On Specifications and Proofs of Timed Circuits

Matthias Függer[1][0000−0001−5765−0301], Christoph Lenzen[2][0000−0002−3290−0674], and Ulrich Schmid[3⋆][0000−0001−9831−8583]

[1] CNRS, LMF, ENS Paris-Saclay, Université Paris-Saclay, Inria
[2] CISPA Saarbrücken
[3] TU Wien

Abstract. Given a discrete-state continuous-time reactive system, like a digital circuit, the classical approach is to first model it as a state transition system and then prove its properties. Our contribution advocates a different approach: to directly operate on the input-output behavior of such systems, without identifying states and their transitions in the first place. We discuss the benefits of this approach at hand of some examples, which demonstrate that it nicely integrates with concepts of self-stabilization and fault-tolerance. We also elaborate on some unexpected artefacts of module composition in our framework, and conclude with some open research questions.

1 Motivation and Overview

Many physical systems dealt with by computational methods today do not operate on discrete values. Examples range from electronic circuits to mechanical systems to chemical processes, which all share that analog information is continuously processed. Whereas natural and engineering sciences, in particular, control theory, have been successful in finding and using accurate models for such continuous systems, primarily based on systems of differential equations, the complexity both involved in model development and model usage is often prohibitive: Model composition and hierarchical modeling is usually difficult, and large simulation times and memory consumption as well as numerical instability typically limit the applicability of the resulting models in practice.

Discrete-valued abstractions. Applying discrete-valued abstractions for modeling continuous-valued systems is hence an attractive alternative, and much of the big success of computer science is owed to their introduction. Apart from being easily specified and understood, discrete abstractions usually involve all-digital information and finite (typically small) state-spaces that can be efficiently processed, transmitted, and stored. Among the many success stories of this approach is digital circuit design, which is one of the key enablers of modern computer systems (and also our main source of application examples): While it is out of question to perform analog simulations of the billions of transistors and other

⋆ Corresponding author (*s@ecs.tuwien.ac.at*). Supported by the Austrian Science Fund project DMAC (P32431).

analog electronic components that implement the logic gates in a modern *very-large scale integration* (VLSI) circuit, applying digital timing simulations and verification techniques is common practice.

The need for accurate timed circuit models. Given the tremendous advances in VLSI technology, with clock speeds in the GHz range and voltage swings well below 1 V [31], modeling accuracy becomes a concern [10]. Additionally, manufacturing *process, temperature, and supply-voltage* (PVT) variations cause a large variability in switching and signal propagation delays. Furthermore, reduced critical charges make the circuits more susceptible to ionizing particles [7, 13] and electromagnetic interference [40], and feature sizes in the 10 nm range also increase the likelihood of permanent errors due to manufacturing defects and wear-out [33, 43]. All these effects together make non-conservative delay predictions, which are required for digital modeling of fast synchronous circuits, difficult to obtain.

Indeed, none of these effects is adequately captured by existing timed digital circuit models. Besides the lack of modeling and analysis support for fault-tolerance, it was shown in [26] that none of the classic digital channel models, including the widely used pure and inertial delay channels [49], faithfully model the propagation of short pulses. The same is true for more advanced models like PID channels [8] (with the notable exception of involution channels [24], though). Since existing digital simulators exclusively use classic delay models, their predictions are hence not always accurate. Moreover, existing digital design tools lack an adequate support for metastability[4] [39] modeling and analysis.

Modeling approaches: state-based and state-oblivious. A natural and powerful tool for modeling such systems are transition systems. A transition system is defined by a set of states, transitions between these states, and rules how *executions*, i.e., (timed) state sequences, are generated by such a system. Transition systems can be white-box or black-box: white-box approaches try to follow the actual implementation and model the dynamics of a system's state, while black-box models just try to capture the dynamics of the system's inputs and outputs. In the latter case, states are merely used as equivalence classes of execution prefixes (histories), to abstract away individual execution prefixes that do not need to be further distinguished when capturing the system's behavior. We will refer to both variants as *state-based specifications* in the following.

On the other hand, the correct behavior of a system can be directly specified by the set of valid executions. For convenience, this is typically done in terms of an input-output function that maps a (timed) input state sequence to a set of

[4] Metastable upsets can occur in any state-holding device with discrete stable states, such as memory cells. If a new state transition is triggered before the state change caused by the previous one has settled, an intermediate output value may be observed arbitrarily late. Even Byzantine (i.e., "worst-case") fault-tolerance techniques are incapable of containing the effects of metastable upsets perfectly [22], since a signal outside the (discrete) value domain is not just an arbitrary regular signal. Note that metastability is not restricted to electrical systems. For example, an engineered genetic toggle switch [28], acting as a memory cell storing 0 or 1, was observed to exhibit metastable behavior besides its two stable states.

allowed (timed) output state sequences. We will refer to this as a *state-oblivious specification*.

While the distinction is not strict, as a state-oblivious specification is easily translated to a state-based specification with the set of states being the set of all execution prefixes, i.e., trivial equivalence classes taken as states, the approaches tend to lead to quite different formalizations and proofs. In computer science, state-based specifications have received lots of attention, and can nowadays draw from a rich set of techniques, e.g., for showing that one system implements another system via simulation relations, or reasoning about properties of compositions of systems.

A take on a state-oblivious modeling framework. In this work, we advocate the considerably less popular alternative of state-oblivious specifications and discuss some of its properties, building on the framework originally presented in [16].

In Section 2–Section 5, we review[5] the cornerstones of state-oblivious formalizations of continuous-time, discrete-valued circuits as introduced in [16]. Like in some existing approaches for reactive systems, such as Broy and Stølen's FOCUS [9], a *module* is specified directly in terms of the output behaviors that it may exhibit in response to a given input signal. As already said, this is very different from existing frameworks that follow a state-based approach, including Alur-Dill Timed Automata [4], Lamport's TLA [35], Timed IO Automatons by Keynar et. al. [32], and discrete abstractions for hybrid systems [3], as well as state-space-based control theory [36], which all act on the (sometimes uncountable) state-space of the underlying system.

We demonstrate that typical timing constraints and fault-tolerance properties (e.g. Byzantine behavior [42]) are easily expressed and dealt with in a state-oblivious framework. In Section 6, we also demonstrate that self-stabilizing systems [18], in which the initial internal state is assumed to be completely arbitrary after a catastrophic (but transient) event, can be described and proved correct appropriately.

In Section 3, we address the important issue of composition of modules. Composition has been extensively studied in state-based approaches. In general, however, it is even difficult to decide whether behavioral specifications match at interface boundaries [2]. While one can prove some generic properties about composition of state-oblivious specifications, they apply to quite restricted settings only. In Section 7, we will show that there are indeed some unexpected module composition artefacts for state-oblivious specifications when one considers more general settings. In particular, the *eventual short pulse filter* (eSPF) module introduced in [26, Sec. 7] reveals that composing modules with bounded delay in a feedback loop may result in a module with finite delay. Even worse, whereas the feedback-free composition of bounded-delay modules is always bounded delay, it turns out that the *feedback-free* composition of finite-delay modules need not

[5] A note to the reviewers: We not only strived for making our paper self-contained, but also tried to explain the concepts introduced in [16] in a more accessible way. We can adapt its length to the final page limit, however, once it has been determined.

always be finite-delay. Some conclusions and problems open for future research are provided in Section 8.

Applications beyond VLSI circuits. Although our framework emerged in the context of digital circuits [16], its applicability extends to other domains also. It needs to be stressed, though, that we do not aim at typical application domains of classic control theory. Despite some similarities, like considering systems as function transformers, our continuous-time discrete-value (typically binary) signals and their properties of interest are very different from the signals considered in either continuous or discrete control theory.

However, the idea to model continuous dynamical systems by discrete-state timed circuits has been successfully applied to genetics as well: Rather than analyzing dependencies of transcription and protein levels by means of differential equations, genetic circuit models have been used for descriptive [1, 48] and synthetic [28, 30, 46, 6] purposes. In the meantime, a body of genetic circuit design principles has been established [41, 29, 44]. For a discussion on differences between classical circuits in silicon and genetic circuits, we refer the reader to [25]. Further, as many biological systems are fault-tolerant and even self-stabilizing to a certain extent, a unified model bears the promise of cross-fertilization between different application domains. Earlier work on biologically inspired self-stabilizing Byzantine fault-tolerant clock synchronization [11, 45] is a promising example of the benefit of this approach.

2 Timed Circuit Models

Before discussing basic properties of state-oblivious specifications via some examples, we very briefly recall the standard synchronous, asynchronous, and partially synchronous timing models for specifying distributed systems. Obviously, they can be also used for modeling gates in a circuit that communicate with each other via interconnecting wires.

In *synchronous systems*, components act in synchronized lock-step *rounds*, each comprising a communication phase and a single computing step of every component. The strict constraints on the order of computing steps thus facilitate algorithms that are simple to analyze and implement, yet can leverage time to, e.g., avoid race conditions and implement communication by time [34]. Unfortunately, implementing the synchronous abstraction, e.g., by central clocking or causal relations enforced by explicit communication [5], can be too inefficient or plainly infeasible.

This fact fuels the interest in *asynchronous systems*, for which no assumptions are made on the order in which computation and communication steps are executed. The standard way of modeling asynchronous executions is to associate a local state with each component, and let a *(fair) scheduler* decide in which order components communicate and update their states (i.e., receive information and perform computation). Viewing synchrony as the temporally most ordered execution model of distributed computations, asynchronous systems are at the other extreme end of the spectrum. Since it is impossible to distinguish very slow

components from such that have suffered from a crash fault, however, proving correct asynchronous distributed algorithms is difficult and often impossible.[6]

To circumvent this problem, a number of intermediate *partially-synchronous* state-based models have been defined (e.g. [14, 19]). However, such models are less popular, and typically serve either special applications or as a vehicle to better understand the fundamental differences between synchronous and asynchronous systems.

2.1 When state-based formalizations are unnecessarily complicated

While synchronous and asynchronous systems are easy to specify within state-based frameworks, the situation becomes different for systems with more complicated timing constraints like partially synchronous systems. The challenge in allowing for general timing constraints is that the elegant and convenient separation of time and the evolution of the system state cannot be maintained. The situation becomes even more involved when the goal is to model circuits, as opposed to software-based computer systems. A major difference is that software-based systems typically reside at a level of abstraction where discrete, well-separated actions are taken naturally by an underlying machine. The evolution of the internal state of this machine is then modeled as a transition system. By contrast, real circuits are analog devices that continuously transform inputs into outputs.

We demonstrate the differences between the state-based and the state-oblivious approach at the example of the arguably simplest circuit, namely, a bounded-delay channel, as instantiated e.g. by an (ideal) wire.

A channel. We consider a binary bounded-delay first-in first-out (FIFO) channel, which has a single input port (= connector) and a single output port. Whereas such channels are also employed in various state-based models, they are usually *part* of the model and typically also the only means of communication. By contrast, we describe the channel as an object in a (to-be-defined) model.

Informally, we require the following: The *input port* is fed by an input signal given as a function $\text{IN} : \mathbb{R} \to \{0, 1\}$. Note that the restriction to binary-valued signals is for simplicity only and could be replaced by arbitrary discrete ranges. The reason why the domain of IN is $\mathbb{R}$ instead of, e.g., the non-negative reals $\mathbb{R}_0^+$, will be explained later. Typically, some further restrictions are made on (input) signals for modeling real circuits, e.g., only a finite number of transitions within each finite interval. For each input signal, the *output port* produces an output signal such that: (i) for each input transition there is exactly one output transition within some time $d > 0$, and (ii) output transitions occur in the same temporal order as their corresponding input transitions.

The state-based approach: the channel as a transition system. A state-based description would model the state of the channel at some time t, as well as the rules for transitioning between states. Obviously, this would allow us to infer the correct behavior of the channel at times greater than t as valid traces of this

[6] Consensus [42], a basic fault-tolerance task, can be solved deterministically in synchronous systems, but not in asynchronous systems [21].

transition system. However, a state-based description would be at odds with our goal of a simple and modular specification of the system:

- Both a physical wire connecting sender and receiver and reliable multi-hop wireless channel are implementations of our channel. Specifications with one of them in mind may differ significantly.
- The state space of the channel would be infinitely large, as it must be able to record an arbitrarily long sequence of alternating input transitions that may have occurred within $[t-d,t]$
- The strategy of breaking down a difficult-to-describe state-based description into smaller building blocks does not help here.

The above problems become even more pronounced when it comes to more interesting modules. Even if we were not discouraged by the above difficulties and went for a state-based definition of the channel, e.g. in terms of Timed I/O Automata [32], we argue that the original advantage of a state-based approach would be lost: the canonical description of the global state of the system as the product of the components' states.

The state-oblivious approach: the channel as an input-output function. We conclude that our preferred option is (i) to treat the channel as a blackbox, and (ii) not to bother with finding states, i.e., equivalence classes of histories, in the first place. That is, we infer the possible output not from some internal state, but rather directly from the input history. By the nature of a channel, however, the feasible output values at time t cannot be determined from $\textsc{in}$ alone: the *output* at times smaller than t enters the bargain as well. Hence, we naturally end up directly relating input and output signals: For each (possible) input signal $\textsc{in}$, there must be a non-empty *set* of *feasible* output functions $\phi(\textsc{in})$, where the *module specification* ϕ maps inputs signals to such feasible output signals. Note that we allow the adversary to choose which of the feasible output signals a module generates in some execution, i.e., we just assume non-determinism here.[7] This also allows to express any given restriction on the inputs, e.g., one that is considered suitable for a given module, simply by permitting *any* output signal for input signals that violate such a restriction.

A state-oblivious specification of our channel can be given in terms of an input-output function ϕ. For every input signal $\textsc{in}$, the output signal $\textsc{out}$ is feasible, i.e., $\textsc{out} \in \phi(\textsc{in})$ if $\textsc{out}(t) = \textsc{in}\left(\delta^{-1}(t)\right)$, where the delay function $\delta : \mathbb{R} \to \mathbb{R}$ is continuous, strictly increasing (hence invertible), and satisfies $t \le \delta(t) \le t + d$ for all $t \in \mathbb{R}$.

Note carefully that, as we model signals as functions of real-time, we do not need special signal values (as in FOCUS [9]) that report the progress of time, and relating different (input, output) signals to each other becomes simple.

It remains to explain why we chose the whole set of reals $\mathbb{R}$ as the (time) domain of our input and output functions. Again, in principle, nothing prevents us from using functions $\mathbb{R}_0^+ \to \{0,1\}$. For the considered channel, it would be reasonably easy to adapt the description: $\textsc{out} \in \phi(\textsc{in})$ is arbitrary on $[0, \delta(0))$

[7] Whereas one could extend our framework to restrict the adversary here, e.g., to capture probabilistic choice, we will not consider this possibility in this paper.

and $\text{OUT}(t) = \text{IN}(\delta^{-1}(t))$ on $[\delta(0), \infty)$, for some continuous, strictly increasing delay function $\delta : \mathbb{R}_0^+ \to \mathbb{R}_0^+$ satisfying that $t \leq \delta(t) \leq t + d$ for all $t \in \mathbb{R}_0^+$.

However, given that our "equivalent" to the channel's state is its input history, it is more natural to rely on input signals with a time domain that contains $[-d, \infty)$ when specifying feasible outputs on $\mathbb{R}_0^+$. Extending the range by a finite value only would not cover all possible values of d, though. Moreover, there are modules whose output may depend on events that lie arbitrarily far in the past, e.g., a memory cell. For simplicity and composability, picking $\mathbb{R}$ as domain is thus preferred here. We remark that this convention does not prevent suitable initialization of a module, say at time $t = 0$, however.

3 Composition

In the previous section, we demonstrated the use of state-oblivious specifications in terms of directly providing input-output functions ϕ for a simple channel. In general, we define:

Definition 1 (Module). *A* signal *is a function from* $\mathbb{R}$ *to* $\{0, 1\}$*. A* module M *has a set of* input ports $I(M)$ *and a set of* output ports $O(M)$*, which are the connectors where input signals are supplied to* M *and output signals leave* M*. The* module specification ϕ_M *maps the input signals* $(\text{IN}_p : \mathbb{R} \to \{0, 1\})_{p \in I(M)}$ *to sets of allowed output signals* $(\text{OUT}_p : \mathbb{R} \to \{0, 1\})_{p \in O(M)}$*. An* execution *of a module is a member of the set*

$$\left\{ \left((\text{IN}_p)_{p \in I(M)}, \phi_M((\text{IN}_p)_{p \in I(M)}) \right) \mid (\text{IN}_p : \mathbb{R} \to \{0, 1\})_{p \in I(M)} \right\} .$$

Note that we typically assume that modules are *causal*, i.e., that images of ϕ_M for two inputs that are identical until time t are identical until time t.

Specifying a module M this way, i.e., by providing ϕ_M, can either be viewed as stating an assumption, in the sense that it is already known how to build a module with the respective behavior, or as stating a problem, i.e., expressing a desired behavior of a module that still needs to be built. We call a module specified this way a *basic module*.

Implementing such a module can be done in two different ways: (i) directly within a target-technology, which leaves the scope of our modeling framework, or (ii) by decomposition into smaller modules within the modeling framework.

Let us now formalize what the latter means in the context of our approach. Intuitively, we will take a set of modules and connect their input and output ports to form a larger *compound module*, whose inputs and outputs are subsets of the ports of these modules (cp. Figure 1). The input-output function of the compound module is then derived from the ones of the submodules and their interconnection.

Definition 2 (Compound module). *A compound module* M *is defined by:*

1. *Decide on the sets of* input ports $I(M)$ *and* output ports $O(M)$ *of* M*.*

2. *Pick the set* S_M *of* submodules *of* M*. Each submodule* $S \in S_M$ *has input ports* $I(S)$*, output ports* $O(S)$*, and a specification* ϕ_S *that maps tuples* $(\text{IN}_p : \mathbb{R} \to \{0,1\})_{p \in I(S)}$ *of functions to sets of tuples* $(\text{OUT}_p : \mathbb{R} \to \{0,1\})_{p \in O(S)}$ *of functions, satisfying the following well-formedness constraints:*
   - *For each output port* $p \in O(M)$*, there is exactly one submodule* $S \in S_M$ *such that* $p \in O(S)$*.*
   - *For each input port* $p \in I(S)$ *of some submodule* $S \in S_M$*, either* $p \in I(M)$ *or there is exactly one submodule* $S' \in S_M$ *so that* $p \in O(S')$*.*
3. *For each* $(\text{IN}_p : \mathbb{R} \to \{0,1\})_{p \in I(M)}$*, we require* $(\text{OUT}_p : \mathbb{R} \to \{0,1\})_{p \in O(M)} \in \phi_M\left((\text{IN}_p)_{p \in I(M)}\right)$ *iff there exist functions* $(f_p : \mathbb{R} \to \{0,1\})_{p \in \bigcup_{S \in S_M} I(S) \cup O(S)}$ *with*
   - $\forall S \in S_M : (f_p)_{p \in O(S)} \in \phi_S((f_p)_{p \in I(S)})$*;*
   - $\forall p \in I(M) : f_p = \text{IN}_p$*; and*
   - $\forall p \in O(M) : f_p = \text{OUT}_p$*.*

Note that choices are made only in Steps 1 and 2, whereas ϕ_M is defined implicitly and non-constructively in Step 3. Informally, the latter just says that any execution, i.e., any pair of input and output signals, of M that leads to feasible executions of all submodules, must be in ϕ_M.

Example 1 (Oscillator). Figure 1 shows an example compound module: a simple resettable digital oscillator. It is composed of an inverter, an AND gate, and a fixed unit delay channel, i.e., a FIFO channel with $\delta(t) := t + 1$. Both gates operate in zero-time, i.e., all delays have been lumped into the FIFO channel.

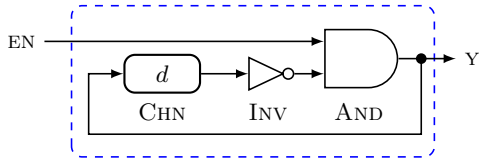

Fig. 1: Compound oscillator module.

The module's behavior is characterized by the fact that it oscillates at its output Y = CHN IN while input EN is 1, and outputs a constant 0 while EN is 0. Up to time 4, the signal trace depicted in Figure 2 shows part of a correct execution of the oscillator module.

The same conceptual design of a negative feedback-loop with delay was used by Stricker et al. [46] in the context of genetic circuits, for synthesizing a genetic oscillator in *Escherichia coli* with an output period in the order of an hour. Figure 3 depicts the genetic design of the feedback-loop of the simplified (second) design proposed by Stricker et al. It shows the DNA segement that is introduced into the bacterial host. The DNA comprises of a promoter (bold arrow in the figure) and a downstream lacI gene (flanked by a ribosome binding site and a

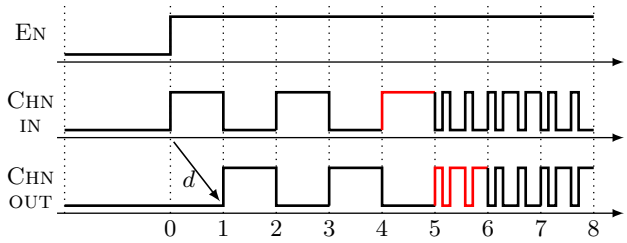

Fig. 2: Execution of oscillator module with transient channel fault (signal mismatch marked red).

terminator that are not shown for simplicity). The lacI gene is transcribed and translated into LacI protein. The promoter is activated if no inhibiting LacI proteins are present (shown as an inhibitory arrow from lacI to the promoter) and externally introduced IPTG molecules are present (not shown in the figure, and assumed to be present throughout). The activation of the promoter leads to transcription and subsequent translation of the downstream lacI gene, resulting in increasing LacI protein levels, which then inactivate the promoter. Only when the concentration of the LacI protein has fallen to a sufficiently low level due to degradation and dilution, the promoter becomes active again. The result is an oscillation of the LacI protein concentration.

A note on state-oblivious specifications: the absence of an initial state. In the previous section, we have argued that, when specifying the channel input-output behavior in a state-oblivious way, we resort to input output signals as functions $\mathbb{R} \to [0, 1]$. In this section, we followed this approach in Definitions 2 and 1. We will later see (in Section 5) that such specifications are also well-suited for specifying so-called self-stabilizing systems.

However, state-oblivious specifications also introduce difficulties that lead to open research questions. More specifically, a useful vehicle for showing that some module implements another one in classical state-based frameworks is by induction on a sequence of input events, starting from some initial state. Simulation and bi-simulation relations are proved this way, with implications on what can be said about using one module instead of the other. These proof techniques fail in our case, however, since signals are defined on the time domain $\mathbb{R}$, without an initial time and "state". While one can argue that induction from a common

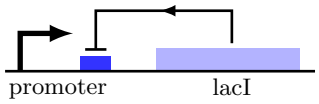

Fig. 3: Negative feedback loop of a genetic oscillator presented by Stricker et al. [46].

time, say 0, can be done into the positive an negative direction, questions about what this means for one module replacing/implementing another are open.

4 Modeling Permanent Faults

Viewed from an outside perspective, all that a faulty module can do is deviating from its specification. Depending on the type of faults considered, there may or may not be constraints on this deviation. In other words, a faulty module M simply follows a *weaker* module specification than ϕ_M, i.e., some module specification $\bar{\phi}_M$ such that $\forall (\text{IN}_p)_{p\in I(M)} : \bar{\phi}_M\left((\text{IN}_p)_{p\in I(M)}\right) \supseteq \phi_M\left((\text{IN}_p)_{p\in I(M)}\right)$.

Definition 3 (Crash and Byzantine fault types). *For the fault type* crash faults *[20], a faulty component simply ceases to operate at some point in time. In this case, $\bar{\phi}_M((\text{IN}_p)_{p\in I(M)})$ can be constructed from $\phi_M((\text{IN}_p)_{p\in I(M)})$ by adding, for each $(\text{OUT}_p)_{p\in O(M)} \in \phi_M((\text{IN}_p)_{p\in I(M)})$ and each $t \in \mathbb{R}$, the output signal*

$$\left(t' \in \mathbb{R} \mapsto \begin{cases} \text{OUT}_p(t') & \text{if } t' < t \\ \text{OUT}_p(t) & \text{else} \end{cases}\right)_{p\in O(M)}$$

to $\bar{\phi}_M((\text{IN}_p)_{p\in I(M)})$. This just keeps ("stuck-at") the last output value before the crash.

The fault-type of Byzantine faults *[42] is even simpler to describe: The behavior of a faulty module is arbitrary, i.e., $\bar{\phi}_M$ is the constant function returning the set of* all *possible output signals, irrespective of the input signal.*

Having defined a faulty type $\bar{\phi}_S$ for a module S accordingly, it seems obvious how to define a fault-tolerant compound module: A compound module M with submodules S_M tolerates failures of a subset $F \subset S_M$, iff $\phi_M = \bar{\phi}_{M,F}$, where $\bar{\phi}_{M,F}$ is the specification of the compound module in which we replace each submodule $S \in F$ by the one with specification $\bar{\phi}_S$.

Example 2 (Fault-tolerant 1-bit adder module). Figure 4 shows an example of a 1-bit adder module. It is built from three (zero-time) 1-bit adder submodules, a (zero-time) majority voter, and FIFO channels with maximal delay d connecting the module's inputs to the adder submodules. The channels account for the module's propagation delay and potentially desynchronized arrivals of input transitions at the submodules. If the module inputs have been stable for d time, however, its output yields the sum of the two inputs, tolerating failure of any one of its three adder submodules and the associated input channels.

While this definition of a fault-tolerant compound module can be useful, it is very restrictive. For instance, our adder compound module cannot tolerate a failure of the majority voter that computes the output. More generally, no matter how a compound module M is constructed, it can never tolerate even a single crash failure of an arbitrary submodule S, unless M is trivial: If S has an output port in common with M, i.e., if S generates this output for M, the only

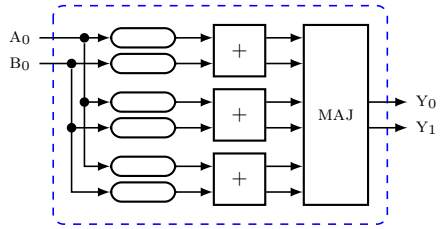

Fig. 4: Fault-tolerant 1-bit adder.

possible guarantee M could make for this output port is a fixed output value at all times, as this is what the crash of S would lead to.

To address this issue, we introduce the concept of a *fault-tolerant implementation* of a module.

Definition 4 (Fault-tolerant implementation). *We say that module M* implements *module M' iff*

$$\forall (\mathrm{IN}_p)_{p\in I(M)} : \phi_M\left((\mathrm{IN}_p)_{p\in I(M)}\right) \subseteq \phi_{M'}\left((\mathrm{IN}_p)_{p\in I(M')}\right) .$$

This requires that $I(M) = I(M')$ and $O(M) = O(M')$. Similarly, for a given fault type $\bar{\cdot}$, M is an implementation of M' that tolerates failures of $F \subset S_M$ *iff*

$$\forall (\mathrm{IN}_p)_{p\in I(M)} : \bar{\phi}_{M,F}\left((\mathrm{IN}_p)_{p\in I(M)}\right) \subseteq \phi_{M'}\left((\mathrm{IN}_p)_{p\in I(M')}\right) ,$$

*where $\bar{\phi}_{M,F}$ is defined according to the fault type. Finally, M is an f-*tolerant implementation of M'*, iff it tolerates faults of $F \subset S_M$ for any F satisfying $|F| \leq f$.*

Example 3 (Fault-tolerant adder). For an adder implementation that is 1-tolerant to Byzantine faults (and thus also any other fault type), *triple-modular redundancy* (TMR) can be used. Here, not just the adders, but also the pair of input and output signals is triplicated. Moreover, the single majority voter at the adder outputs is replaced by three majority voters at the adder inputs: Since they vote on the replicated input signals, we can guarantee that all three adders receive identical inputs if no voter fails, whereas two adders receive identical inputs and produce identical outputs if one voter is faulty. Note that relaxing the specification and using an implementation relation is necessary here, as otherwise the same reasoning as before would prevent a 1-tolerant solution.

The problem of developing a fault-tolerant implementation of an oscillator was addressed in the DARTS project [23, 27]: Using a predecessor of the proposed modeling framework, it was shown that a circuit comprising $3f+1$ tick generator nodes, in which the output of each node is fed back as input to all nodes (including the node itself), tolerates up to f Byzantine faulty nodes. The circuitry of a single tick generator node for $n = 4$ and $f = 1$ is depicted in Figure 5. Informally, it counts the difference of clock transitions generated by itself (Local

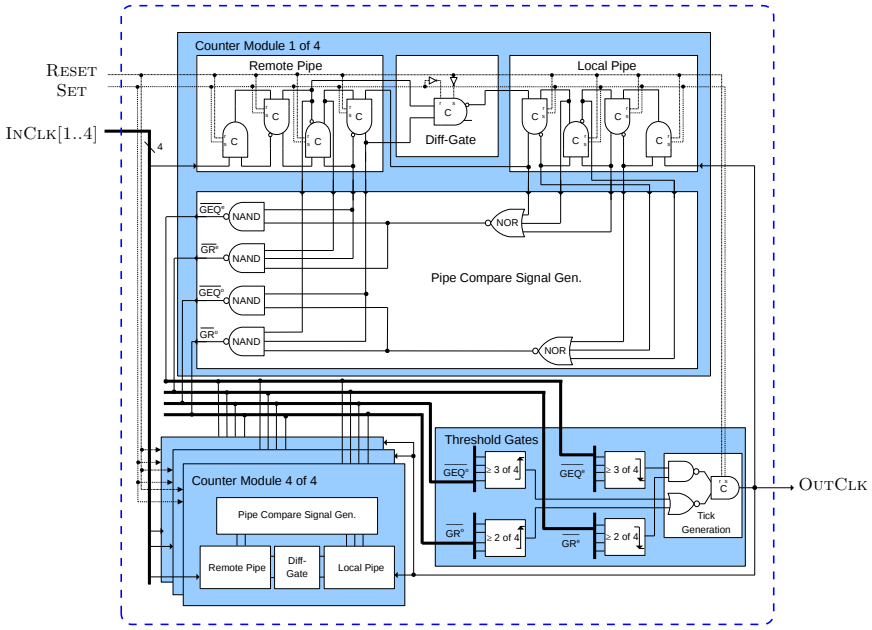

Fig. 5: Node of the fault-tolerant DARTS oscillator.

Pipe) and those received from other nodes (Remote Pipe) by means of Counter Modules implemented via elastic pipelines [47]. When sufficiently many nodes (not all, since there might be a fault) are not too far behind (determined by the Threshold Gates), it generates a new local clock transition (Tick Generation).

5 Modeling Transient Faults and Self-Stabilization

Transient faults are assumed to be temporary in nature, in the sense that the *cause* of the fault eventually vanishes. Suitable recovery techniques can hence be used to resume correct operation later on. The most extreme requirement is *self-stabilization* [12], where the system must eventually resume correct operation even after all of its components experienced arbitrary transient faults. Since the latter results in arbitrary states, this is equivalent to requiring that the system re-establishes correct operation from arbitrary initial states in finite time. The maximal time it may take to do so is called *stabilization time*.

Self-stabilization plays a crucial role in mission-critical systems, where even the assumption that a certain fraction (e.g., less than a third, as in DARTS) of the subcomponents can fail is too optimistic, or for applications that cannot afford the amount of redundancy needed for fully masking faults. Unsurprisingly, self-stabilization and related concepts also play a vital role in biological systems. For example, Albert and Othmer [1] modeled part of the control circuit that regulates gene expression in the fruit fly Drosophila Melanogaster by a binary

circuit, and observed that a considerable number of initial circuit states finally lead to the wild-type stable state. For the lobster heart, it has been established that it is, in essence, self-stabilizing even in the presence of ongoing Byzantine behavior [11, 45].

Transferring the idea of self-stabilization to our state-oblivious framework requires some effort, but we will demonstrate that it can be integrated very well. Most notably, there is no notion of a state (besides from trivial states), which means that we cannot define self-stabilization in the conventional state-based manner.

The first important observation underlying input-output functions in a state-oblivious specification of self-stabilization is that, since a basic module specification describes the desired behavior *from the viewpoint of an external observer*, we consider a module correct even if it merely *seems* to be operating correctly in terms of its input-output behavior. In other words, it does not matter whether the module internally operates as intended, as long as it produces correct results.

Second, when defining basic modules (Definition 1), we only resorted to input and output signals from $\mathbb{R} \to [0,1]$. The input-output function ϕ_M of a module M then maps input signals to allowed output signals. For modules that are intended to be self-stabilizing (or that suffer from transient faults), this is not anymore convenient since they are not either correct or faulty during all of the execution. Merely, we would like to define how they should behave if they were *correct during a time interval* $[t^-, t^+]$.

Redefining correctness for transient faults. An immediate solution to this is to define ϕ_M on all signal restrictions to all sub-intervals $I = [t^-, t^+] \subseteq \mathbb{R}$. To make such interval-restrictions explicit, we will sometimes write σ^I, IN^I, OUT^I, E^I etc. Note that such intervals I could also be open (t^-, t^+), closed $[t^-, t^+]$, or half-open, but are always contiguous.

Definition 5 (Basic module—interval-restricted specification). *An interval-restricted execution E^I of a basic module is correct during $I = [t^-, t^+]$ if its interval-restricted output signals OUT^I are within the image $\phi_M(\text{IN}^I)$ of its interval-restricted input signals.*

We termed this the *interval-restricted specification*, since it requires the definition of ϕ_M on all these sub-intervals.

However, care has to be taken: the input-output function ϕ_M has to be restricted to ensure that it adheres to an intuitive notion of correctness. For example, we expect an execution of M that is correct within $[t^-, t^+]$ to be also correct within all subintervals of $[t^-, t^+]$. One possibility is to add all those restrictions explicitly. Indeed, such specifications are powerful in expressiveness [16], but at the same time our experience in the early stages of [16] was that using this approach is tedious for simple modules, and practically guarantees mistakes for complex modules. The reason for this is that the subset-closedness of the correctness definition is easily violated in an interval-restricted specification even for simple modules like channels.

We thus primarily resort to another definition, which does not change the domain of ϕ_M, i.e, where the domain of all signals is $\mathbb{R}$, which we call a *definition*

by extension. With this definition, correctness is subset-closed for time intervals by construction (see [16, Lem. 3.3]), such that the natural notion of correctness is also guaranteed by construction.

Definition 6 (Basic module, correct during time interval—definition by extension). *An interval-restricted execution E^I of a basic module M is correct during $I = [t^-, t^+]$, iff there is a (complete, i.e., with time domain $\mathbb{R}$) execution E' of a basic module M such that: (i) the input and output signals of E^I and E' are identical during $[t^-, t^+]$, and (ii) for execution E', letting i be the input signals and o the output signals of E', $o \in \phi_M(i)$.*

If not stated otherwise, we will resort to the definition by extension for basic modules. For ease of notation, we will, however, extend ϕ_M to input and output signals with time domains that are sub-intervals $[t^-, t^+]$ of $\mathbb{R}$ in the following: Writing $\text{OUT}^I \in \phi_M(\text{IN}^I)$ where $\text{IN}^I, \text{OUT}^I$ have time domain $I = [t^-, t^+]$ is just a short-hand notation for: For any execution E of M that behaves according to $\text{IN}^I, \text{OUT}^I$ during I, basic module M is correct during I.

Definition 7 (Extendible module). *We say that a basic module is* extendible*, if its input-output function ϕ_M has the properties of a definition by extension. That is, executions that are correct on a subinterval can be extended to executions that are correct on $\mathbb{R}$.*

The above definition of correctness introduced above also implies that a sub-execution on some interval $[t^-, t^+] \subset \mathbb{R}$ that is considered correct can be extended to a (complete) correct execution on $\mathbb{R}$. This is natural for basic modules, but inappropriate for self-stabilizing compound modules: these take the role of algorithms, and making this a requirement would be equivalent to disallowing transient faults—or, more precisely, to implicitly turn them into persistent faults. To illustrate this issue, consider again the compound module implementing the oscillator shown in Figure 1 and the signal trace shown in Figure 2: The execution segment during time interval $[6, 8]$ must be considered correct, since it fulfills all input-output constraints of the involved circuit components during this interval. We know, however, that such a high-frequency oscillation can never occur in a (complete) correct execution of the compound module on $\mathbb{R}$, for which the only possible oscillator frequency is one transition per time unit.

To allow for a meaningful notion of self-stabilization, we will hence treat compound modules differently: When analyzing their stabilizing behavior, we assume that all sub-modules themselves operate correctly, whereas the "convergence" of the compound module's externally visible behavior to a correct one must be enforced. This is captured by defining correct executions of compound modules on time intervals $[t^-, t^+] \subset \mathbb{R}$ by the same process as in Definition 2, except that we replace $\mathbb{R}$ by $[t^-, t^+]$.

Definition 8 (Compound module—interval-restriced specification). *For any $I = [t^-, t^+]$ and any interval-restricted input signal $(\text{IN}_p^I)_{p \in I(M)}$, we require that the interval-restricted output signal $(\text{OUT}_p^I)_{p \in O(M)} \in \phi_M\big((\text{IN}_p^I)_{p \in I(M)}\big)$*

iff there exist interval-restricted input and output signals $(f_p^I)_{p\in\bigcup_{S\in S_M} I(S)\cup O(S)}$ *for all submodules* S *of* M *so that all the properties below hold:*

- $\forall S \in S_M : (f_p^I)_{p\in O(S)} \in \phi_S((f_p^I)_{p\in I(S)})$
- $\forall p \in I(M) : f_p = \text{IN}_p^I$
- $\forall p \in O(M) : f_p = \text{OUT}_p^I$

Note that this definition is recursive; we can iteratively extend all module specifications to inputs on arbitrary intervals $[t^-, t^+] \subseteq \mathbb{R}$, starting from the specifications of basic modules for inputs on $\mathbb{R}$.

With these definitions in place, we can now proceed to defining a suitable notion of self-stabilization in our framework.

Definition 9 (Self-stabilizing implementation). *A module* M *is called a* T-stabilizing implementation of module M', *iff* $I(M) = I(M')$, $O(M) = O(M')$ *and, for all* $I = [t^-, t^+] \subseteq \mathbb{R}$ *with* $t^+ \geq t_- + T$, $I' = [t^- + T, t^+]$ *and each* $(\text{OUT}_p^I)_{p\in O(M)} \in \phi_M\left((\text{IN}_p^I)_{p\in I(M)}\right)$, *it holds that*

$$(\text{OUT}_p^{I'})_{p\in O(M)} \in \phi_{M'}\left((\text{IN}_p^{I'})_{p\in I(M')}\right).$$

Informally, cutting off the first T *time units from any interval-restricted execution of* M *must yield a correct interval-restricted execution of* M'.

Module M *is a* self-stabilizing implementation of M', *iff it is a* T*-stabilizing implementation of* M' *for some* $T < \infty$.

Example 4 (Self-stabilization). According to Definition 9, the oscillator implementation from Figure 1 is not self-stabilizing, as illustrated by Figure 2: After a transient fault of the channel component during time $[5, 6]$, all circuit components operate correctly again from time 6 on, but the behavior of circuit output $\text{Y} = \text{CHN IN}$ never returns to the behavior of Y that could be observed in an execution on $\mathbb{R}$.

For a positive example, recall the 1-bit adder depicted in Figure 4. Its self-stabilization properties follow, without the need of a custom analysis, from a general principle (called forgetfulness), which will be introduced in the next section.

6 Example: A Self-Stabilizing Oscillator

In view of the state-obliviousness and generality of our modeling framework, one might ask whether it indeed allows to derive meaningful results. As a proof of concept, we will thus elaborate more on self-stabilizing compound modules.

First, we will formalize the statement that if a compound module M is made up of submodules S whose output at time t depends only on the input during $[t - T_S, t]$ (for $T_S \in \mathbb{R}_0^+$) and contains no feedback-loops (like, e.g., the adder in Figure 4, but unlike the oscillator in Figure 1), then M is self-stabilizing. Interestingly, this result sometimes *does* also apply to systems that do have internal feedback loops; this holds true whenever we can contain the loop in a submodule and (separately) show that it is self-stabilizing.

Definition 10 (Forgetfulness). *For $F \geq 0$, module M is* F-forgetful *iff:*

1. *For any $I = [t^-, t^+] \subseteq \mathbb{R}$ with $t^+ \geq t^- + F$, pick any interval-restricted output $(\text{OUT}_p^I)_{p \in O(M)} \in \phi_M\left((\text{IN}_p^I)_{p \in I(M)}\right)$.*
2. *For each input port $p \in I(M)$, pick any input signal $\text{IN}'_p : \mathbb{R} \to \{0,1\}$ so that IN'_p restricted to I equals IN_p^I.*
3. *Then $\left(\text{OUT}'_p : \mathbb{R} \to \{0,1\}\right)_{p \in O(M)} \in \phi_M\left((\text{IN}'_p : \mathbb{R} \to \{0,1\})_{p \in I(M)}\right)$ exists so that for all output ports $p \in O(M)$ the restrictions of OUT_p and OUT'_p to the interval $[t^- + F, t^+]$ are equal.*

In other words, the output of a F-forgetful module during $[t^- + F, t^+]$ reveals no information regarding the input during $(-\infty, t^-)$.

Example 5. A simple example of a d-forgetful module is a FIFO channel with maximum delay d.

Definition 11 (Feedback-free module). *Let the* circuit graph *of a compound module M be the directed graph whose nodes are the submodules S_M of M, and for each output port p of $S \in S_M$ that is an input port of another submodule $S' \in S_M$, there is a directed edge from S to S'. We say M is* feedback-free *iff all its submodules are forgetful and its circuit graph is acyclic.*

One can then show that feedback-free compound modules made up of forgetful submodules are self-stabilizing:

Theorem 1 ([16], Theorem 3.7). *Given a feedback-free compound module M, denote by $\mathcal{P}$ the set of paths in its circuit graph. Suppose that each submodule $S \in \mathcal{S}_M$ is F_S-forgetful for some $F_S \in \mathbb{R}_0^+$. Then, M is F-forgetful with*

$$F = \max_{(S_1,\dots,S_k) \in \mathcal{P}} \left\{ \sum_{i=1}^{k} F_{S_i} \right\} .$$

Using this theorem (possibly recursively applied, in the case of compound modules made up of compound submodules), one can show that a given feedback-free compound module is forgetful. Moreover, for such a module M, it is sufficient to show that it behaves like another module M' in correct executions (i.e., those on $\mathbb{R}$, rather than on certain time intervals) for proving that M is a self-stabilizing implementation of M'.

Corollary 1. *Suppose that compound module M satisfies the prerequisites of Theorem 1. Moreover, for a module M' with $I(M') = I(M)$ and $O(M') = O(M)$, assume that: For all input signals $(\text{IN}_p : \mathbb{R} \to \{0,1\})_{p \in I(M)}$, it holds for its output signals that $\phi_M\left((\text{IN}_p)_{p \in I(M)}\right) \subseteq \phi_{M'}\left((\text{IN}_p)_{p \in I(M)}\right)$. Then, M is a self-stabilizing implementation of M'.*

These results ensure that self-stabilization follows without further ado not only in trivial cases where an erroneous state is *instantaneously* forgotten and overwritten by new input. By using compound modules in a hierarchical manner,

one can encapsulate the heart of a proof of self-stabilization in the appropriate system layer and separate aspects from different layers that are unrelated. This plays along nicely with the standard approach for proving self-stabilization of complex systems, which is to establish properties of increasing strength and complexity in a bottom-up fashion, advancing from very basic aspects to high-level arguments.

Beyond feedback-free compound modules. Unfortunately, however, it is not hard to see that if (the circuit graph of) a compound module M is not feedback-free, self-stabilization of M does not necessarily follow from the fact that all sub-modules are forgetful. An example of such a circuit is the oscillator in Figure 1. There are, however, circuits with feedback loops that stabilize.

Example 6. Figure 6 shows a self-stabilizing variant of the oscillator from Figure 1 (without enable input). The module consists of (i) a watchdog-timed memory-cell MEM whose output MEM OUT $= Y$; the output is 1 at time t iff there is a time $t' \in (t-T, t)$ where output $Y(t') = 0$ and input $X(t') = 1$, (ii) a succeeding fixed delay channel with delay $d \leq T$, and (iii) an inverter. Note that (i) implies that the set $\{t \in \mathbb{R}_0^+ \mid Y(t) = 0\}$ is closed.

We now demonstrate how to formalize this example and its proof in our state-oblivious framework. The (basic) module OSC has no input and one output Y. Recall that, for basic modules, specifications involve defining ϕ_M on executions on $\mathbb{R}$ only. Hence, the module specification ϕ_{osc} is fully defined by deciding whether some function $(\text{OUT}_Y : \mathbb{R} \to \mathbb{R}) \in \phi_{osc}(\emptyset)$ or not. We define this to be the case for all functions satisfying that $\exists \delta \in [0, T+d)$ so that

$$\text{OUT}_Y(t) = \begin{cases} 1 & \text{if } \exists z \in \mathbb{Z}, \exists \tau \in (0,T) : t = \delta + z(T+d) + \tau \\ 0 & \text{else}\,. \end{cases}$$

Intuitively, δ denotes the fixed time offset of the signal, and τ the time the signal is 1 during the period $t + d$.

If restricted to times $[0, 5]$, Figure 7 shows the execution for $\delta = 0$, $d = 1$ and $T = 1.5d = 1.5$. The channel incorrectly forwards the red signal at its input during time $[4, 5]$ to the red signal at its output during $[5, 6]$, i.e., is not correct during $[5, 6]$. However, the module quickly recovers: Starting from time 6.3, the circuit has returned to a feasible periodic behavior with $\delta = 0.3$ and $\tau = T$. We next show that this stabilizing behavior is guaranteed.

Lemma 1. *If $T \geq d$, the compound module given in Figure 6 is a $(T + 2d)$-stabilizing implementation of* OSC.

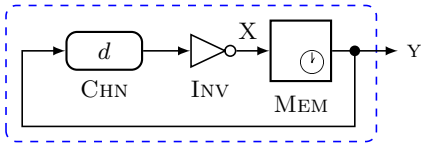

Fig. 6: Self-stabilizing oscillator module.

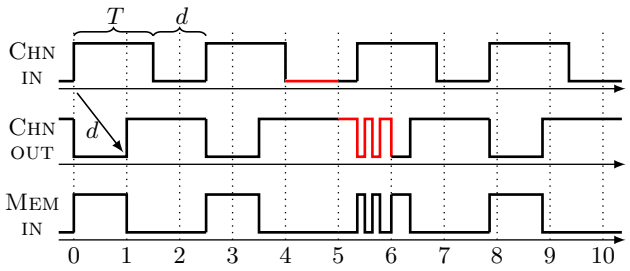

Fig. 7: Execution part of self-stabilizing oscillator module with transient channel fault (incorrect propagation in red).

Proof. Wlog., assume that the compound module follows its specification during $[t^-, t^+) = [0, \infty)$.

1. There is some time $t^* \in [0, T + d]$ when $Y(t^*) = 0$. (Otherwise, the input to MEM at every time $t \in [d, T + d]$ would be $X(t) = \overline{Y}(t - d) = 0$ (with $\overline{Y}$ denoting negation), entailing $Y(T + d) = 0$ by the specification of MEM—a contradiction.)
2. Let $t_0 \in [t^*, t^* + d]$ be maximal with the property that $Y(t) = 0$ for all $t \in [t^*, t_0]$ (t_0 exists because $Y(t^*) = 0$ and $\{t \in \mathbb{R}_0^+ \mid Y(t) = 0\}$ is closed).
3. $X(t) = 0 \vee Y(t) = 1$ for every $t \in (t_0 - T, t_0)$ (specification of MEM).
4. (a) $t_0 < t^* + d$: Then, $Y(t_0^+) = \lim_{\varepsilon \to 0+} Y(t_0 + \varepsilon) = 1$ by maximality of t_0; hence $X(t_0) = 1$ by the specification of MEM and 3.
   (b) $t_0 = t^* + d$: Then, the channel ensures $X(t_0) = \overline{Y}(t^*) = 1$.
5. $Y(t) = 1$ for $t \in (t_0, t_0 + T)$ (specification of MEM).
6. $X(t) = \overline{Y}(t - d) = 0$ for $t \in (t_0 + d, t_0 + T + d) \supseteq (t_0 + T, t_0 + T + d)$ (as $T \geq d$).
7. $Y(t) = 0$ for $t \in [t_0 + T, t_0 + T + d]$ (specification of MEM).

We can now determine Y for larger times inductively, showing for $t_i := t_0 + i(T + d)$, $i \in \mathbb{N}$, that $Y(t) = 1$ for $t \in (t_i + d, t_i + T + d)$ and $Y(t) = 0$ for $t \in [t_i + T, t_i + T + d]$. Hence, the execution is feasible for module OSC during $[t_0, \infty)$. As $t_0 \leq t^* + d \leq T + 2d$, the claim follows.

By contrast, choosing $d > T$ leads to a circuit that does not necessarily self-stabilize.

While the circuit in Figure 6 is a self-stabilizing implementation of OSC, it is not fault-tolerant. A single permanent fault will stop it from operating correctly. However, the principle of using "forgetful" memory to achieve self-stabilization of oscillatory circuits can be carried over to fault-tolerant *distributed* oscillators like DARTS: In [16, 17], we leveraged the approach in the design of fault-tolerant and self-stabilizing solutions to clock generation (and clock distribution [15]). FATAL$^+$, the proposed clock generation scheme, is essentially a distributed oscillator composed of $n \geq 3f + 1$ clock generation nodes, which self-stabilizes in time $\mathcal{O}(n)$ with probability $1 - 2^{-\Omega(n)}$ even in the presence of up to f Byzantine faults [17].

7 Module Composition Artefacts: The Weird Module

In this section, we will show that module composition in our framework sometimes leads to surprising effects. As a consequence, one has to be careful when composing innocently looking modules that hide their true complexity behind deceptively simple specifications.

Like in Section 5, we will restrict signals and feasible executions in module specifications from $\mathbb{R}$ to arbitrary subintervals $I = [t^-, t^+] \subseteq \mathbb{R}$. To make such interval-restrictions explicit, we will sometimes write σ^I, IN^I, OUT^I, E^I etc. Note that such intervals I could also be open (t^-, t^+), closed $[t^-, t^+]$, or half-open, but are always contiguous.

A sequence of signals $(\sigma_i)_{i\in C}$ defined on intervals $I_1 \subseteq I_2 \subseteq \ldots$ with $I_i \subseteq I_{i+1}$ for all $i \in C$, for $C = \{1, \ldots, n\}$ or $C = \mathbb{N}$, is called a *covering* of a signal σ defined on $I = \bigcup_{i\in C} I_i$ if, for all $i \in C$, $\sigma_i = \sigma^{I_i}$. Clearly, any sequence of signals $(\sigma_i)_{i\in C}$ on $I_1 \subseteq I_2 \subseteq \ldots$ with the property that $\sigma_i = \sigma_{i+1}^{I_i}$ for all $i \in C$ defines a unique σ on $I = \bigcup_{i\in C} I_i$ such that $(\sigma_i)_{i\in C}$ is a covering of σ. For $C = \mathbb{N}$, we can hence set $\lim_{i\to\infty} \sigma_i = \sigma$, where σ is defined on $\lim_{i\to\infty} I^i = I$. These definitions and results naturally carry over to interval-restricted executions, i.e., pairs of sets of input and output signals of a module.

Definition 12 (Limit-closure). *Module M is* limit-closed *iff, for every covering $(E_i)_{i\in\mathbb{N}}$ consisting of interval-restricted executions E_i in the set $\mathcal{E}_M$ of all interval-restricted executions of M, it holds that* $\lim_{i\to\infty} E_i \in \mathcal{E}_M$.

Not every module is limit-closed, as the following example demonstrates.

Example 7. Consider the module specification WM, subsequently called the *weird module*: It has no inputs and only a single output, which is required to switch from 0 to 1 within finite time and have no other transitions.

The WM can be seen as an archetypal asynchronous module, as the transition must occur in finite time, but there is no known bound on the time until this happens.

For every $i \in \mathbb{N}$, the execution E_i defined on $[-i, i]$ with the output signal being constant 0 is feasible for WM, as it can be extended to some execution on $\mathbb{R}$ where the transition to 1 occurs, e.g., at time $i+1$; it is thus an extendible module according to Definition 7. However, the limit of $E = \lim_{i\to\infty} E_i$ is the unique execution on $\mathbb{R}$ with output constant 0, which is infeasible for WM. According to Definition 12, the specification of WM is hence extendable but not limit-closed. Conversely, limit-closure does not necessarily imply extendibility either, as the latter requires that *every* execution defined on some interval I can be extended to an execution on $\mathbb{R}$; limit-closure guarantees this only for interval-restricted executions that are part of coverings.

Definition 13 (Finite-delay & bounded-delay module). *Module M has* finite delay *(FD), iff every infeasible execution $E_M \notin \mathcal{E}_M$ has a finite infeasible restriction, i.e.,* $(E_M \notin \mathcal{E}_M) \Rightarrow \big(\exists$ *finite* $I \subset \mathbb{R} : E_M^I \notin \mathcal{E}_M\big)$. *An FD module M*

is a bounded-delay module *(BD), if I may not depend on the particular E_M in the FD definition. Finally, M is a* bounded delay module with delay bound $B \in \mathbb{R}_0^+$*, if it is BD and* $|I| \leq B$.

Recall that if a module is correct in an execution during an interval I it is always also correct within a subinterval of I in the same execution. On the other hand, the other implication $(\exists \text{ finite } I \subset \mathbb{R} : E_M^I \notin \mathcal{E}_M) \Rightarrow (E_M \notin \mathcal{E}_M)$ always holds, by our definition of a restriction. Thus, for FD modules, it holds that $(E_M \notin \mathcal{E}_M) \Leftrightarrow \big(\exists \text{ finite } I \subset \mathbb{R} : E_M^I \notin \mathcal{E}_M\big)$.

According to Definition 13, WM is not a finite-delay module, as any finite restriction of the infeasible all-zero trace on $\mathbb{R}$ is feasible. More generally, we have the following lemma:

Lemma 2. *A module is limit-closed iff it has finite delay.*

Proof. Suppose M is limit-closed. Given an arbitrary $E \notin \mathcal{E}_M$, defined on $I \subseteq \mathbb{R}$, consider the covering $E_i = E^{I \cap [-i,i]}$, $i \in \mathbb{N}$. Then, either there is some i so that $E_i \notin \mathcal{E}_M$, or we reach the contradiction that $E = \lim_{i\to\infty} E_i \in \mathcal{E}_M$ as M is limit-closed.

Conversely, suppose that M is a finite delay module. Consider an arbitrary infinite covering $\{E_i \,|\, E_i \in \mathcal{E}_M\}_{i\in\mathbb{N}}$, and denote by E its limit. If $E \notin \mathcal{E}_M$, then by Definition 13 there is a finite $E^I \notin \mathcal{E}_M$ that is a restriction of E. As $\{E_i\}_{i\in\mathbb{N}}$ is a covering, for sufficiently large i, it holds that the interval I on which E^I is defined is contained in the interval on which E_i is defined. Hence, $E^I \notin \mathcal{E}_M$ is a restriction of $E_i \in \mathcal{E}_M$, which is a contradiction to the fact that E^I must be feasible.

At that point, the question arises whether and when the composition of modules preserves bounded resp. finite delays. The following Corollary 2 shows that this is the case for feedback-free compositions of BD modules:

Corollary 2 (Preservation of BD). *Suppose compound module M is feedback-free with circuit graph G_M and each of its submodules $S \in \mathcal{S}_M$ is FD. Then, M is FD. Moreover, if $S \in \mathcal{S}_M$ is BD with delay bound B_S, then $\mathcal{S}_M$ is BD with delay bound*

$$B = \max_{\substack{(S_1,\dots,S_k)\\ path\ in\ G_M}} \left\{ \sum_{i=1}^{k} B_S \right\}.$$

BD is not preserved in arbitrary compound modules. Unfortunately, the above corollary does not hold if feedback-loops are allowed. A compound module made up of BD submodules in a feedback-loop need not be BD, and sometimes not even FD.

As an example of the former, consider the *eventual short-pulse filter* (eSPF) introduced in [26], which has a single input and a single output port, initially 0. Given a single pulse of duration $\Delta > 0$ at time 0 at the input, there is a time $T = T(\Delta)$ with $\lim_{\Delta\to 0} T = \infty$ such that the output $o(t) = 1$ for all $t \geq T$. Yet, the execution where the output never settles to 1 is not feasible (unless there is

no input pulse). eSPF can be implemented as a compound module consisting of a two-input zero-time OR gate and a two pure-delay channels (with delay 1 and $\sqrt{2}$, respectively) in a feedback-loop. By adding an inertial delay channel [24] to the output of eSPF, which suppresses all pulses with duration less than 1 (and is hence also a BD module[8]), we obtain a module eSPF$'$ that generates exactly one transition from 0 to 1 at the output. Module $eSPF'$ is FD, as a finite interval $I = [0, T]$ that guarantees $E^I \in \mathcal{E}_{\text{eSPF}'}$ can be computed from the known Δ in every given execution E. However, eSPF$'$ is not BD, albeit all its submodules are BD. Consequently, for compound modules that are not feedback-free, Corollary 2 need not hold.

Unexpected properties of the WM module. While the results on BD align with our intuition, similar properties do not hold for FD modules. To show this, let us add another BD basic submodule that acts as a random generator (which is of course also BD) for generating an input pulse of duration $\Delta > 0$ to eSPF (now considered a basic module). We obtain a feedback-free compound module implementation of WM, which is not even FD! Consequently, and surprisingly, one cannot generalize Corollary 2 to the preservation of FD: Feedback-free compound modules composed from FD submodules are *not* always FD: eSPF$'$ is FD, and the random generator is even BD, yet the resulting WM is not FD.

The problem can be traced back to the fact that compound modules hide internal ports (the input port of eSPF fed by the random generator in our WM compound module), which does no longer allow to identify appropriate infeasible finite executions in infinite executions according to Definition 13. Our modeling framework allows to completely abstract away this important submodule-internal information, which in turn creates this artefact. This intuitively suggests that one should not entirely discard the internal structure of a compound module, but rather simulate a "glass box view" of a submodule as advocated in [9] by exposing important submodule-signals when composing modules. A formal understanding of these problems is open, however.

8 Outlook

We discussed an alternative to the classical state-based modeling and analysis approach. In particular, we reviewed the state-oblivious modeling and analysis framework introduced in [16], and argued its utility by means of some examples, in particular, a self-stabilizing oscillator. We also showed that it may create some subtle artefacts when composing modules, which need careful consideration and possibly mitigation.

While we believe that the modeling framework discussed in this article is a sound basis for the formal study of digital circuits and even biological systems, it currently lacks several important features that are left open for further research:

The first one is the choice of a formal language for describing signals and module specifications. Whereas our simple signals could of course be described

[8] Since we can consider the inertial delay channel to be a basic module here, we need not care about implementability at that point.

within a first-order theory on $\mathbb{R}$, it is not clear whether this is the most appropriate formalism for concisely expressing the most relevant properties of interest. Moreover, module specifications often require (all-)quantification over signals, which suggests the need for a second-order theory.

A somewhat related open problem is the definition of a proper notion of simulation equivalence for modules with *different* interfaces, and simulation-type proof techniques similar to the ones known for both untimed [37] and timed [38] distributed systems. Unfortunately, the state-obliviousness and the unconventional domain $\mathbb{R}$ of our framework does not allow one to just take over state-based simulation techniques.

Another open issue is the explicit handling of metastability, which can currently only be expressed by mapping a metastable state to a (high-frequency) pulse train. An obvious alternative is to use a three-valued logic, also providing a dedicated metastable state M, as advocated in [22]. While this extension appears relatively straightforward at the specification level, it should also be accompanied by ways of specifying metastable upsets and metastability propagation.

References

1. Albert, R., Othmer, H.G.: The topology of the regulatory interactions predicts the expression pattern of the segment polarity genes in drosophila melanogaster. J. Theor. Biol. **223**(1), 1–18 (2003)
2. de Alfaro, L., Henzinger, T.A., Stoelinga, M.: Timed interfaces. In: Proc. EMSOFT. pp. 108–122 (2002)
3. Alur, R., Henzinger, T., Lafferriere, G., Pappas, G.: Discrete abstractions of hybrid systems. Proceedings of the IEEE **88**(7), 971–984 (July 2000). https://doi.org/10.1109/5.871304
4. Alur, R., Dill, D.L.: A theory of timed automata. Theoretical Computer Science **126**(2), 183 – 235 (1994). https://doi.org/http://dx.doi.org/10.1016/0304-3975(94)90010-8, http://www.sciencedirect.com/science/article/pii/0304397594900108
5. Awerbuch, B.: Complexity of network synchronization. JACM **32**(4), 804–823 (1985)
6. Bartocci, E., Bortolussi, L., Nenzi, L.: A Temporal Logic Approach to Modular Design of Synthetic Biological Circuits. In: Proc. CMSB. pp. 164–177 (2013)
7. Baumann, R.: Radiation-Induced Soft Errors in Advanced Semiconductor Technologies. IEEE Transactions on Device and Materials Reliability **5**(3), 305–316 (2005)
8. Bellido-Diaz, M.J., Juan-Chico, J., Acosta, A., Valencia, M., Huertas, J.L.: Logical modelling of delay degradation effect in static cmos gates. IEE Proceedings - Circuits, Devices and Systems **147**(2), 107–117 (2000)
9. Broy, M., Stølen, K.: Specification and Development of Interactive Systems: Focus on Streams, Interfaces, and Refinement. Springer-Verlag New York, Inc. (2001)
10. Constantinescu, C.: Trends and Challenges in VLSI Circuit Reliability. IEEE Micro **23**(4), 14–19 (2003)
11. Daliot, A., Dolev, D., Parnas, H.: Self-Stabilizing Pulse Synchronization Inspired by Biological Pacemaker Networks. In: Proc. SSS. pp. 32–48 (2003)

12. Dijkstra, E.W.: Self-Stabilizing Systems in Spite of Distributed Control. CACM **17**(11), 643–644 (1974)
13. Dixit, A., Wood, A.: The Impact of New Technology on Soft Error Rates. In: Proc. IRPS. pp. 5B.4.1–5B.4.7 (2011)
14. Dolev, D., Dwork, C., Stockmeyer, L.: On the Minimal Synchronism Needed for Distributed Consensus. JACM **34**(1), 77–97 (1987)
15. Dolev, D., Függer, M., Lenzen, C., Perner, M., Schmid, U.: HEX: Scaling Honeycombs is Easier than Scaling Clock Trees. In: Proc. 25th ACM Symp. on Parallelism in Algorithms and Architectures (SPAA'13). pp. 164–175 (2013)
16. Dolev, D., Függer, M., Lenzen, C., Posch, M., Schmid, U., Steininger, A.: Rigorously Modeling Self-Stabilizing Fault-Tolerant Circuits: An Ultra-Robust Clocking Scheme for Systems-on-Chip. JCSS **80**(4), 860–900 (2014)
17. Dolev, D., Függer, M., Lenzen, C., Schmid, U.: **F**ault-tolerant **A**lgorithms for **T**ick-generation in **A**synchronous **L**ogic: Robust Pulse Generation. J. ACM **61**(5), 30:1–30:74 (Sep 2014). https://doi.org/10.1145/2560561
18. Dolev, S.: Self-Stabilization. MIT Press (2000)
19. Dwork, C., Lynch, N., Stockmeyer, L.: Consensus in the Presence of Partial Synchrony. JACM **35**(2), 288–323 (1988)
20. Fischer, M.J.: The Consensus Problem in Unreliable Distributed Systems (a Brief Survey). In: Proc. FCT. pp. 127–140 (1983)
21. Fischer, M., Lynch, N., Paterson, M.: Impossibility of distributed consensus with one faulty process. JACM **32**(2), 374–382 (1985)
22. Friedrichs, S., Függer, M., Lenzen, C.: Metastability-containing circuits. IEEE Trans. Computers **67**(8), 1167–1183 (2018). https://doi.org/10.1109/TC.2018.2808185, https://doi.org/10.1109/TC.2018.2808185
23. Fuchs, G., Steininger, A.: VLSI Implementation of a Distributed Algorithm for Fault-Tolerant Clock Generation. J. Electr. Comput. Eng. **2011**(936712) (2011)
24. Függer, M., Najvirt, R., Nowak, T., Schmid, U.: A faithful binary circuit model. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems **39**(10), 2784–2797 (October 2020). https://doi.org/10.1109/TCAD.2019.2937748
25. Függer, M., Kushwaha, M., Nowak, T.: Digital circuit design for biological and silicon computers. Advances in Synthetic Biology pp. 153–171 (2020)
26. Függer, M., Nowak, T., Schmid, U.: Unfaithful glitch propagation in existing binary circuit models. IEEE Transactions on Computers **65**(3), 964–978 (March 2016). https://doi.org/10.1109/TC.2015.2435791, http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=7110587
27. Függer, M., Schmid, U.: Reconciling Fault-Tolerant Distributed Computing and Systems-on-Chip. Distributed Computing **24**(6), 323–355 (2012)
28. Gardner, T.S., Cantor, C.R., Collins, J.J.: Construction of a genetic toggle switch in escherichia coli. Nature **403**, 339–342 (2000)
29. Gorochowski, T.E., Espah Borujeni, A., Park, Y., Nielsen, A.A., Zhang, J., Der, B.S., Gordon, D.B., Voigt, C.A.: Genetic circuit characterization and debugging using rna-seq. Molecular systems biology **13**(11), 952 (2017)
30. Hasty, J., McMillen, D., Collins, J.J.: Engineered Gene Circuits. Nature **420**, 224–230 (2002)
31. International Technology Roadmap for Semiconductors (2012), http://www.itrs.net
32. Kaynar, D.K., Lynch, N., Segala, R., Vaandrager, F.: The Theory of Timed I/O Automata. Morgan & Claypool Publishers (2006)

33. Koren, I., Koren, Z.: Defect tolerance in VLSI circuits: Techniques and yield analysis. Proceedings of the IEEE **86**(9), 1819–1838 (Sep 1998). https://doi.org/10.1109/5.705525
34. Lamport, L.: Time, Clocks, and the Ordering of Events in a Distributed System. CACM **21**(7), 558–565 (1978)
35. Lamport, L.: The temporal logic of actions. ACM Trans. Program. Lang. Syst. **16**(3), 872–923 (1994)
36. Lee, E.A., Varaiya, P.: Structure and Interpretation of Signals and Systems. LeeVaraiya.org, 2nd edn. (2011)
37. Lynch, N., Vaandrager, F.: Forward and backward simulations, I: Untimed systems. Information and Computation **121**(2), 214–233 (Sep 1995)
38. Lynch, N., Vaandrager, F.: Forward and backward simulations, II: Timing-based systems. Information and Computation **128**(1), 1–25 (Jul 1996)
39. Marino, L.: General Theory of Metastable Operation. IEEE Transactions on Computers **C-30**(2), 107–115 (1981)
40. Maza, M.S., Aranda, M.L.: Analysis of Clock Distribution Networks in the Presence of Crosstalk and Groundbounce. In: Proc. ICECS. pp. 773–776 (2001)
41. Nielsen, A.A., Der, B.S., Shin, J., Vaidyanathan, P., Paralanov, V., Strychalski, E.A., Ross, D., Densmore, D., Voigt, C.A.: Genetic circuit design automation. Science **352**(6281) (2016)
42. Pease, M., Shostak, R., Lamport, L.: Reaching Agreement in the Presence of Faults. JACM **27**, 228–234 (1980)
43. Peercy, M., Banerjee, P.: Fault Tolerant VLSI Systems. Proceedings of the IEEE **81**(5), 745–758 (1993)
44. Santos-Moreno, J., Tasiudi, E., Stelling, J., Schaerli, Y.: Multistable and dynamic crispri-based synthetic circuits. Nature communications **11**(1), 1–8 (2020)
45. Sivan, E., Parnas, H., Dolev, D.: Fault Tolerance in the Cardiac Ganglion of the Lobster. Biological Cybernetics **81**(1), 11–23 (1999)
46. Stricker, J., Cookson, S., Bennett, M.R., Mather, W.H., Tsimring, L.S., Hasty, J.: A Fast, Robust and Tunable Synthetic Gene Oscillator. Nature **456**, 516–519 (2008)
47. Sutherland, I.E.: Micropipelines. CACM **32**(6), 720–738 (1989)
48. Thomas, R.: Boolean formalization of genetic control circuits. J. Theor. Biol. **42**(3), 563–585 (1973)
49. Unger, S.H.: Asynchronous sequential switching circuits with unrestricted input changes. IEEE Trans. Computers **20**(12), 1437–1444 (1971)