A Novel WCET semantics of Synchronous Programs

A Novel WCET Semantics of Synchronous Programs

Michael Mendler[1], Partha S Roop[2,4], and Bruno Bodin[3]

[1] Bamberg University, Germany
[2] University of Auckland, New Zealand
[3] University of Edinburgh, United Kingdom
[4] Mercator Fellow, Bamberg University, Germany

Abstract. Semantics for synchronous programming languages are well known. They capture the execution behaviour of reactive systems using precise formal operational or denotational models for verification and unambiguous semantics-preserving compilation. As synchronous programs are highly time critical, there is an imminent need for the development of an execution time aware semantics that can be used as the formal basis of WCET tools. To this end we propose such a compositional semantics for synchronous programs. Our approach, which is algebraic and based on formal power series in min-max-plus algebra, combines in one setting both the linear system theory for timing and constructive Gödel-Dummet logic for functional specification of synchronisation behaviour. The developed semantics is illustrated using a running example in the SCCharts language.

1 Introduction

The synchronous paradigm [4] is ideal for designing safety critical, real-time systems in aviation, automotive and industrial automation. The issue of timing correctness is at the heart of such systems and is the topic of our interest. In this paper, we concentrate on Esterel [5] style imperative synchronous languages (ISP) and their graphical counter parts such as Argos [17], SyncCharts [2], and SCCharts [23]. Semantics of such languages are well known [5, 17, 23]. These semantics primarily express execution behaviour using unambiguous mathematical notation. However, these semantics are time-abstract and unsuitable from the point of view of worst case execution time (WCET) analysis.

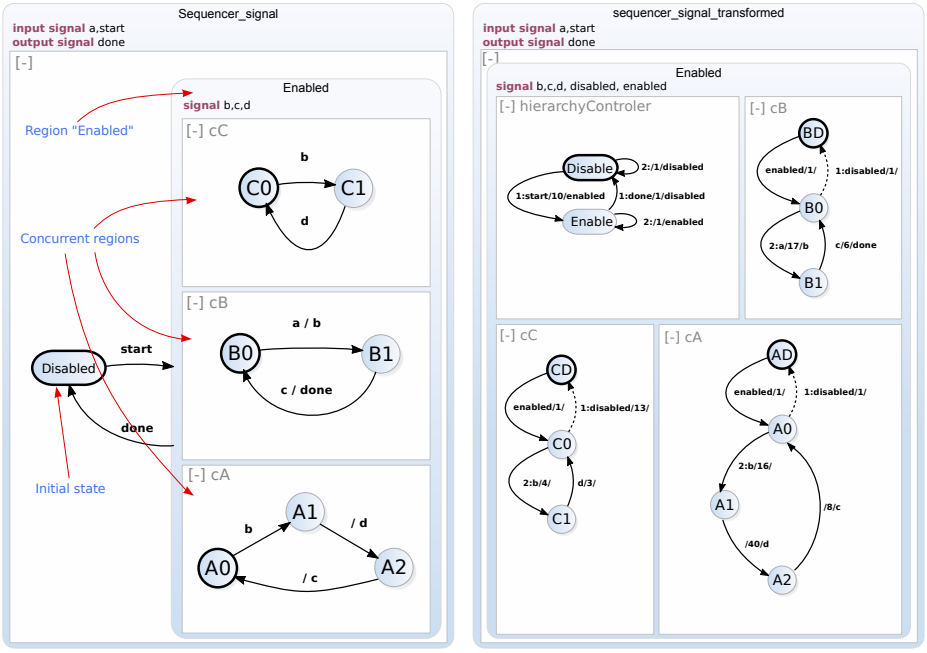
Existing WCET techniques for ISP [21, 24, 20] have been largely guided by heuristics using general-purpose analysis tools such as ILP, model-checking, SAT-solving, and micro-architectural modelling. The evaluations of the methods are based on empirical benchmarking. Systematic studies of semantic soundness and computational complexity of WCET heuristics are rare. To master the complexity of the WCET problem for ISP, so we believe, it will be necessary to balance the trade-off between efficiency and precision in a semantic model that permits the tight coupling of function and timing and is applicable at all levels of

abstraction, from high-level ISP programs down to low-level assembly or hardware, while also being abstract and language-independent. This paper proposes such a WCET semantics of synchronous languages based on logic and algebra. The application of this semantics outlined here is based on some, arguably strong, assumptions. First, we consider that programs are executed on precision timed architectures [10]. These simplify static timing analysis without sacrificing throughput by using thread-interleaved pipelines without pipeline speculation. They also use scratchpad memories instead of caches and are devoid of timing anomalies. This assumption may be relaxed to some extent by compositional use of techniques for architecture modelling [7]. Second, we assume that for each synchronous thread there is sufficient computation between two state boundaries. This assumption is essential for the annotation of timing cost with every transition of a synchronous thread. Third, we assume that concurrency is modelled by thread interleaving rather than multi-processing. Note, however, that this work is mainly theoretical. Our motivating running example does not necessarily demonstrate the most general use of the proposed algebraic semantics.

An overview of the paper is as follows. We introduce a running example with hierarchy, concurrency, and reactivity using an SCChart [23], which is presented in Section 2. We propose input-output Boolean tick cost automata (IO-BTCA) as an intermediate representation of synchronous threads. This is presented in Section 3. As our main contribution we develop an expressive constructive logic of formal power series extending Gödel-Dummett's intuitionistic logic as an algebraic semantics for IO-BTCA and their compositions. The theory is expounded in Section 4 and its application is illustrated Section 5 using the running example. Conclusions relative to related work is presented in Section 7.

2 Illustrative SCCharts Example

We use the SCCharts language to model the running example as shown in Figure 1a. The figure is annotated with the key features of the language used in this example. This language is a synchronous Statecharts [14] and the reader is referred to [23] for a detailed discussion on the language. The *sequencer* needs a `start` input signal to make progress from its initial state `Disabled` to the state `Enabled`. The state `Enabled` implements the actual specification of reaction: *after two ticks receiving the input* `a`*, the output* `done` *is emitted*. The sequencer uses local signals `b`, `c` and `d` to synchronize three concurrent threads `cC`, `cA` and `cB` (also called *regions*). Each thread is specified using a finite state machine with a unique start state e.g. $A0$, $B0$ and $C0$, indicated using bold circumferences. Each transition is labelled as `i/o`, where `i` is the guard that must be true for the transition to trigger, and `o` is the output part that is emitted when the transition is taken. Transitions are *non-immediate* and cannot be taken in the same tick when their source state is entered, except when they are marked as *immediate* using a dotted arrow as seen in Figure 1b. Concurrent regions are nested within a higher-level region. The first thread `cA` synchronizes with the second thread `cB` and the third thread `cC` using the local signals `b, c, d`. The



(a) The hierarchical, concurrent sequencer as it is seen in a SCChart visualizer.

(b) Sequencer flattened with timing back-annotations.

Fig. 1: Running sequencer example.

three concurrent threads have 36 possible configurations. Due to synchronous execution, however, only the three combinations $A0/B0/C0$, $A1/B1/C1$ and $A2/B1/C0$ are feasible. WCET analysis techniques for synchronous programs must detect such infeasibility to ensure tightness.

Intermediate representation without preemption. The hierarchical transition in Figure 1a is a weak preemption transition enabled by `done`. When this happens, all three threads are preempted and the behaviour moves to the initial state `Disabled`. Weak preemptions indicate causality i.e. the body terminates by generating an event that leads to the preemption transition being taken. Preemptions are handled in conventional semantics by introducing a separate hierarchical concurrency operator [17]. However, in the compilation chain towards executable sequential code, structural translation rules typically "compile away" the hierarchical transitions, which simplify WCET analysis.

Figure 1b shows the model generated after this structural translation. Each hierarchical region may be restructured by introducing one concurrent region per level of hierarchy. The concurrent region acts as a `controller` to activate and deactivate the appropriate sub-region (in the original specification). For example, we have introduced the region `hC` (hierarchyController) that waits until the `start` event to send an `enable` command to the other three concurrent regions.

These regions have an additional state $AD/BD/CD$ to indicate their disabled status. These regions can progress to their enabled state state $A0/B0/C0$ only when the `enable` event is provided by the controller. We have used *immediate* transitions from $A0/B0/C0$ to their respective disable state $AD/BD/CD$ upon receipt of the *done* event. This emulates the weak preemption in the original specification in Figure 1a.

To aid WCET analysis, transitions are annotated with upper bound timing cost. For instance the transition t_{C1C0} leading from state $C1$ to state $C0$ has an upper bound timing cost of $\mathsf{wcet}(t_{C1C0}) = 3$ while t_{B0B1} has $\mathsf{wcet}(t_{B0B1}) = 17$. The timing annotations seen in Fig. 1b are entirely fictive though technical feasibility of obtaining these values has been illustrated earlier in [16, 11].

3 Intermediate Level Semantics: Tick Cost Automata

WCET analysis is formulated over graph representations of conventional programs. We propose to model the timing-enriched behaviour of a sequential (single-threaded) synchronous program as an *input-output Boolean tick cost automaton* (IO-BTCA). Following the convention in SCCharts, we will draw non-immediate transitions as solid arrows and immediate transitions as dashed arrows, in the graphical representation of an IO-BTCA. Also, we label a transition t with the triple $grd(t)/del(t)/act(t)$. A state which has at least one non-immediate transition exiting from it is called a *pause* state. All other states are *transient* states. We say an automaton *pauses* if control reaches a pause state and the guards of all immediate transitions leaving the state, if any, are false. An immediate transition whose guard is true must be taken in the same tick in which the state is entered. The activation of a non-immediate transition is checked only in the next tick.

Definition 1. *An* input-output Boolean tick cost automaton *(IO-BTCA) is* $M = \langle Q, e, I, O, \rightarrow, e\rangle$*, where* $Q = states(M)$ *is a finite set of* states *with a distinguished* entry *state* $e = entry(M) \in Q$*.* $I = In(M)$ *and* $O = Out(M)$ *denote the set of* input *and* output *signals, respectively. The* transition relation $\rightarrow$ *is partitioned into the set of* immediate *transitions* $\rightarrow_i$ *and* non-immediate *transitions* $\rightarrow_n$*, i.e.,* $\rightarrow = \rightarrow_i \uplus \rightarrow_n$*. Each type of transitions is a relation* $\rightarrow \subseteq Q \times \mathcal{B}(I) \times \mathbb{N} \times 2^O \times Q$*, where* $\mathcal{B}(I)$ *denotes the set of Booleans over* I*. A transition* $t = (q_1, b, d, o, q_2) \in \rightarrow$ *connects a* source *state* q_1 *with a* target *state* q_2*. It is labelled by a Boolean* guard $b = grd(t)$ *over* I *specifying the condition under which the transition triggers, a* delay $d = del(t)$ *describing its worst case timing cost and a* set *of emitted signals* $o = act(t)$*.*

WCET of an IO-BTCA. An example of an IO-BTCA is shown in Fig. 2. This automaton A has transient states $A0$, $A5$ and $A6$ drawn as solid circles, and pause states $A1$, $A2$, $A3$ and $A4$ drawn as two half-circles. The transient entry node $A6$ is indicated by a transition arrow without source state. Each pause state is split into two parts. The upper half of each pause state represents the *surface* of the state. When the surface is reached, it can be left immediately in

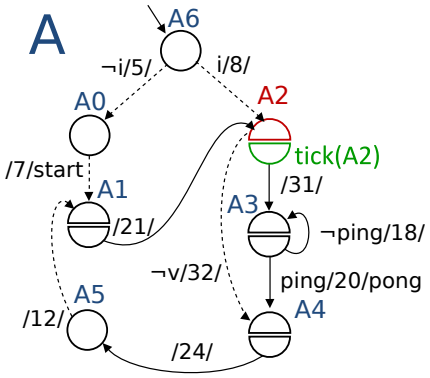


Fig. 2: A IO-BTCA A to illustrate the different features of the model. Immediate transitions are dashed arrows and non-immediate transition are plain arrows.

the same tick. As an example, on the state $A2$, if the condition $\neg v$ is true, it goes directly to $A4$. If there is no activated transition out of the surface, the control flow pauses there to wait for the clock tick. The occurrence of the clock tick switches activation to the lower half of the state, called the *depth*, from where the successive tick then is started. To express the synchronising behaviour of the clock tick we always use q for the surface and $tick(q)$ for the depth of a pause state in an IO-BTCA. This is indicated only for state $A2$ in Fig. 2 but applies to all other pause states, too.

Following the terminology of [19] we distinguish two types of execution paths in an IO-BTCA. A *sink path* starts in $entry(A)$, passes through immediate transitions ends in a pause state. An *internal path* starts the automaton in some pause state $tick(Ai)$ (the depth part) at the beginning of the tick, then activates a sequence of transitions and finally pauses in the surface of another pause state Aj.

Parallel composition of IO-BTCAs. Consider the synchronous multi-threaded composition $\mathsf{cA}\|\mathsf{cB}\|\mathsf{cC}$ shown in Fig. 3. The IO-BTCAs run concurrently and signals emitted by one machine are broadcast to the others. This may trigger a chain reaction of transition executions which are all executed in the same tick. The ticks are synchronised so that when one component pauses it stops and waits for the others to complete any sequence of enabled immediate transitions they may have. The composition $\mathsf{cA}\|\mathsf{cB}\|\mathsf{cC}$ pauses when *each* of cA, cB and cC pauses. For simplicity we look at the subsystem $\mathsf{cA}\|\mathsf{cB}$ only. Note that from the 12 possible joint configurations of $\mathsf{cA}\|\mathsf{cB}$ only 5 are actually reachable, while 7 state pairs do not align. The states which do align are indicated in Fig. 3 by the horizontal lines connecting the three automata. Without consideration of this alignment the possible maximum WCET for this example would be over-approximated $40+17+13=70$, induced by the transitions $A1 \to A2$, $B0 \to B1$ and $C0 \to CD$. But this is infeasible. As the tick lines show no two of them can

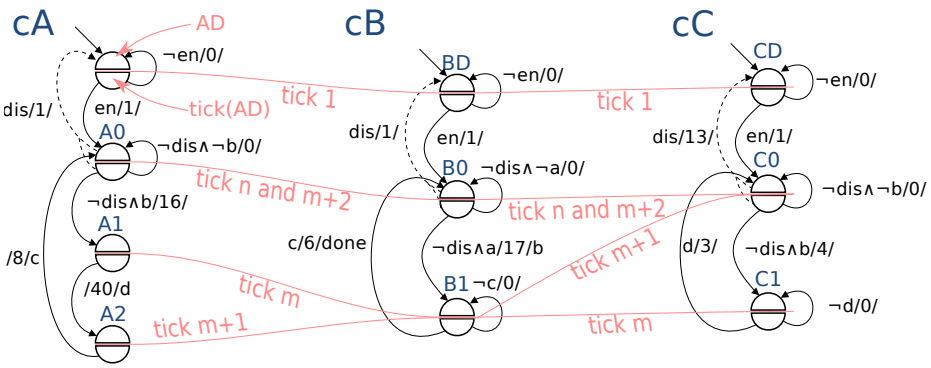


Fig. 3: Three IO-BTCAs representing the threads cA, cB and cC in our running example of Fig. 1b.

occur in the same tick. The actual WCET of cA‖cB‖cC in arbitrary environments is 43.

4 Min-Max-Plus Semantics of IO-BTCA

Here we present the semantics of IO-BTCA in terms of denotational fixed point equations. We show that the synchronous reaction behaviour and tick cost of every IO-BTCA can be described as a recursive equation system in the algebra of max-plus formal power series [3]. More details on these semantics can be found in an additional report [18].

4.1 Min-Max-Plus Algebra

Semi-ring structure. Our timing analysis will be expressed in the discrete max-plus structure over natural numbers $(\mathbb{N}_\infty, \oplus, \odot, \mathbb{0}, \mathbb{1})$ where $\mathbb{N}_\infty =_{df} \mathbb{N} \cup \{-\infty, +\infty\}$ and $\oplus$ stands for the maximum and $\odot$ for addition on $\mathbb{N}_\infty$. Both binary operators are commutative, associative and have the 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., $x \odot (y \oplus z) = x + max(y, z) = max(x+y, x+z) = (x \odot y) \oplus (x \odot z)$. However, $\oplus$ does not distribute over $\odot$, for instance, $4 \oplus (5 \odot 2) = max(4, 5+2) = 7$ while $(4 \oplus 5) \odot (4 \oplus 2) = max(4, 5) + max(4, 2) = 9$. This induces on $\mathbb{N}_\infty$ a (commutative, idempotent) semi-ring. The choice of notation[5] $\odot$ and $\oplus$ highlights the multiplicative and additive nature, respectively, of the operators. Following convention, multiplicative expressions $x \odot y$ are written also without $\odot$ simply as $x\,y$ and $\odot$ is assumed to bind more strongly than $\oplus$.

[5] In [3] the constants $-\infty$ and 0 are symbolised as ϵ and e, respectively. Alain Girault suggested to us the notation $\mathbb{0}$ and $\mathbb{1}$ which we find more suggestive.

Logical interpretation. $\mathbb{N}_\infty$ is not only a semi-ring but also a lattice structure with the natural ordering $\leq$. Meet and join, respectively, are $x \wedge y = min(x, y)$ and $x \vee y = max(x, y) = x \oplus y$. With its two infinities $-\infty$ and $+\infty$ the order structure $(\mathbb{N}_\infty, \leq, -\infty, +\infty)$ is a complete lattice. This means we can construct least and greatest solutions of fixed-point equations by taking infinite join $\bigvee$ and meet $\bigwedge$, respectively.

Max-plus algebra (over integers and real numbers) is well-known and widely exploited for discrete event system analysis (see, e.g., [3, 12]). 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 a signal 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 $\bot$. A signal with an upper bound stabilisation time of $+\infty$ on the other hand is known to become *present eventually*, though we cannot give an upper bound. This is simple logical truth, normally 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 $\bot = -\infty = \mathbb{0}$ with respect to $\wedge$ and $\vee$ follows the classical Boolean truth tables. However, a logic is not a logic without negation. The natural *implication* operation $\supset$ is given such that $x \supset y = y$ if $y < x$, $+\infty$ otherwise. This defines the residual with respect to minimum $\wedge$, i.e, $x \supset y$ is the largest element z such that $x \wedge z \leq y$. Implication internalises the ordering relation in the sense that $x \supset y = \top$ iff $x \leq y$. It generates a *negation operation* in the usual way as $\neg x =_{df} x \supset \bot$ with the property that $\neg x = \top$ if $x = \bot$ and $\neg x = \bot$ if $x \geq 0$. This turns the lattice $\mathbb{N}_\infty$ into an *intuitionistic logic* or a (complete) Heyting algebra [22]. In fact, the specific Heyting algebra $(\mathbb{N}_\infty, \wedge, \vee, \supset, \bot, \top)$ is Gödel-Dummet logic, called **LC**, which is decidable and completely axiomatised by the laws of intuitionistic logic plus the linearity axiom $(x \supset y) \vee (y \supset x)$, see [9].

Intuitionistic logic. For us both the semiring structure $(\mathbb{N}_\infty, \oplus, \odot, \mathbb{0}, \mathbb{1})$ and the logical interpretation $(\mathbb{N}_\infty, \wedge, \vee, \supset, \bot, \top)$ are equally important. The former to calculate WCET timing and the latter to express signals and reaction behaviour. Both are overlapping with the identities $\oplus = \vee$ and $\mathbb{0} = \bot$. Every element in $\mathbb{N}_\infty$ is at the same time a delay value and a constructive truth value. Every algebraic expression is at the same time the computation of a WCET and a logical activation condition. This makes min-max-plus algebra an ideal candidate to specify the constructive semantics of synchronous programming, at the exception that negation does not behave like in classical logic. Specifically, the law of the excluded middle $x \vee \neg x = \top$ fails to hold. For instance, if an Esterel program has a feedback cycle in which it emits a signal a if a is absent, this would be specified by $\neg a \supset a$. In classical logic we could prove (by case analysis) that necessarily $a = \top$, i.e., a is present (eventually). This is inconsistent with the constructive semantics of Esterel in which the program would be rejected as

non-causal. Intuitionistic Gödel-Dummet logic is causality-sensitive: $\neg a \supset a$ has an infinite number of solutions, viz. all $a \geq 0$. So, the program has no unique (bounded) response on signal a, thus explaining why it must be rejected. In this paper we do not expand on constructiveness analysis and so do not exploit the intuitionistic nature of the logic.

4.2 Formal Max-Plus Power Series

The structure $\mathbb{N}_\infty$ plays the role of scalars in the algebra of IO-BTCAs where automata are specified with formal power series over $\mathbb{N}_\infty$. These are obtained by freely adjoining to $\mathbb{N}_\infty$ a formal variable X to represent the synchronous tick that separates one instant from the next. More specifically, a *(max-plus) formal power series*, *fps* for short, is a (finite or ω-infinite) sequence

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

with $a_i \in \mathbb{N}_\infty$ and where exponentiation is repeated multiplication, i.e., $X^0 = \mathbb{1}$ and $X^{k+1} = X\, X^k = X \odot X^k$. A formal power series 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.

Such a power series may model an automaton's timing behaviour measuring the time cost to complete each tick or to reach a given state in given tick. If $a_i = \mathbb{0}$ then this means that A is not executed during the tick i and thus not contributing to the tick cost, or that a given 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 automaton 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 reaction time across all ticks.

However, A could also be used to model a signal. In this context, $a_i = \mathbb{0}$ is equivalent to the signal being absent in tick i, $a_i = \mathbb{1}$ implies that s is present from the beginning of the tick, and $a_i > \mathbb{0}$ would mean that A becomes present during tick i with a maximal delay of a_i.

The tick sequences we will generate from finite state IO-BTCA are *rational*, i.e., ultimately periodic. These have the form $A = A_\tau \oplus X^k\, A_\phi$ where the first part $A_\tau = t_0 \oplus t_1 X \oplus \cdots \oplus t_k X^k$ is a finite initial *transient* sequence and the second part $A_\phi = r_0 X \oplus \cdots \oplus r_{n-1} X^n \oplus X^n A_\phi$ a finite *recurrent* loop. For notational convenience we will write such a rational series A in short form as $A = t_0{:}t_1{:}\cdots{:}t_k{:}(r_0\, r_1\, \cdots r_{n-1})^\omega$. When $n = 1$ we call A an *ultimately constant* fps.

5 Modelling Signal-dependent WCET

We will now show how our min-max-plus algebra can fully express the synchronous semantics of a IO-BTCA, in particular how it captures signal dependency and tick alignment of the timing, at different levels of precision. Rather

than presenting a general semantic translation we illustrate the procedure using the example in Fig. 1b. We will derive for each automaton M a fps $\mathsf{wcet}(M)$ for the sequence of tick costs generated by M when started in its initial state. Moreover, we will derive for each state $S \in \mathit{states}(M)$ its worst case *activation* behaviour. This is a fps $\mathsf{wcet}(S)$ that gives for each tick the maximum waiting time for S to become active. If S is reachable in tick n then $\mathsf{wcet}(S)(n) \geq \mathbb{1}$, otherwise $\mathsf{wcet}(S)(n) = \mathbb{0}$. The value $\mathsf{wcet}(S)(i) = \top$ would indicate unbounded reachability but without a specified upper bound. These fps are defined purely algebraically by recursive equation systems following the automaton's structure. The reason why $\mathsf{wcet}(M)$ and $\mathsf{wcet}(S)$ exist as unique least fixed point solutions is that $(\mathbb{N}_\infty[X], \leq, \mathbb{0})$ is a complete partial ordering and the operations appearing in the recursion are continuous.

5.1 The WCET of IO-BTCAs.

Let us now consider the IO-BTCA cC, seen in Fig. 3. Since no state in cC is visited more than once during any tick, the cost $\mathsf{wcet}(\mathsf{cC})(i)$ of tick i is the worst case delay $\mathsf{wcet}(S)(i)$ of reaching any state $S \in \{CD, C0, C1\}$ in cC during tick i. Once we have $\mathsf{wcet}(S) = \bigoplus_i \mathsf{wcet}(S)(i)$ for each state $S \in \{CD, C0, C1\}$ we obtain the total tick cost as the sum (tick-wise maximum)

$$\mathsf{wcet}(\mathsf{cC}) = \mathsf{wcet}(CD) \oplus \mathsf{wcet}(C0) \oplus \mathsf{wcet}(C1). \tag{2}$$

Observe how the equation (2) later repeated in the equation (11) can constitute a max-plus definition of the WCET timing of our parallel system Enabled. Crucially for precision, however, this is the max-plus on formal time series and also these time series are parametric in signals.

We specify the timings $\mathsf{wcet}(S)$ of the states S inside cC in reaction to the input signals in terms of a mutually recursive system of min-max-plus recurrence equations. Here is state CD:

$$\mathsf{wcet}(CD)(0) = \mathbb{1} \tag{3}$$

$$\begin{aligned}\mathsf{wcet}(CD)(n+1) = {} & (\neg \mathit{en}(n+1) \wedge (0 \odot (\mathbb{1} \wedge \mathsf{wcet}(CD)(n)))) \\ & \oplus (\mathit{dis}(n+1) \wedge (13 \odot \mathsf{wcet}(C0)(n+1))) \\ & \oplus (\mathit{dis}(n+1) \wedge (13 \odot (\mathbb{1} \wedge \neg \mathit{dis}(n) \wedge \mathsf{wcet}(C0)(n)))).\end{aligned} \tag{4}$$

These equations are directly extracted from the structure of cC. The first equation (3) says that state CD can be reached before the first tick with max cost $\mathbb{1} = 0$. This is correct since CD is the initial state of cC and we assume that the start up is delay-free. The second equation (4) looks at the cost of activating CD in some later tick $n+1$. If CD is reachable at all in tick $n+1$, then there are only two possibilities for where the control flow can arrive from:

(i) Control has already paused in state CD in the previous tick n and signal en is absent now in tick $n+1$. This activates the delay-free self-loop on CD.
(ii) Control has reached $C0$ in the same tick $n+1$ and immediately continues along immediate $C0 \to CD$ with *additional* cost 13.

(iii) Control has paused in $C0$ in tick n with *dis* being absent, while now in tick $n+1$ signal *dis* is present.

The recurrences (3)–(4) can be lifted to fps, thus eliminating tick count n:

$$\begin{aligned}\mathsf{wcet}(CD) = \mathbb{1} \oplus (\neg en \wedge (0 \odot \mathit{tick}(\mathsf{wcet}(CD)))) \oplus (\mathit{dis} \wedge (13 \odot \mathsf{wcet}(C0))) \\ \oplus (\mathit{dis} \wedge (13 \odot \mathit{tick}(\neg \mathit{dis} \wedge \mathsf{wcet}(C0)))) \end{aligned} \tag{5}$$

where $\mathit{tick}(A) =_{df} X \odot (\mathbb{1}^\omega \wedge A)$ computes a "start time" for state A in each tick: We have $\mathit{tick}(A)(n+1) = \mathbb{1}$ if $A(n) \geq \mathbb{1}$ and $\mathit{tick}(A)(n+1) = \mathbb{0}$ if $A(n) = \mathbb{0}$.

The equations for cost series $\mathsf{wcet}(C0)$ and $\mathsf{wcet}(C1)$ are obtained similarly:

$$\begin{aligned}\mathsf{wcet}(C0) = (\neg b \wedge \neg \mathit{dis} \wedge (0 \odot \mathit{tick}(\neg \mathit{dis} \wedge \mathsf{wcet}(C0)))) \\ \oplus (d \wedge (3 \odot \mathit{tick}(\mathsf{wcet}(C1)))) \oplus (en \wedge (1 \odot \mathit{tick}(\mathsf{wcet}(CD))))\end{aligned} \tag{6}$$

$$\begin{aligned}\mathsf{wcet}(C1) = (\neg \mathit{dis} \wedge b \wedge (4 \odot \mathit{tick}(\neg \mathit{dis} \wedge \mathsf{wcet}(C0)))) \\ \oplus (\neg d \wedge (0 \odot \mathit{tick}(\mathsf{wcet}(C1)))).\end{aligned} \tag{7}$$

The simultaneously recursive equations (5)–(7) can be vectorised

$$(\mathsf{wcet}(CD), \mathsf{wcet}(C0), \mathsf{wcet}(C1)) = [\![\mathsf{cC}]\!](\mathsf{wcet}(CD), \mathsf{wcet}(C0), \mathsf{wcet}(C1)),$$

in which $[\![\mathsf{cC}]\!]$, for any fixed signals *en*, *dis*, *b*, *d* is a continuous function in the complete semi-lattice $(\mathbb{N}_\infty[X]^3, \leq, \oplus, (\mathbb{0}, \mathbb{0}, \mathbb{0}))$. Its least solution is obtained by fixed point iteration $\bigoplus_{n \geq 0} [\![\mathsf{cC}]\!]^n$ where $[\![\mathsf{cC}]\!]^0 = (\mathbb{0}, \mathbb{0}, \mathbb{0})$ and $[\![\mathsf{cC}]\!]^{n+1} = [\![\mathsf{cC}]\!]([\![\mathsf{cC}]\!]^n)$.

Approximative WCET. With (5)–(7) at hand the cost series (2) is completely specified in reaction to the signals in the environment in which cC is running. Using the equations (5)–(7) directly is possible via the equational laws of min-max-plus algebra over $\mathbb{N}_\infty[X]$ but computationally costly. Therefore we are now going to discuss two natural abstractions that introduce over-approximation on the tick costs for the benefit of computational efficiency. The first and most drastic abstraction ignores signals dependency altogether giving tick costs $\mathsf{wcet}_{abs}(M) \geq \mathsf{wcet}(M)$ and $\mathsf{wcet}_{abs}(S) \geq \mathsf{wcet}(S)$. This will give polynomial complexity. The second abstraction keeps signal dependencies for local analysis but ignores the environment. This gives local costs $\mathsf{wcet}_{loc}(M)$ and $\mathsf{wcet}_{loc}(S)$ which are worst-case over all environments. This yields more precise results, $\mathsf{wcet}_{abs}(M) \geq \mathsf{wcet}_{loc}(M) \geq \mathsf{wcet}(M)$ and $\mathsf{wcet}_{abs}(S) \geq \mathsf{wcet}_{loc}(S) \geq \mathsf{wcet}(S)$ but has NPTIME complexity.

Signal abstraction. We start with full signal abstraction where we do not bother to make any assumption on signals. Branching on signals is modelled by full non-determinism. We exploit monotonicity of $[\![\mathsf{cC}]\!]$ and abstract from the signals using the upper approximations $s \leq \top^\omega$ and $\neg s \leq \top^\omega$ for every signal $s \in \mathit{In}(\mathsf{cC})$. This simplifies the equations (5)–(7) for $\mathsf{wcet}(s)$ into equations for

approximations $\mathsf{wcet}_{abs}(s) \geq \mathsf{wcet}(s)$:

$$\begin{aligned}\mathsf{wcet}_{abs}(CD) &= \mathbb{1} \oplus \mathit{tick}(\mathsf{wcet}_{abs}(CD)) \oplus 13 \odot \mathsf{wcet}_{abs}(C0) \\ &\quad \oplus 13 \odot \mathit{tick}(\mathsf{wcet}_{abs}(C0)) \qquad (8)\\ \mathsf{wcet}_{abs}(C0) &= \mathit{tick}(\mathsf{wcet}_{abs}(C0)) \oplus\ (3 \odot \mathit{tick}(\mathsf{wcet}_{abs}(C1))) \\ &\quad \oplus (1 \odot \mathit{tick}(\mathsf{wcet}_{abs}(CD))) \qquad (9)\\ \mathsf{wcet}_{abs}(C1) &= (4 \odot \mathit{tick}(\mathsf{wcet}_{abs}(C0))) \oplus \mathit{tick}(\mathsf{wcet}_{abs}(C1)), \qquad (10)\end{aligned}$$

considering that $\top \wedge x = x$, $\mathbb{0} \oplus x = x$ and $0 \odot x = x$. This abstracted system $[\![\mathsf{cC}]\!]_{abs}$ corresponds to the automaton cC from Fig. 3 stripped of all IO signals. By direct calculations unfolding (8)–(10) we find that the sequence $[\![\mathsf{cC}]\!]^1_{abs}, [\![\mathsf{cC}]\!]^2_{abs}, [\![\mathsf{cC}]\!]^3_{abs}, \ldots$ has the limit solution

$$\mathsf{wcet}_{abs}(CD) = 0{:}14{:}14{:}16^\omega \quad \mathsf{wcet}_{abs}(C0) = \mathbb{0}{:}1{:}1{:}3^\omega \quad \mathsf{wcet}_{abs}(C1) = \mathbb{0}{:}\mathbb{0}{:}4^\omega.$$

From this we get the approximation $\mathsf{wcet}(\mathsf{cC}) \leq \mathsf{wcet}_{abs}(\mathsf{cC})$ where $\mathsf{wcet}_{abs}(\mathsf{cC}) = \mathsf{wcet}_{abs}(CD) \oplus \mathsf{wcet}_{abs}(C0) \oplus \mathsf{wcet}_{abs}(C1) = 0{:}14{:}14{:}16^\omega$. Solving the equation system for $\mathsf{wcet}_{abs}(S)$ amounts to computing the longest path, between all reachable states for a given tick. Let $\mathit{reachable}(M, n) =_{df} \{S \in \mathit{states}(M) \mid \mathsf{wcet}_{abs}(S)(n) \geq \mathbb{1}\}$ be all of M's reachable states in tick n. One can show that $\mathsf{wcet}_{abs}(S)(n{+}1)$ is the maximal length of any internal path of M starting in any state in $R_n = \mathit{reachable}(M, n)$ and ending in S. This is computable in polynomial time. However, determining the sequence of subsets $R_0, R_1, R_2, \ldots$ reachable in each tick incurs a potential combinatorial explosion. In principle, every subset of states can occur as the set R_n. As we increase the tick count, exponentially many such state combinations may appear. Hence, it is not clear if the initial transient part of a cost series $\mathsf{wcet}_{abs}(s)$ is polynomially bounded for general IO-BTCA. However, we can show it is in PTIME for the special automata generated from SCCharts such as Enabled. The special feature is that the initial states CD, BD, AD (in fact all states) have self loops in which the environment can idle the automaton for as many ticks as it wants. As a consequence, the reachability of a state is monotonic. We call these *patient* IO-BTCA.

Tick alignment abstraction. For *general* IO-BTCAs a polynomially solvable WCET problem is obtained if we not only abstract from signals but also from the tick alignment of costs. This is a single worst case value $\mathsf{wcet}_{abs}(S)[\mathbb{1}] \in \mathbb{N}_\infty$ over all ticks. First consider that $\mathit{tick}(\mathsf{wcet}_{abs}(S))[\mathbb{1}] = \mathbb{1}$ iff S is reachable from the initial state by any path and $\mathit{tick}(\mathsf{wcet}_{abs}(S))[\mathbb{1}] = \mathbb{0}$ otherwise. Thus, $\mathit{tick}(\mathsf{wcet}_{abs}(S))[\mathbb{1}]$ is computable in polynomial time. The laws $(A \oplus B)[\mathbb{1}] = A[\mathbb{1}] \oplus B[\mathbb{1}]$ and $(d \odot A)[\mathbb{1}] = d \odot A[\mathbb{1}]$ permit us to replace all references to $\mathit{tick}(\mathsf{wcet}_{abs}(S))[\mathbb{1}]$ by $\mathbb{0}$ or $\mathbb{1}$ in equation system for $[\![M]\!]_{abs}$. The result is merely a max-plus equation system in variables $\mathsf{wcet}_{abs}(S)[\mathbb{1}] \in \mathbb{N}_\infty$ which can be solved by a max path algorithms in polynomial time. This is the same as finding the max cost internal path from the set of reachable states. From the equations (8)–(10) we obtain $\mathsf{wcet}_{abs}(CD)[\mathbb{1}] = 16$, $\mathsf{wcet}_{abs}(C0)[\mathbb{1}] = 3$ and $\mathsf{wcet}_{abs}(C1)[\mathbb{1}] = 4$. The polynomial efficiency is achieved by solving the abstracted equation sys-

tem in $\mathbb{N}_\infty$ rather then solving the original system over $\mathbb{N}_\infty[X]$ and then abstracting the result. On the other hand, of course, the tick aligned solutions $\mathsf{wcet}_{abs}(CD) = 0{:}14{:}14{:}16^\omega$, $\mathsf{wcet}_{abs}(C0) = \mathbb{0}{:}1{:}1{:}3^\omega$ and $\mathsf{wcet}_{abs}(C1) = \mathbb{0}{:}\mathbb{0}{:}4^\omega$ are more informative and more precise in compositional WCET analysis.

Environment abstraction. This leads us to our second level of abstraction: Let $\mathsf{wcet}_{loc}(S)$ be the worst case under arbitrary environment signals. In general, $\mathsf{wcet}(S) \leq \mathsf{wcet}_{loc}(S) \leq \mathsf{wcet}_{abs}(S)$. Computing $\mathsf{wcet}_{loc}(S)$ is the same as solving a max cost executable path problem for each of the sets $\mathit{reachable}(M, n)$ of reachable state combinations, where we check sensitisation conditions arising from the transition guards. In a worst-case environment there is no coupling between ticks and so this satisfiability problem can be solved independently at every tick. In summary, for each tick n the feasibility of a state S being a possible starting state $S \in \mathit{reachable}(M, n)$ can be expressed by a logical expression in a polynomial number of Boolean signal statuses. The key observation again is that for patient IO-BTCA, even under signal control, the reachable set is monotonically increasing $\mathit{reachable}(M, n) \subseteq \mathit{reachable}(M, n+1)$. More concretely, by induction, if we know the set $\mathit{reachable}(M, n)$ of states reachable in tick n, then these are the feasible start states of tick $n+1$. We replace each occurrence of $\mathsf{wcet}(S)(n+1)$ in the system equations of M by $\mathbb{1}$ if $S \in \mathit{reachable}(M, n)$ and by $\mathbb{0}$ otherwise. We then search for the maximal cost feasible path beginning in any state from $\mathit{reachable}(M, n)$, taking into account the signals conditions and the signals emitted by M in this tick. Solving the Boolean satisfiability conditions can be done in NPTIME. In the other direction, it is easy to show that the computation of $\mathsf{wcet}(S)$ is NP-hard. Any SAT can be coded into a patient IO-BTCA using only immediate transition so that $\mathsf{wcet}(S) = 1$ if the SAT is satisfiable and $\mathsf{wcet}(S) = 0$, otherwise.

Contextual dependency. The sequence $\mathsf{wcet}_{loc}(\mathsf{cC})$ is obtained by local analysis and it describes the worst-case under all possible environments. For specific environments the cost may be smaller. For instance, if *en* and *dis* are both constant true, expressed by the condition $\mathit{en} \wedge \mathit{dis} = \top^\omega$, then cC cycles along transitions between CD and C0 in each tick. This yields the cost series $\mathsf{wcet}_{cond}(CD) = 0{:}14^\omega \leq \mathsf{wcet}_{loc}(CD) = 0{:}14{:}14{:}16^\omega$.

5.2 The WCET of a Composition of IO-BTCAs

The cost series $\mathsf{wcet}(\mathsf{Enabled}) = \bigoplus_{i \geq 0} \mathsf{wcet}(\mathsf{Enabled})(i)\, X^i$ of the node $\mathsf{Enabled}$ in Fig. 1b is the parallel composition (tick-wise addition) of the constituent automata's tick cost series,

$$\mathsf{wcet}(\mathsf{Enabled}) = \mathsf{wcet}(\mathsf{hC}) \parallel \mathsf{wcet}(\mathsf{cA}) \parallel \mathsf{wcet}(\mathsf{cB}) \parallel \mathsf{wcet}(\mathsf{cC}). \tag{11}$$

Following the previously defined worst case in an arbitrary environment wcet_{loc}, we calculate those abstracted series $\mathsf{wcet}_{loc}(\mathsf{hC}) = 0{:}10^\omega$, $\mathsf{wcet}_{loc}(\mathsf{cA}) = 0{:}2{:}16{:}40^\omega$, $\mathsf{wcet}_{loc}(\mathsf{cB}) = 0{:}2{:}17^\omega$ and $\mathsf{wcet}_{loc}(\mathsf{cC}) = 0{:}14{:}14{:}16^\omega$. For patient IO-BTCA the length of these sequences is polynomial.

Modelling a max-plus approach. At the top-level we are not actually interested in the cost series but merely its worst-case $\mathsf{wcet}(\mathsf{Enabled}) = \mathsf{wcet}(\mathsf{Enabled})[\mathbb{1}]$ over all ticks. Instead of computing the parallel composition of the time sequences in $\mathbb{N}_\infty[X]$ we may compose their worst-case values in $\mathbb{N}_\infty$. Specifically,

$$\begin{aligned}
&\mathsf{wcet}_{loc}(\mathsf{Enabled})[\mathbb{1}] \\
&\quad = (\mathsf{wcet}_{loc}(\mathsf{hC}) \parallel \mathsf{wcet}_{loc}(\mathsf{cA}) \parallel \mathsf{wcet}_{loc}(\mathsf{cB}) \parallel \mathsf{wcet}_{loc}(\mathsf{cC}))[\mathbb{1}] \\
&\quad \leq \mathsf{wcet}_{loc}(\mathsf{hC})[\mathbb{1}] \odot \mathsf{wcet}_{loc}(\mathsf{cA})[\mathbb{1}] \odot \mathsf{wcet}_{loc}(\mathsf{cB})[\mathbb{1}] \odot \mathsf{wcet}_{loc}(\mathsf{cC})[\mathbb{1}] \\
&\quad = 10 + 40 + 17 + 16 = 83.
\end{aligned}$$

This is the so-called *max-plus* approach [19], which takes sum of the maximal tick cost from each parallel component. This calculation can be done in linear time but incurs a loss of precision in general.

Modelling a tick alignment sensitive approach. Both the locally abstracted series $\mathsf{wcet}_{loc}(M)$ and their collapsed worst case $\mathsf{wcet}_{loc}(M)[\mathbb{1}]$ suffer from one major deficiency compared to the exact specification $\mathsf{wcet}(M)$: The local view does not account for *tick alignment*. The worst case depends on the environment sensitising in one and the same tick all the transitions whose cost adds up to the value $\mathsf{wcet}_{loc}(M)[\mathbb{1}]$. But in a parallel system the environment of M is constrained and may not be able to exercise the sequence of sensitisations to reach the worst case configuration. In order to get tighter WCET results practical approaches have used full state space exploration [1], *context-sensitive* WCET analysis [15] or iterative narrowing using flow facts generated by model checking [20], or *tick expressions* [24]. All these approaches depend on preserving some or all of the sequencing information of the IO-BTCAs and their synchronisation via signals to detect incompatibility of local states or transitions.

Indeed, for Enabled in Fig. 1b to exhibit the worst case $\mathsf{wcet}_{loc}(\mathsf{Enabled})[\mathbb{1}] = 83$ we must activate in the same tick the transitions $Disable \to Enable$ from hC, $C1 \to C0 \to CD$ from cC, $B0 \to B1$ in cB and $A1 \to A2$ in cA. However, these transitions do not align. As indicated by the horizontal tick lines in Fig. 3, it is not possible for the environment of *Enabled* to drive the automata so the states $DisableC$, $A1$, $B0$ and $C0$ become simultaneously active in the same tick.

Practically, let us define $\mathsf{clk}(S) = \top^\omega \odot \mathsf{wcet}(S)$ as the *clock* of S giving full reachability information for a state S across all ticks and depending on all signals. If $\mathsf{clk}(S)(n) = \bot = -\infty$ then S is not reachable in tick n, while if $\mathsf{clk}(S)(n) = \top = +\infty$ then S is reachable. We intersect the two clocks $\mathsf{clk}(DisableC) \wedge \mathsf{clk}(A1)$ and use the recursive definitions from the specification of hC and cA to find that $\mathsf{clk}(DisableC) \wedge \mathsf{clk}(A1) = \bot^\omega$, i.e., both clock are incompatible. We exploit this pairwise incompatibility information to run a second iteration of our local analysis, this time however, tracking the states $DisableC$ and $A1$. We use $\mathsf{wcet}_{A1}(S)$ which retains information on the dependency on (the clock of) state $A1$. It is more informative than $\mathsf{wcet}_{abs}(S)$ but less informative than $\mathsf{wcet}(S)$. Recalculating the abstraction for the full program

$$\begin{aligned}
\mathsf{wcet}(\mathsf{Enabled}) &\leq (\mathsf{wcet}_{DisableC}(\mathsf{hC}) \parallel \mathsf{wcet}_{A1}(\mathsf{cA})) \parallel \mathsf{wcet}_{abs}(\mathsf{cB}) \parallel \mathsf{wcet}_{abs}(\mathsf{cC}) \\
&= 0{:}12{:}26{:}41^\omega \parallel 0{:}2{:}17^\omega \parallel 0{:}14{:}14{:}16^\omega \;=\; 0{:}28{:}57{:}74^\omega
\end{aligned}$$

yields a tighter worst-case abstraction than the max-plus result $0{:}28{:}57{:}83^{\omega}$.

6 Related Work

The algebraic formulation of [19] for Esterel is closest to our approach. However, this does not consider the issue of tick alignment and signal dependencies. Logothetis et.al. [16] show how to instrument the compilation process of Quartz for back-annotations of WCET timing into timed Kripke structures (TKS) modelling synchronous programs. However, timing semantics is not integrated into the algebraic semantics unlike our model.

Our work may be seen in the tradition of data-flow analyses for general imperative programs. Blieberger [6] presents WCET analysis using generating functions in plus-mult linear algebra considering loop counts. However, this semantics is not developed for signal dependencies and tick alignment, unlike the proposed approach. Max-plus algebra is also used for streaming applications to model actor firing times and execution dependencies [12]. Those techniques have been used, among other things, to solve throughput evaluation. The throughput of a streaming application is comparable to the WCET of a synchronous language. More recently, those techniques were extended using iterative narrowing [13] that, we believe, follows a similar direction as the iterative feasibility analysis we presented in section 5.2.

Unlike the above references, it is essential to also consider architectural modelling for effective timing analysis. In our framework, we have assumed the precision timed architectures [10]. These architectures are non-speculative and have enabled us to focus on the nuances of synchronous programming instead of architectural modelling. However, our formulation could be extended in the future, along the lines of [8, 7]. UPPAAL is used for precise micro-architectural modelling, including the modelling of architectures with timing anomalies, as illustrated in [7]. These works consider a network of timed automata for such models, unlike a network of IO-BTCAs considered in our semantics. Hence in our formulation it will be sufficient to consider model checking using bounded integers rather than real-valued clocks, as illustrated already in [21].

7 Conclusions

Design of safety-critical systems need both functional and timing correctness. Synchronous languages offer a deterministic concurrency model that is ideal for the design of such systems. To ensure timing correctness, several WCET analysis techniques have been developed. However, the study of timing correctness, from a semantic viewpoint is lacking, which could provide a sound basis for the design of WCET analysis tools. This paper, for the first time, develops a comprehensive semantics of synchronous languages using min-max-plus Gödel-Dummett algebra. The proposed semantics is compositional and may be used to describe the WCET behaviour of an individual thread (an automaton) or the

composition of a set of threads. To facilitate precise analysis, the approach formalises the modelling of signals and the signal dependency between the threads. It also models, precisely, the tick-based lock-step execution of the threads, by formalising the *tick alignment problem* [21]. While the semantics enables precise approaches for analysis, it also facilitates abstractions and over-approximations. By abstracting a given feature, the designer may trade-off precision for scalability. Thus, the approach paves the way for the design of suitable analysis algorithms for WCET computation, that are founded on these sound semantics. In the near future, we will develop timing analysis tools for the SCCharts language by leveraging the developed semantics. We will also consider architectural modelling to support complex pipelines and memory architectures, unlike the PRET approach followed in this proposal. Another direction of future research would involve operational semantics of IO-BTCA structures and notions of simulation and equivalence among these structures unlike the fps-based semantics developed here.

8 Acknowledgment

We thank our anonymous reviewers and Insa Fuhrmann for the constructive feedback. We acknowledge the Precision-Timed Synchronous Reactive Processing (PRETSY2) project by the German Research Foundation DFG (ME 1427/6-2, HA 4407/6-2). Partha Roop acknowledges the research and study leave from Auckland University. Bruno Bodin acknowledges funding from the EPSRC grant PAMELA EP/K008730/1.

References

1. S. Andalam, P. S. Roop, and A. Girault. Pruning infeasible paths for tight wcrt analysis of synchronous programs. In *Design, Automation Test in Europe Conference (DATE), 2011*, pages 1 –6, march 2011.
2. Ch. André. Syncharts: A visual representation of reactive behaviors. *Rapport de recherche tr95-52, Université de Nice-Sophia Antipolis*, 1995.
3. F. L. Baccelli, G. Cohen, G. J. Olsder, and J.-P. Quadrat. *Synchronisation and Linearity*. John Wiley & Sons, 1992.
4. A. Benvenist, P. Caspi, S. A. Edwards, N. Halbwachs, P. Le Guernic, and R. de Simone. The synchronous languages 12 years later. *Proceedings of the IEEE*, 91(1):64 – 83, Jan 2003.
5. G. Berry. The foundations of Esterel. In *Proof, language, and interaction*, pages 425–454, 2000.
6. J. Blieberger. Data-flow frameworks for worst-case execution time analysis. *Real-Time Systems*, 22(3):183–227, 2002.
7. F. Cassez and J.-L. Béchennec. Timing analysis of binary programs with UPPAAL. In *ACSD*, pages 41–50, 2013.
8. A. E. Dalsgaard, M. Ch. Olesen, M. Toft, R. R. Hansen, and K. G. Larsen. Metamoc: Modular execution time analysis using model checking. In *OASIcs-OpenAccess Series in Informatics*, volume 15. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2010.

9. M. Dummett. A propositional calculus with a denumerable matrix. *Journal on Symbolic Logic*, 24:97–106, 1959.
10. S. A. Edwards and E. A. Lee. The case for the precision timed (PRET) machine. In *Proceedings of the 44th annual Design Automation Conference*, pages 264–265. ACM, 2007.
11. I. Fuhrmann, D. Broman, S. Smyth, and R. von Hanxleden. Towards interactive timing analysis for designing reactive systems. reconciling performace and predictability (RePP'14) satellite event of etaps'14. Technical report, Also as technical report: EECS Department, University of California, Berkeley, UCB/EECS-2014-26, 2014.
12. M. Geilen and S. Stuijk. Worst-case performance analysis of synchronous dataflow networks. In *CODES+ISSS'10*, Scottsdale, Arizona, USA, October 2010. ACM.
13. R. De Groote, P. K. F. Hölzenspies, J. Kuper, and G. J. M. Smit. Incremental analysis of cyclo-static synchronous dataflow graphs. *ACM Transactions on Embedded Computing Systems (TECS)*, 14(4):68, 2015.
14. D. Harel. Statecharts: A visual formalism for complex systems. *Science of computer programming*, 8(3):231–274, 1987.
15. L. Ju, B. K. Huynh, S. Chakraborty, and A. Roychoudhury. Context-sensitive timing analysis of Esterel programs. In *DAC'09: Proceedings of the 46th Annual Design Automation Conference*, pages 870–873, New York, NY, USA, 2009. ACM.
16. G. Logothetis, K. Schneider, and C. Metzler. Generating formal models for real-time verification by exact low-level runtime analysis of synchronous programs. In *International Real-Time Systems Symposium (RTSS)*, pages 256–264, Cancun, Mexico, 2003. IEEE Computer Society.
17. F. Maraninchi and Y. Rémond. Argos: an automaton-based synchronous language. *Computer languages*, 27(1):61–92, 2001.
18. M. Mendler, P. S. Roop, and B. Bodin. A novel wcert semantics of synchronous programs. Technical report, University of Bamberg, Nr. 101, 2016.
19. M. Mendler, R. von Hanxleden, and C. Traulsen. WCRT Algebra and Interfaces for Esterel-Style Synchronous Processing. In *Proceedings of the Design, Automation and Test in Europe Conference (DATE'09)*, Nice, France, April 2009.
20. P. Raymond, C. Maiza, C. Parent-Vigouroux, F. Carrier, and M. Asavoae. Timing analysis enhancement for synchronous programs. *Real-Time Systems*, 51:192–220, 2015.
21. P. S. Roop, S. Andalam, R. von Hanxleden, S. Yuan, and C. Traulsen. Tight WCRT analysis of synchronous C programs. *Proceedings of the 2009 international conference on Compilers, architecture, and synthesis for embedded systems - CASES '09*, page 205, 2009.
22. D. van Dalen. Intuitionistic logic. In D. Gabbay and F. Guenthner, editors, *Handbook of Philosophical Logic*, volume III, chapter 4, pages 225–339. Reidel, 1986.
23. R. von Hanxleden, B. Duderstadt, Ch. Motika, S. Smyth, M. Mendler, J. Aguado, S. Mercer, and O. O'Brien. SCCharts: Sequentially Constructive Statecharts for safety-critical applications. In *Proc. ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI'14)*, Edinburgh, UK, June 2014. ACM.
24. J. J. Wang, P. S. Roop, and S. Andalam. ILPc : A novel approach for scalable timing analysis of synchronous programs. In *CASE 2013*, 2013.