (multi-threaded) model. The cost of a sink path is the addition of costs of both sink paths (interleaving) or the cost of the through path of one component plus the cost of the sink path of the other component. A source path of $A\|B$ must reach a (global) end state. This occurs when the one parallel components takes a source path and the other component follows a source path or has already reached its (local) end state. For the latter, we define a synchronisation operator by the recursion $\mathsf{sync}^\star(\bot \oplus XC) = \bot \oplus X\mathsf{sync}^\star(C)$ and $\mathsf{sync}^\star(a \oplus XC) = a \oplus X(C \vee \mathbb{1}^\omega)$ if $\mathbb{0} < a$. The *termination aligned* source cost $\mathsf{sync}^\star(A_{src})$ then contributes the cost of A_{src} in each tick where A terminates, i.e., where A_{src} is positive, and the cost of $\mathbb{1}$ in each tick in which A does not terminate, i.e., where A_{src} is $\mathbb{0}$, but where it has terminated before. The term $\mathsf{sync}^\star(B_{src})$ is symmetric. The cost of the internal path of $A\|B$ at each instant is the maximum cost among the tick-wise additions of the costs of internal paths from both components or the cost of an internal path of one component plus a source path of the other, termination aligned in case it has already terminated.

• The other control structures such as branching and loops can be modelled in a similar fashion.

The WCRT of any mTCA A gets specified by

$$\mathsf{wcrt}(A) = A_{snk} \oplus A_{int} \oplus A_{thr} \oplus A_{src}$$

Thus $\mathsf{wcrt}(A)[\mathbb{1}]$ essentially takes the maximum path costs for A over all the ticks. This computes the WCRT of A independently of any context, so it assumes that the box A is isolated and activated (entered) in the first tick. In this view, primitive control nodes of a TCCFG are special case of boxes. From Fig. 1, take for instance the EOT node $B5$ where $\mathsf{wcrt}(B5) = 10$ corresponding to a sink path, or the computation node $B10$ with $\mathsf{wcrt}(B10) = 30$ representing an internal path. Now, the WCRT of a box has been obtained independently of a context, say for instance, the cost of thread $T2$, i.e., $\mathsf{wcrt}(t_5, t_3) = T_{snk} \oplus T_{int} \oplus T_{thr} \oplus T_{src}$. Then we can compute the cost considering an activation context (using clocks) whenever the entry transition t_5 is activated, i.e., $\mathsf{wcrt}(T2) = (t_5 \wedge \mathbb{1}^\omega) \odot \mathsf{wcrt}(t_5, t_3)[\mathbb{1}]$.

This modelling technique significantly improves on that of [15] in compositionality because we are able to specify any box inside a TCCFG independent of its activation context. In [15] the activation context of a box is hard-wired and fixed by the specific TCCFG in which it appears. Both this paper and [15] depend on the abstraction of signals/data from the control–flow structure. Computing WCRT in this way gives an over-approximation which could add the time of concurrent control-flow paths that do not arise simultaneously due to data dependencies. In some cases, however, it is still possible to encode the data dependencies directly in the program to get a better WCRT approximation.

VI. Tick Alignment & Maximum Weight Cliques

The modularity of the approach in Sec. V is obtained by over-approximating control flow branching by non-deterministic choice. There is experimental evidence [15] that on typical synchronous programs this signal-abstract algebraic modelling via mTCAs performs significantly better than standard approaches via model-checking or ILP. Still, as exhibited on synthetic benchmarks in [15], the worst-case complexity remains exponential. This is because the naive algebraic expansion of a parallel composition $A \,\|\, B$ of fps amounts to a state exploration whose termination depends on the least common multiple of the cycle lengths of A and B.

We now show how one can avoid the state explosion algebraically by clock decomposition and frequency domain transformation. This result points towards a direct connection between formal power series, ILP modelling and a reduction of the tick alignment problem (TAP) to the maximum weighted clique problem (MWCP).

Let T be a TCCFG with nodes $\mathcal{B}(T) = \{B_1, B_2, \ldots, B_n\}$ and costs $b_i = \mathsf{wcrt}(B_i) \in \mathbb{N}$ as in Sec. IV. We want to compute $\mathsf{wcrt}(T)[\mathbb{1}]$ from the clock-decomposed form (12).

Unraveling the operators in (12) this is the same as

$$\begin{aligned}\mathsf{wcrt}(T)[\mathbb{1}] &= \bigoplus_{t \geq 0} \bigodot_{1 \leq i \leq n} T_{i,t} \\ &= max_{t \geq 0} \sum \{b_i \mid B_{i,t} = \top\} \qquad (17)\end{aligned}$$

with coefficients $B_{i,t} \in \{\bot, \top\}$ from the node clocks $B_i = \bigoplus_{t \geq 0} B_{i,t} X^t$ and $T_{i,t} = (b_i \wedge B_{i,t}) \vee \mathbb{1}$. Computing (17) generates the *Tick Alignment Problem* (TAP): Finding the maximum sum $\sum\{b_i \mid B_i \in \mathcal{B}\}$ for any *tick-aligned* set $\mathcal{B} \subseteq \mathcal{B}(T)$ of nodes, i.e., such that there exists $t \geq 0$ with $B_{i,t} = \top$ for all $B_i \in \mathcal{B}$. This can be rephrased as a finite combinatorial problem considering that each B_i is ultimately periodic with transient length τ_i and cycle length ϕ_i:

$$B_i = B_{i,0}{:}B_{i,1}{:}\cdots{:}B_{i,\tau_i-1}{:}(B_{i,\tau_i}{:}B_{i,\tau_i+1}{:}\cdots{:}B_{i,\tau_i+\phi_i-1})^\omega.$$

Because of periodicity, $B_{i,t_1} = B_{i,t_2}$ iff $min\{t_1, t_2\} < \tau_i$ and $t_1 = t_2$, or $min\{t_1, t_2\} \geq \tau_i$ and $t_1 - \tau_i \equiv_{\phi_i} t_2 - \tau_i$. Here and in the following $x \equiv_m y$ stands for congruence modulo m, i.e., $x \bmod m = y \bmod m$.

Proposition VI.1 (Tick Alignment Problem).
A candidate set $\mathcal{B} = \{B_i \mid i \in I\} \subseteq \mathcal{B}(T)$ is aligned iff there exist $0 \leq t_i < \tau_i + \phi_i$ for all $i \in I$ such that for all pairs of indices $i, j \in I$, we have $t_i = t_j$ or $\tau_i \leq t_i$, $\tau_j \leq t_j$ and $t_i - \tau_i \equiv_{gcd(\phi_i, \phi_j)} t_j - \tau_j$.

Proposition VI.1 suggests a decision procedure. We build a *tick alignment graph* $G_T = \langle V_T, E_T, w_T \rangle$ with vertices $V_T =_{df} \{(i, t_i) \mid 1 \leq i \leq n, 0 \leq t_i < \tau_i + \phi_i\}$ and weights $w(i, t_i) =_{df} T_{i,t_i}$. The edges E_T connect two vertices (i_1, t_{i_1}) and (i_2, t_{i_2}) if $t_{i_1} = t_{i_2}$ or both $\tau_{i_1} \leq t_{i_1}$, $\tau_{i_2} \leq t_{i_2}$ and $t_{i_1} - \tau_{i_1} \equiv_{g_{i_1 i_2}} t_{i_2} - \tau_{i_2}$ where $g_{i_1 i_2} = gcd(\phi_{i_1}, \phi_{i_2})$. We then search for a maximal weighted clique in G_T.

Proposition VI.2 (Max Weight Clique Problem)**.** *A candidate sum $T_{1,t_1} \odot T_{2,t_2} \odot \cdots \odot T_{n,t_n}$ is aligned iff the nodes $S = \{(i, t_i) \mid 1 \leq i \leq n\}$ form a clique in the TAG G_T. Hence, $\mathit{wcrt}(T)[\mathbb{1}] = max\,\{w(S) \mid S \text{ clique in } G_T\}$.*

Prop. VI.2 reduces the TAP to the Maximum Weight Clique Problem (MWCP), which is known to be NP-complete for arbitrary graphs [20]. This means that WCRT (under signal abstraction) is in NP which will be better behaved than the PSPACE approach of [15] using naive algebraic expansion of parallel composition in WCRT algebra.

VII. Practical Algorithms for TAP

Many algorithms have been proposed to solve the MWCP. The most well-known are encodings in Integer Linear Programming (ILP) style, see e.g. [20], or branch-and-bound search algorithms such as [21]. All these can be applied to obtain exact solutions for the TAP. We use the `wclique` program [21] which is publicly available and hence suitable for rapid prototyping. We also compare with StateExploration [9], [10]. Though these algorithms have exponential worst-case behaviour on arbitrary graphs, it is not known how they fare on tick alignment graphs. We conducted experiments to find out and the results are reported here. Alongside, we observed that the incremental WCRT evaluation method ILP$_C$ [11] is also based on a linear programming formulation. Exploiting Props. VI.1 and VI.2 we were able to obtain a simple but rather efficient improvement of ILP$_C$, which we term as ILP$_{CP}$.

The ILP$_C$ algorithm [11] starts with a linear program ILP$_{\text{BASE}}$ which approximates (17) by considering a set of nodes $\mathcal{B}$ aligned as long as the $B_i \in \mathcal{B}$ are in concurrent threads or connected by sequential control flow paths not containing any EOT nodes. Solving ILP$_{\text{BASE}}$ gives the same WCRT as the Maximum Thread Cost (MaxTC) approach [5], [6]. For instance, for the TCCFG of Fig. 1 such MaxTC candidates are the nodes $\mathcal{B}_0 = \{B2, B3, B8, B12, B13, B14, B15\}$ with a total cost of 109. This is a safe overapproximation. The incompatibility of these nodes is discovered using another linear program called ILP$_{\text{CHECK}}$ which finds that the intersection of the node clocks $B_i \in \mathcal{B}_0$, called *tick expressions* in [11], is empty. The principle of ILP$_C$ is then to improve the ILP$_{\text{BASE}}$ model iteratively by adding new constraints that rule out these infeasible combinations until reaching a valid one. In our example, the constraint $B2 \wedge B3 \wedge B8 \wedge B12 \wedge B13 \wedge B14 \wedge B15 = \perp^\omega$ is added to ILP$_{\text{BASE}}$. By thus refining ILP$_{\text{BASE}}$ iteratively, the optimal feasible solution $\mathcal{B}_* = \{B3, B4, B8, B10, B11\}$ with cost of 100 eventually appears.

In our improved version ILP$_{CP}$ we replace ILP$_{\text{CHECK}}$ by a polynomial infeasibility test using the tick alignment graph

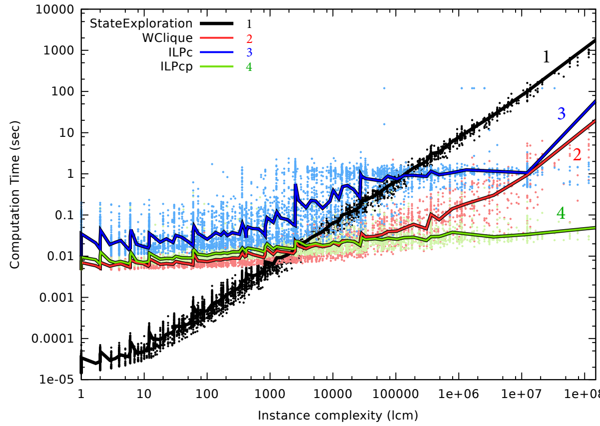

Fig. 3. Evaluation Results

G_T as per Prop. VI.2. We check *every pair* of candidate nodes for a connection in G_T. Thus, while ILP$_{\text{CHECK}}$ only rules out a *specific* candidate set such as $\mathcal{B}_0$, we rule out *every* candidate set that shares some infeasible pair. For instance, because $\{B2, B8\}$ and $\{B8, B12\}$ cannot be active in the same tick we add the constraints $B2 \wedge B8 = \perp^\omega$ and $B8 \wedge B12 = \perp^\omega$. Each missing edge in T_G not only witnesses the infeasibility of the given candidate sum but also of others. This tightens up the ILP$_{\text{BASE}}$ more effectively, whence we need fewer iterations.

Fig. 3 presents the results from our evaluation. In order to obtain an accurate performance estimation of our proposed method, we managed to randomly produce set of synthetic benchmarks composed of more than 8000 TAP instances of varying complexity i.e., the size of the reachability state expansion. Every point is corresponding to a particular instance of a TAP and the lines show the average trend of each evaluated method. It is not surprising that `wclique` is far superior to state exploration. The most surprising observation is that `wclique` is also superior to ILP$_C$, a domain-specific algorithm designed for WCRT analysis on signal-abstracted TCCFGs. However, ILP$_{CP}$ which exploits MWCP information in the narrowing loop, allows us to improve ILP$_C$ again, so it outperforms `wclique`.

From these experiments we believe that it could be interesting to adapt MWCP algorithms for extending the scope of the compositional algebraic method described in Sec. V which itself has already shown excellent performance on typical TCCFG structures [15].

VIII. Conclusions

Synchronous programs react to the environment using discrete instants, called reactions. Worst case reaction time analysis (WCRT) is essential to validate the correctness of the implementation of a program on an given architecture . Precise analysis requires the elimination of infeasible paths and infeasible state combinations from concurrent threads, known as the *tick alignment problem* (TAP).

This paper presents, for the first time, a compositional algebraic semantics of synchronous control-flow programs

(TCCFGs) to give a precise definition of the WCRT and the TAP. It is precise (called $(1,0)$-*timing compositional* in the terminology of [16]) because it combines both signal-dependent function and timing, unlike [5], [11], [15] which also use max-plus algebra but abstract from signals. It is general because it captures arbitrary hierarchical structures of sequential and concurrent control flow, unlike [14] which also uses formal power series in min-max-plus algebra but is restricted flat parallel compositions of sequential synchronous automata. Observe that the TAP is non-trivial because of the multi-threading semantics of PRET-C. Under multi-processing the total cost of a parallel is the maximum of its threads. Hence, we compute $max_{t \geq 0}\{b_i \mid B_{i,t} = \top\}$ instead of a max of sums as in (17) which is trival to obtain from the clocks B_i.

The computational complexity of solving the system equations to to determine $\mathsf{wcrt}(T)[\mathbb{1}]$ is unknown. There are two sources of combinatorial explosion. The first is the dependency on environment signals. To compute the exact worst-case we need to do computationally expensive case analysis on all possible behavioural patterns of environment signals (such as S and T in Fig. 1). Therefore, all practical timing analyses must abstract from signals in some way, as discussed in [14]. In this paper we have shown how the precision of signal-abstract WCRT can be improved efficiently using the number-theoretic structure of periodic activation clocks to reduce tick alignment to MWCP. The inclusion of signal dependencies is left to future work.

Existing works such as [22], [23] also exploit mathematical abstractions to obtain compositional real-time performance analyses. However, these typically abstract from causality and control flow which we model, while being able to express stochastic timing properties which we ignore.

Note that min-max-plus algebra can be used at all levels of abstraction, from low-level hardware to high-level program code. Here, we use it to analyse high-level TCCFG program behaviour. This has little meaning unless the timing parameters are linked with some compiled binary code. This is addressed by work such as [17], [24], [13]. The purpose of our approach is to complement low-level WCRT analyses to form a genuine round-trip process. The high-level design is informed by a prescription of the intended tick cost timing. Once the implementation is fixed, the actual low-level timing can be determined and back-annotated into the high-level design, see e.g. [25]. This enables the designer to change the program structure in order to fix or optimize the timing, e.g., by refactoring or the shifting of computations (code blocks) between tick boundaries. This holistic view has been termed *interactive timing analysis*, see e.g., [26].

Acknowledgement

This work has been supported by the German Research Council DFG under grant number ME-1427/6-2.

References

[1] A. Benveniste, P. Caspi, S. Edwards, N. Halbwachs, P. Le Guernic, and R. de Simone, "The synchronous languages 12 years later," *Proceedings of the IEEE*, vol. 91, no. 1, pp. 64 – 83, Jan 2003.

[2] R. Wilhelm, J. Engblom, A. Ermedahl, N. Holsti, S. Thesing, D. Whalley, G. Bernat, C. Ferdinand, R. Heckmann, T. Mitra, F. Mueller, I. Puaut, P. Puschner, J. Staschulat, and P. Stenström, "The worst-case execution-time problem—overview of methods and survey of tools," *Trans. on Embedded Computing Sys.*, vol. 7, no. 3, pp. 1–53, 2008.

[3] D. Potop-Butucaru and I. Puaut, "Integrated Worst-Case Execution Time Estimation of Multicore Applications," in *WCET'13*, Dagstuhl, Germany, 2013, pp. 21–31.

[4] H. Rihani, M. Moy, C. Maiza, R. I. Davis, and S. Altmeyer, "Response time analysis of synchronous data flow programs on a many-core processor," in *RTNS'16*. New York, NY, USA: ACM, 2016, pp. 67–76.

[5] M. Boldt, C. Traulsen, and R. von Hanxleden, "Worst Case Reaction Time Analysis of Concurrent Reactive Programs," *ENTCS*, vol. 203, no. 4, pp. 65–79, 2008.

[6] M. Mendler, R. von Hanxleden, and C. Traulsen, "WCRT Algebra and Interfaces for Esterel-Style Synchronous Processing," in *DATE'09*, Nice, France, Apr. 2009.

[7] L. Ju, B. K. Huynh, S. Chakraborty, and A. Roychoudhury, "Context-sensitive timing analysis of Esterel programs," in *DAC'09*. New York, NY, USA: ACM, 2009, pp. 870–873.

[8] L. Ju, B. K. Huynh, A. Roychoudhury, and S. Chakraborty, "Performance debugging of esterel specifications." *Real-Time Systems*, vol. 48, no. 5, pp. 570–600, 2012.

[9] M. Kuo, R. Sinha, and P. S. Roop, "Efficient WCRT analysis of synchronous programs using reachability," *DAC'11*, pp. 480–485, 2011.

[10] E. Yip, P. S. Roop, M. Biglari-Abhari, and A. Girault, "Programming and timing analysis of parallel programs on multicores," in *ACSD'13*, Barcelona, Spain, July 2013, pp. 160–169.

[11] J. J. Wang, P. S. Roop, and S. Andalam, "ILPc : A novel approach for scalable timing analysis of synchronous programs," in *CASES'13*, Montreal, Canada, Sept–Oct 2013, pp. 20:1–20:10.

[12] P. S. Roop, S. Andalam, R. von Hanxleden, S. Yuan, and C. Traulsen, "Tight WCRT analysis of synchronous C programs," in *CASES'09*, Grenoble, France, October 2009, pp. 205–214.

[13] P. Raymond, C. Maiza, C. Parent-Vigouroux, F. Carrier, and M. Asavoae, "Timing analysis enhancement for synchronous program," *Real-Time Systems*, vol. 51, no. 2, pp. 192–220, 2015.

[14] M. Mendler, P. S. Roop, and B. Bodin, "A novel WCET semantics of synchronous programs," in *FORMATS'16*, Quebec, Canada, August 2016, pp. 195–210.

[15] J. J. Wang, M. Mendler, P. Roop, and B. Bodin, "Timing analysis of synchronous programs using WCRT algebra: scalability through abstraction," in *EMSOFT'17*, Seoul, South Korea, October 2017.

[16] S. Hahn, J. Reineke, and R. Wilhelm, "Towards compositionality in execution time analysis: Definition and challenges," *SIGBED Rev.*, vol. 12, no. 1, pp. 28–36, Mar. 2015.

[17] R. Heckmann and C. Ferdinand, "Verifying safety-critical timing and memory-usage properties of embedded software by abstract interpretation," in *DATE'05*, Mar. 2005, pp. 618–619.

[18] F. L. Baccelli, G. Cohen, G. J. Olsder, and J.-P. Quadrat, *Synchronisation and Linearity*. John Wiley & Sons, 1992.

[19] T. A. Cormen, C. E. Leiserson, and R. L. Rivest, *Introduction to Algorithms*. MIT Press, 1990.

[20] P. M. Pardalos and J. Xue, "The maximum clique problem," *Journal of Global Optimization*, vol. 4, pp. 301–321, 1994.

[21] P. J. R. Östergård, "A new algorithm for the maximum-weight clique problem," *Nordic Journal of Computing*, vol. 8, pp. 424–436, 2001.

[22] E. Wandeler and L. Thiele, "Real-time interfaces for interface-based design of real-time systems with fixed priority scheduling," in *EMSOFT'05*, 2005.

[23] T. Henzinger and S. Matic, "An interface algebra for real-time components," in *RTAS'06*, Los Alamitos, USA, 2006, pp. 253–266.

[24] C. Ferdinand, R. Heckmann, T. L. Sergent, B. M. D. Lopes, X. Fornari, and F. Martin, "Combining a high-level design tool for safety-critical systems with a tool for WCET analysis on executables," in *ERTS'08*, Toulouse, France, February 2008.

[25] G. Logothetis, K. Schneider, and C. Metzler, "Exact low-level runtime analysis of synchronous programs for formal verification of real-time systems," in *FDL'03*, Frankfurt, Germany, 2003, pp. 385–405.

[26] I. Fuhrmann, D. Broman, R. von Hanxleden, and A. Schulz-Rosengarten, "Time for reactive system modeling: Interactive timing analysis with hotspot highlighting," in *RTNS'16*, Brest, France, 2016, pp. 289–298.