Emergence at the Fundamental Systems Level: Existence Conditions for Iterative Specifications

Bernard P Zeigler, Alexandre Muzy

systems

MDPI

Article

Emergence at the Fundamental Systems Level: Existence Conditions for Iterative Specifications

Bernard P. Zeigler [1,*] and Alexandre Muzy [2]

[1] Co-Director of the Arizona Center for Integrative Modeling and Simulation (ACIMS), University of Arizona and Chief Scientist, RTSync Corp. 12500 Park Potomac Ave. #905-S, Potomac, MD 20854, USA
[2] CNRS, I3S, Université Côte d'Azur, 06900 Sophia Antipolis, France; alexandre.muzy@cnrs.fr
* Correspondence: zeigler@rtsync.com

Academic Editors: Gianfranco Minati, Eliano Pessa and Ignazio Licata
Received: 17 August 2016; Accepted: 25 October 2016; Published: 9 November 2016

Abstract: Conditions under which compositions of component systems form a well-defined system-of-systems are here formulated at a fundamental level. Statement of what defines a well-defined composition and sufficient conditions guaranteeing such a result offers insight into exemplars that can be found in special cases such as differential equation and discrete event systems. For any given global state of a composition, two requirements can be stated informally as: (1) the system can leave this state, i.e., there is at least one trajectory defined that starts from the state; and (2) the trajectory evolves over time without getting stuck at a point in time. Considered for every global state, these conditions determine whether the resultant is a well-defined system and, if so, whether it is non-deterministic or deterministic. We formulate these questions within the framework of iterative specifications for mathematical system models that are shown to be behaviorally equivalent to the Discrete Event System Specification (DEVS) formalism. This formalization supports definitions and proofs of the afore-mentioned conditions. Implications are drawn at the fundamental level of existence where the emergence of a system from an assemblage of components can be characterized. We focus on systems with feedback coupling where existence and uniqueness of solutions is problematic.

Keywords: emergence; uniqueness; existence of solutions; input/output system; system specifications; Discrete Event System Specification

1. Introduction

Emergence has been characterized as taking place in strong and week forms. Mittal [1] pointed out that strong emergent behavior results in generation of new knowledge about the system representing previously unperceived complex interactions. This can occur in the form of one or more of new abstraction levels and linguistic descriptions, new hierarchical structures and couplings, new component behaviors, and new feedback loops. Once understood and curated, the behavior returns to the weak form, as it is no longer intriguing, and then can begin to be treated in regularized fashion. Emergent behavior is likely an inherent feature of any complex system model because abstracting a continuous real-world system (e.g., any complex natural system) to a constructed system-model must leave gaps of representation that may diverge in unanticipated directions. In [2] philosophically, following Ashby [3] and Foo and Zeigler [4], the perceived global behavior (holism) of a model might be characterized as: Components (reductionism) + interactions (computation) + higher-order effects where the latter can be considered as the source of emergent behaviors [5,6].

The Discrete Event Systems Specification (DEVS) formalism has been advocated as an advantageous vehicle for researching such structure–behavior relationships because it provides the components and couplings for models of complex systems and supports dynamic structure for genuine

adaption and evolution. Furthermore, DEVS enables fundamental emergence modeling because it operationalizes the closure-under-coupling conditions that form the basis of well-defined resultants of system composition especially where feedback coupling prevails [7,8].

In this paper, we formulate conditions under which a composition of component systems form a well-defined system-of-systems at a fundamental level. Formal statement of what defines a well-defined composition and sufficient conditions guaranteeing such a result offer insight into exemplars that can be found in special cases such as differential equation and discrete event systems. Informally stated, we show that for any given global state of a composition, two requirements can be stated as: (1) the system can leave this state, i.e., there is at least one trajectory defined that starts from the state; and (2) the trajectory evolves over time without getting stuck at a point in time. Considered for every global state, these conditions determine whether the resultant is a well-defined system and if so, whether it is non-deterministic or deterministic. We formulate these questions within the framework of iterative specifications for mathematical system models that are shown to be behaviorally equivalent to the Discrete Event System Specification (DEVS) formalism. This formalization supports definitions and proofs of the afore-mentioned conditions and allows us to exhibit examples and counter-examples of condition satisfaction. Drawing on Turing machine halting decidability, we investigate the probability of legitimacy for randomly constructed DEVS models. We close with implications for further research on the emergence of new classes of well-defined systems.

Let us start with a well-known concept, the Turing Machine (TM) (cf. [9]). Usually, it is presented in a holistic, unitary manner but as shown in Figure 1b, we can decompose it into two stand-alone independent systems: the TM Control (*S*1) and the Tape System, (*S*2). Foo and Zeigler [4] argued that the re-composition of the two parts was an easily understood example of emergence wherein each standalone system has very limited power but their composition has universal computation capabilities. Examining this in more depth, the Tape system shown in Figure 1a is the dumber of the two, serving a memory with a slave mentality, it gets a symbol (sym) and a move (mv) instruction as input, writes the symbol to the tape square under the head, moves the head according to the instruction, and outputs the symbol found at the new head location. The power of the Tape system derives from its physicality—its ability to store and retrieve a potentially infinite amount of data—but this can only be exploited by a device that can properly interface with it. The TM Control by contrast has only finite memory but its capacity to make decisions (i.e., use its transition table to jump to a new state and produce state-dependent output) makes it the smarter executive. The composition of the two exhibits "weak emergence" in that the resultant system behavior is of a higher order of complexity than those of the components (logically undecidable versus finitely decidable), the behavior that results can be shown explicitly to be a direct consequence of the component behaviors and their essential feedback coupling—cross-connecting their outputs to inputs as shown by the arrows of Figure 1b. We are going to use this example to discuss the general issues in dealing with such compositions.

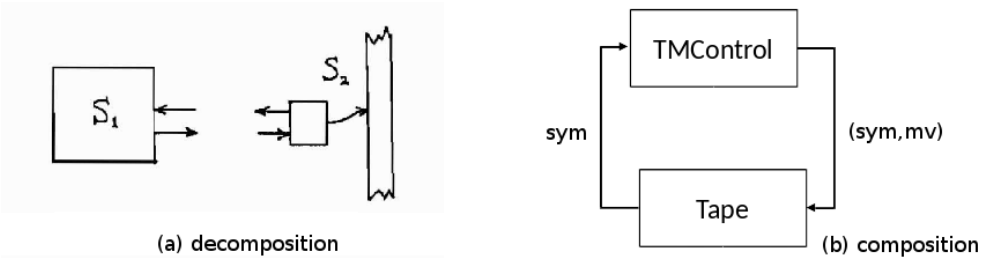

Figure 1. Turing Machine decomposition (**a**) and modular composition (**b**).

The interdependent form of the interaction between the TM control and the tape system illustrates a pattern found in numerous information technology and process control systems and recognized early on by simulation language developers [10] (though not necessarily in the modular form described here). In this interaction, each component system alternates between two phases, active and passive.

When one system is active the other is passive—only one can be active at any time. The active system does two actions: (1) it sends an input to the passive system that activates it (puts it into the active phase); and (2) it transits to the passive phase to await subsequent re-activation. For example, in the re-composed Turing Machine, the TM control starts a cycle of interaction by sending a symbol and move instruction to the tape system then waiting passively for a new scanned symbol to arrive. The tape system waits passively for the sym, mv pair. When it arrives, it executes the instruction and sends the symbol now under the head to the waiting control.

Such active–passive compositions provide a class of systems from which we can draw intuition and examples for generalizations about system emergence at the fundamental level. We will employ the modeling and simulation framework based on system theory formulated in [10] especially focusing on its concepts of iterative specification and the Discrete Event Systems Specification (DEVS) formalism. Special cases of memory-less systems and the pattern of active–passive compositions are discussed to exemplify the conditions resulting ill-definition, deterministic, and non-deterministic as well as probabilistic systems. We provide sufficient conditions, meaningful especially for feedback coupled assemblages, under which iterative system specifications can be composed to create a well-defined resultant and that moreover can be simulated in the DEVS formalism.

However, to address more fundamental issues, we need to start with a more primitive and perhaps more intuitive notion of a system. As in Figure 2, consider a concept of system with states, transitions, and times associated with transitions. For example, there are transitions from state *S1* to state *S3* and from *S3* to *S4* which each takes 1 time unit and there is a cycle of transitions involving $S4, \ldots, S7$ each of which take zero time. There is a self-transition involving *S2* which consumes an infinite amount of time (signifying that it is passive, remaining in that state forever.) This is distinguished from the absence of any transitions out of *S8*. A state trajectory is a sequence of states following along existing transitions, e.g., *S1*, *S3*, *S4* is such a trajectory.

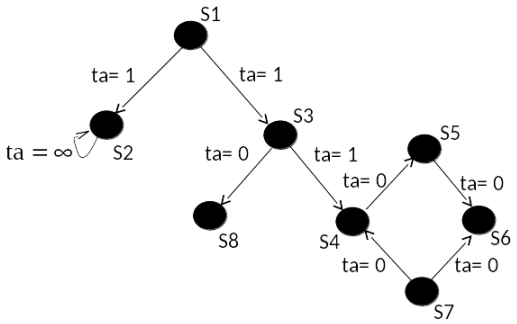

Figure 2. System with tuned transitions.

This example gives us a quick understanding of the conditions for system existence at the fundamental level.

We say that the system is:

- *not defined* at *S8* because there is no trajectory emerging from it;
- *non-deterministic* at *S1* because there are two distinct outbound transitions defined for it; and
- *deterministic* at *S2* and *S4* because there is only one outbound transition for each.

We say that the system is *well-defined* if it is defined at all its states. These conditions are relative to static properties, i.e., they relate to states not how the states follow one another over time. In contrast, state trajectories relate to dynamic and temporal properties. When moving along a trajectory, we keep adding the time advances to get the total traversal time, e.g., the time taken to go from *S2* to *S4* is 2. Here, a trajectory is said to be *progressive* in time if time always advances as we extend the trajectory. For example, the cycle of states $S4 \ldots S7$ is not progressive because as we keep adding the time advances the sum never increases. Conceptually, let us start a clock at 0 and, starting from a given

state, we let the system evolve following existing transitions and advancing the clock according to the time advances on the transitions. If we then ask what the state of the system will be at some time later, we will always be able to answer if the system is well-defined and progressive. A well-defined system that is not progressive signifies that the system gets stuck in time and after some time, it becomes impossible to ask what the state of the system is in after that time. Zeno's paradox offers a well-known metaphor where the time advances diminish so that time accumulates to a point rather than continue to progress and offers an example showing that the pathology does not necessarily involve a finite cycle. Our concept of progressiveness generalizes the concept of legitimacy for DEVS [10] and deals with the "zenoness" property which has been much studied in the literature [11]. We return to it in more detail later.

Thus, we have laid the conceptual groundwork in which a system has to be well-defined (static condition) and progressive (temporal dynamic condition) if it is to have achieved independent existence when emerging from a composition of components.

2. Formal Background

Hereafter are presented basic and coupled specifications.

2.1. Basic Discrete Event System Specification

As presented in [12], a *Discrete Event System Specification* (*DEVS*) is a structure

$$M =< X, S, Y, \delta_{int}, \delta_{ext}, \lambda, ta >$$

where X is the *set of input values*, S is a *set of states*, Y is the *set of output values*, $\delta int : S \to S$ is the *internal transition function*, $\delta_{ext} : Q \times X \to S$ is the *external transition function*, where $Q = \{(s, e) | s \in S, 0 \leq e \leq ta(s)\}$ is the *total state* where e is the *time elapsed* since last transition, $\lambda : S \to Y$ is the *output function*, $ta : S \to \mathbb{R}^+_\infty$ is the *time advance function*.

In the Turing machine example of Figure 1, the tape system and control engine are each atomic DEVS. For the tape system, a state is a triple $(tape, pos, mv)$ where the tape is an infinite sequence of zeros and ones (symbols), pos represents the position of the head, and mv is a specification for moving left or right. An external transition accepts a symbol, move pair, writes the symbol in the square of the current head position and stores the move for the subsequent internal transition that executed the specified move. For the control engine, a state is a pair (st, sym) where st is a control state and sym is a stored symbol. An external transition stores the received symbol for subsequent use. An internal transition applies the TM transition table to the (st, sym) pair and transitions to the specified control state.

The core of this concept simplifies when the system is input-free (or autonomous, i.e., not responding to inputs) and outputs are not considered. Then, we have a transition system

$$M =< S, \delta, ta >$$

where $\delta \subseteq S \times S$ and $ta : S \times S \to \mathbb{R}^+_\infty$.

Here, we are allowing the transition system to be non-deterministic so that rather than δ and ta being functions they are presented as relations.

For example, in Figure 2, we have $S = \{S1, S2, S3, S4, S5, S6, S7\}$, $\delta = \{(S1, S3), (S3, S4), ...\}$, and $ta(S1, S3) = 1$, $ta(S3, S4) = 1$,...

In this formal version of the informal statements above, the transition system, M, is

- *not defined* at $S8$ because there is no transition pair with S8 as the left member in δ;
- *non-deterministic* at $S1$ because it is a left member of two transition pairs $(S1, S2)$ and $(S1, S3)$; and
- *deterministic* at $S2$ and $S4$ because there is only one transition pair involving each one as a left member.

This gives concrete and formal expression to the earlier definition of these concepts.

2.2. Coupled DEVS Models

DEVS models can be coupled to form coupled models that themselves are DEVS models manifesting closure under coupling. To keep the presentation as straightforward as possible in this paper we will limit the discussion to coupling of two components without external inputs or outputs. This is illustrated in Figure 1b where the coupling recipe maps the output of the control engine *(symbol, move pair)* to the input of the tape system and likewise, the output of the tape system *(symbol)* to the input of the control engine. This mapping is assumed to take zero time (any delay in a real manifestation can be modeled by inserting delay components). To sketch the way the resultant of coupling is computed and show that it is expressed as a DEVS, let $M =< X, S, Y, \delta_{int}, \delta_{ext}, \lambda, ta >$ and $M' =< X', S', Y', \delta'_{int}, \delta'_{ext}, \lambda', ta' >$.

Then, the resultant is a DEVS:

$$M_{coup} =< Q_{coup}, \delta_{coup}, ta_{coup} >$$

where $Q_{coup} = Q \times Q'$, $ta_{coup} : Q_{coup} \to R^{+}_{\infty}$, $\delta_{coup} : Q_{coup} \to Q_{coup}$ and $ta((s,e),(s',e')) = \min\{ta(s) - e, ta'(s') - e'\}$.

To explain, the next event will occur according the time advance and will be driven by the imminent component (whose next event time is the minimum of the two—neglecting ties for simplicity here). For illustration, let t^* be the time advance and let the second component be imminent. This component generates its output and sends it to the first via the coupling which reacts to it using its external transition function establishing its next state $\delta_{ext}(s, t^*, \lambda'(s'))$; at the same time, the imminent component applies its internal transition function to establish its next state $\delta'_{int}(s')$. Thus, we have $\delta_{coup}((s,e),(s',e')) = (\delta_{ext}(s, t^*, \lambda'(s')), \delta'_{int}(s'))$.

Having briefly reviewed the basics of DEVS, we are ready to consider the general case of iterative specification of which DEVS is an example.

3. Iterative System Specifications

We briefly review an approach to Iterative Specification of Systems that was introduced in [12] and provide more detail in Appendix B. I/O systems describe system behavior with a global transition function that determines the final state given the initial state and the applied input segment. Since input segments are left segmentable and closed under composition, we are able to generate the state and output values along the entire input interval. However, such an approach is not very practical. What we need is a way to generate state and output trajectories in an iterative way going from one state along the trajectory to the next.

Iterative specification of systems is a general scheme for defining systems by iterative applications of *generator segments*. These are elementary segments from which all input segments of a system can be generated. Having such a concept we define a *generator state transition function* and iteratively apply generator segments to the generator state transition function. The results produced by the generator state transitions constitute the state trajectory for the input segment resulting from the composition of the generator segments. The general scheme of iterative specification forms a basis for more specialized types of specifications of systems. System specification formalisms are special forms of iterative specifications with their special type of generator segments and generator state transition functions.

Consider (Z, T) the set of all segments $\{\omega :< 0, t_1 >\to Z \mid t_1 \in T\}$. Here, the notation, $\omega :< 0, t_1 >\to Z\}$ means that ω is a mapping from an interval of time base, T to a set of values Z. For a subset Γ of (Z, T), the *concatenation closure* of Γ is denoted Γ^{+}. Example generators are bounded continuous segments, and constant segments of variable length generating bounded piecewise continuous segments piecewise constant segments, respectively. Unfortunately, if Γ generates Ω, we *cannot* expect each $\omega \in \Omega$ to have a unique decomposition by Γ. A single representative, or *canonical* decomposition can be computed using a *Maximal Length Segmentation* (MLS). First we

find ω_1, the longest generator in Γ that is also a left segment of ω. This process is repeated with what remains of ω after ω_1 is removed, generating ω_2, and so on. If the process stops after n repetitions, then $\omega = \omega_1\omega_2 \ldots \omega_n$. We say that Γ is an *admissible* set of generators for Ω if Γ generates Ω and for each $\omega \in \Omega$, a unique MLS decomposition of ω by Γ exists.

The following is the basis for further analysis in this paper:

Theorem 1. *Sufficient Conditions for Admissibility. If Γ satisfies the following conditions, it admissibly generates Γ^+:*

1. *Existence of longest initial segments: $\omega \in \Gamma^+ \Rightarrow max\{t \mid \omega_{t>} \in \Gamma\}$ exists*
2. *Closure under right segmentation: $\omega \in \Gamma \Rightarrow \omega_{<\tau} \in \Gamma$ for all $\tau \in dom(\omega)$*

An *iterative specification* of a system is a structure

$$G = < T, X, \Omega_G, Y, Q, \delta_G, \lambda >$$

where T, X, Y, and Q have the same interpretation as for I/O systems, Ω_G is an admissible set of input segment generators, $\delta_G : Q \times \Omega_G \to Q$ is the single segment state transition function, and $\lambda : Q \times X \to Y$ is the output function.

An iterative specification $G = < T, X, \Omega_G, Y, Q, \delta_G, \lambda >$ specifies a time invariant system, $S_G = < T, X, \Omega_G^+, Y, Q, \delta_G^+, \lambda >$. The system is well-defined if δ_G^+, the extension of δ_G, has the composition property, i.e., $\delta_G^+(q, \omega_1 \bullet \omega_2) = \delta_G^+(\delta_G^+(q, \omega_1), \omega_2)$, for all $\omega_1, \omega_2 \in \Omega_G^+$.

For more details, see Appendix B.

3.1. DEVS Simulation of Iterative Specification

As reviewed in Appendix B, the notion of iterative specification was introduced to characterize diverse classes of systems such as differential equation systems and discrete time systems. With the motivation of including discrete event systems under the same umbrella as more familiar systems, DEVS was defined using iterative specification. Here we show that the converse is also true, namely, that DEVS can directly represent iterative specifications. Given an iterative specification $G = < T, X, \Omega_G, Y, Q, \delta_G, \lambda >$, we construct a DEVS model $M = < X, S, Y, \delta_{int}, \delta_{ext}, \lambda, ta >$ that can simulate it in a step-by-step manner moving from one input segment to the next. The basic idea is that we build the construction around an encoding of the input segments of G into the event segments of M as illustrated in Figure 3. This is based on the MLS which gives a unique decomposition of input segments to G into generator subsegments. The encoding maps the generator subsegments. ω_i in the order they occur into corresponding events which contain all the information of the segment itself. To do the simulation, the model M stores the state of G. It also uses its external transition function to store its version of G's input generator subsegment when it receives it. M then simulates G by using its internal transition function to apply its version of G's transition function to update its state maintaining correspondence with the state of G.

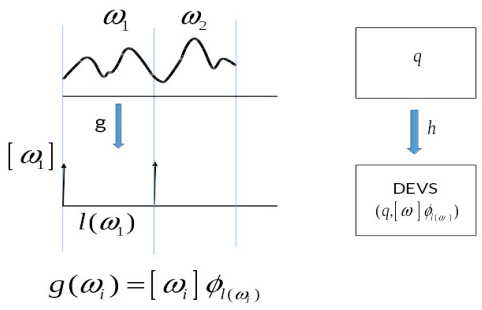

Figure 3. DEVS Simulation of an Iterative Specification.

Details of the construction are provided in Appendix C. We note that the key to the proof is that iterative specifications can be expressed within the explicit event-like constraints of the DEVS formalism, itself defined through an iterative specification. Thus DEVS can be viewed as the computational basis for system classes that can be specified in iterative form satisfying all the requirements for admissibility.

3.2. Coupled Iterative Specification

Although closure of coupling holds for DEVS, the generalization to iterative specification does not immediately follow. The problem is illustrated in Figure 4 where two iterative system specifications, ISP1 and ISP2, are cross-coupled such as exemplified by the Turing machine example in Figure 1. The cross-coupling introduces two constraints shown by the equalities in the figure, namely, the input of ISP2, $\omega 2$ must equal the output of ISP1, $\rho 1$, and the input of ISP1, $\omega 1$ must equal the output of ISP2, $\rho 2$. Here we are referring to input and output trajectories over time as suggested graphically in Figure 4. Since each system imposes its own constraints on its input/output relation, the conjunction of the four constraints (two coupling-imposed, two system-imposed) may have zero, one, or multiple solutions.

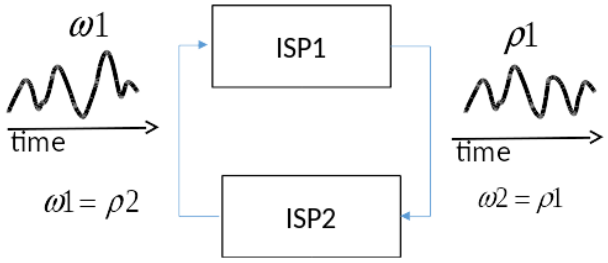

Figure 4. Input and output trajectories of coupled systems.

Definition 1. *A Coupled Iterative Specification is a network specification of components and coupling where the components are iterative specifications at the I/O System level.*

As indicated, in the sequel, we deal with the simplified case of two coupled components. However, the results can be readily generalized with use of more complex notation. We begin with a definition of the relation of input and output segments that a system imposes on its interface. We need this definition to describe the interaction of systems brought on through the coupling of outputs to inputs.

Definition 2. *The I/O Relation of an iterative specification is inherited from the I/O Relation of the system that it specifies. Likewise, the set of I/O Functions of an iterative specification is inherited from the system it specifies. Formally, let* $S_G = < T, X, \Omega_G^+, Y, Q, \delta_G^+, \lambda >$ *be the system specified by* $G = < T, X, \Omega_G, Y, Q, \delta_G, \lambda >$. *Then the I/O Functions associated with G is* $\beta : Q \times \Omega_G^+ \rightarrow \Omega_G^+$ *where for* $q \in Q, \omega \in \Omega_G^+$, $\beta(q, \omega) = \lambda(\delta_G^+(q, \omega))$.

Let $\beta 1$ and $\beta 2$ represent the I/O functions of the iterative specifications, ISP1 and ISP2, respectively. Applying the coupling constraints expressed in the equalities above, we make the definition:

Definition 3. *A pair of output trajectories (ρ1,ρ2) is a consistent output trajectory for the state pair (q1,q2) if* $\rho 1 = \beta 1(q1,\rho 2)$ *and* $\rho 2 = \beta 2(q2,\rho 1)$.

Definition 4. *A Coupled Iterative Specification has unique solutions if there is a function,* $F{:}Q1 \times Q2 \rightarrow \Omega$; *with* $F(q1,q2) = (\rho 1,\rho 2)$, *where there is exactly one consistent pair (ρ1,ρ2) with infinite domain for every initial state (q1,q2). Infinite domain is needed for convenience in applying segmentation.*

Definition 5. *A Coupled Iterative Specification is* admissible *if it has unique solutions.*

The following theorems are proved in Appendix D.

Theorem 2. *An admissible Coupled Iterative Specification specifies a* well-defined *Iterative Specification at the I/O System level.*

Theorem 3. *The set of Iterative Specifications is closed under admissible coupling.*

Theorem 4. *DEVS coupled model can component-wise simulate a coupled Iterative Specification.*

Proof. The coupled model has components, which are DEVS representations of the individual Iterative Specifications (according to Theorem D in Appendix D) and also a coordinator as shown in Figure 5.

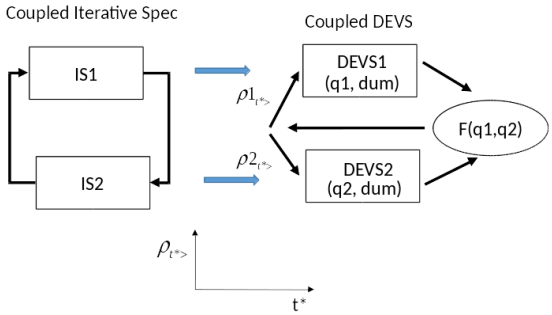

Figure 5. DEVS Component-wise simulation of Coupled Iterative Specifications.

The coordinator receives the current states of the components and applies the F function to compute unique consistent output segments. After segmentation using the MLS as in the proof of Theorem D in Appendix D, it packages each as a single event in a DEVS segment as shown. Each DEVS component computes the state of its Iterative Specification at the end of the segment as in Theorem D. Then it sends this state to the coordinator and the cycle repeats. This completes an informal version of the proof which would formally proceed by induction. □

The solution function F represents an idealization of the fixed point solutions required for Differential Equation System Specification (DESS) and the local solution approaches of Discrete Time System Specification (DTSS) and Quantized DEVS [12]. Generalized Discrete Event System Specification (GDEVS) [13] polynomial representation of trajectories is the closest realization but the approach opens the door to realization by other trajectory prediction methods.

3.3. Special Case: Memoryless Systems

Consider the case where each component's output does not depend on its state but only on its input.

Let gr represent the ground state in which the device is always found (all states are represented by this state since output does not depend on state.) In the following $R1$ and $R2$ are the I/O relations of systems 1 and 2, respectively. In this case, they take special forms:

$$\begin{aligned}
&\rho 1 = \beta 1(gr, \omega 1) \Leftrightarrow (\omega 1, \rho 1) \in R1\\
&\rho 2 = \beta 2(gr, \omega 2) \Leftrightarrow (\omega 2, \rho 2) \in R2\\
&\omega 2 = \rho 1\\
&\omega 1 = \rho 2\\
&\rho 2 = \beta 2(q2, \omega 2) \Leftrightarrow (\rho 1, \omega 1) \in R2 \Leftrightarrow (\omega 1, \rho 1) \in R2^{-1}
\end{aligned}$$

i.e., $(\rho 2, \rho 1)$ is consistent $\Leftrightarrow (\rho 2, \rho 1) \in R1 \cap R2^{-1}$

Let f and g be defined in the following way:

$$f(\rho 2) = \beta 1(gr, \rho 2)$$
$$g(\rho 1) = \beta 2(gr, \rho 1)$$

So for any $\rho 1$, $\rho 1 = f(\rho 2) = f(g(\rho 1))$ and $g = f^{-1}$ (considered as a relation).
Finally, $F(q1, q2) = (\rho 1, f^{-1}(\rho 1))$ has:

- no solutions if f^{-1} does not exist, yielding no resultant;
- a unique solution for every input if f^{-1} exists, yielding a deterministic resultant; and
- multiple solutions for a given segment ρ if $f^{-1}(\rho)$ is multivalued, yielding a non-deterministic resultant.

For examples, consider an *adder* that always adds 1 to its input, i.e.,

$$\beta(gr, \rho) = f(\rho) = \rho + 1$$
$$i.e., \forall t \in T, \beta(gr, \rho)(t) = \rho(t) + 1$$

Cross-coupling a pair of adders does not yield a well-defined resultant because $f^{-1}(\rho) = \rho - 1 \neq f$. However, coupling an adder to a *subtracter*, its inverse, $\beta(gr, \rho) = \rho - 1$, yields a well-defined deterministic system.

For other examples, consider combinatorial elements whose output is a logic function of the input and consider a pair of gates of the same type connected in a feedback loop:

- A NOT gate has an inverse $0 \rightarrow 1, 1 \rightarrow 0$, so the composition has two solutions one for each of two complementary assignments, i.e., $F(gr, gr) = \{(1, 0), (0, 1)\}$ yielding a non-deterministic system.
- An AND gate with one of its input held to 0 always maps the other input into 0 so has a solution only for inputs of 0 to each component, i.e., $F(gr, gr) = (0, 0)$, yielding a deterministic system.
- An AND gate with a stuck-at input of 1 is the identity mapping and has multiple solutions i.e., $F(gr, gr) = \{(0, 0), (1, 1)\}$, yielding a non-deterministic system.

3.4. Active-Passive Systems

We now show that active–passive systems as described earlier offer a class of systems for which the iterative specifications of components satisfy the admissibility conditions specified in Theorem 4. As in Figure 6a, consider a pair of cross-coupled systems each having input generators that represent null and non-null segments.

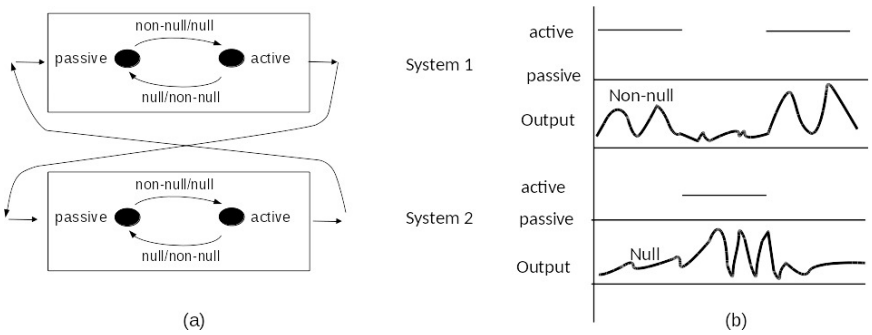

Figure 6. Active–Passive Example of Admissible Progressive Coupled Iterative Specifications.

A null generator represents the output of a passive system whereas a non-null generator represents the output of an active system. For example in the TM case, a non-null generator is a segment that represents transmission of a symbol by the tape unit and of a symbol, move pair for the TM control. As in Figure 6b let *S1* and *S2* start as active and passive, respectively. Then, they output non-null and null generators, respectively. Since the null generator has infinite extent, the end-time of the

non-null generator determines the time of next event, t^* (as defined earlier) and we apply the transition functions in Figure 6a to find that at t^* the systems have reversed phases with *S1* in passive and *S2* in active.

Definition 6. *Define a triple* $< Q, \delta_G, \lambda >$ *by* $Q = \{active, passive\} \times \{active, passive\}$ *with* $F(active, passive) = (non\text{-}null, null)$ $\delta_G(active, passive) = (\delta(active, null_{t^*>}) = \delta(passive, non\text{-}null_{t*>}))$ $= (passive, active)$ *with* $t^*(active, passive)$= *endtime of the non-null generator segment* $\lambda(active, passive) = (non\text{-}null, null)$

Let the non-null and null generators stand for sets of concrete segments that are equivalent with respect to state transitions and outputs. Then, such a scheme can define a deterministic or non-deterministic resultant depending on the number of consistent pairs of *(non-null, null)* generator segments are possible from the *(active, passive*) state. Furthermore, mutatis-mutandis for the state *(passive, active)*.

4. Temporal Progress: Legitimacy, Zenoness

Although we have shown how a DEVS coupled model can simulate a coupling of iterative specifications, it does not guarantee that such a model is legitimate [12]. Indeed, backing up, legitimacy of the simulating DEVS is reflective of the temporal progress character of the original coupling of specifications. We see that the time advance has a value provided that the function F is defined for current state. However, as in the case of zenoness, a sequence of such values may accumulate at a finite point thus not allowing the system to progress beyond this point. Consequently, we need to extend the requirements for coupled iterative specifications to include temporal progress in order to fully characterize the resultant composition of systems. Thus, we can extend the definitions:

Definition 7. *An Iterative Specification is* progressive *if every sequence of time advances diverges in sum.*

Definition 8. *A Coupled Iterative Specification is* progressively admissible *if its components are each progressive and the resultant Iterative Specification at the I/O System level is progressive.*

Definition 9. *A progressive Iterative Specification can be simulated by a* legitimate *DEVS.*

Definition 10. *A* progressively admissible *Coupled Iterative Specification can be component-wise simulated by a legitimate DEVS.*

Zeigler et al. [12] showed that the question of whether a DEVS is legitimate is undecidable. The approach was to express a Turing Machine as a DEVS and showing that solving the halting problem is equivalent to establishing legitimacy for this DEVS. We can transfer this approach to the formulation of the TM in Figure 1 by setting the time advances of both components to zero except for a halt state which is a passive state (and has an infinite time advance). Then a TM DEVS is legitimate just in case the TM it implements ever halts. Since there is no algorithm to decide whether an arbitrary TM will halt or not the same is true for legitimacy. Since DEVS is an iterative specification, the problem of determining whether an Iterative Specification is progressive is also undecidable. Likewise, the question of whether a coupled iterative specification is progressive is undecidable.

Fundamental Systems Existence: Probabilistic Characterization of Halting

The existence of an iteratively specified system requires temporal progress and there is no algorithm to guarantee in a finite time that such progress is true. The underlying point is that, given an arbitrary TM, we have to simulate it step-by-step to determine if it will halt. Importantly, if a TM has not halted after some time, however long, this gives us no further information on its potential halting in the future. However, while this is true for individual TMs, the situation may be

different for the statistics of subclasses of TMs. In other words, for a given class of TMs, the probability of an instance halting may be found to increase or decrease for the future given that it has not yet halted.

We performed an empirical study of randomly generated TMs described in Appendix E. The results seem to show that TMs break into two classes—those that halt within a small number of steps (e.g., 16) and those that do not halt. Indeed, the data suggest that approximately one-third of the sampled TMs halt, while two-thirds do not. For those that halt, the probability of halting at the first step is greatest and then decreases exponentially and so is well-described by a geometric distribution with probability of success approximately $1/6$. In other words the probability of halting is $1/6$ at each step given the TM has not halted earlier for the set that will eventually halt.

Summarizing, the existence of an iteratively specified system is not algorithmically decidable but there may be useful probabilistic formulations that can be applied to sub-classes of such specifications. For example, it seems to be that with high probability the halting of a two-symbol, three-state TM can be decided within 16 simulation steps. Therefore, it might be tractable for an assemblage of components to find a way to solve its composition problems in a finite time for certain classes of components couplings.

5. Discussion: Future Research

The discussion presented here was limited to compositions having two components with cross-coupled connections. However, this scope was sufficient to expose the essential effect of feedback in creating the need for consistent assignments of input/output pairs and the additional—distinct—requirement for time progression. Further development can extend the scope to arbitrary compositions generalizing the statement of consistent assignments to finite and perhaps, infinite sets of components.

Further, when extending to arbitrary composition sizes, the analysis revealing fundamental conditions for the emergent existence of systems from component systems suggests new classes of systems that can be defined using iterative specifications. For example, we might consider the active–passive systems in a more general setting. First allowing any finite number of them in a coupling can satisfy the requirement for at most one active component at any time by restricting the coupling to a single influencer for each component. In fixed coupling exhibiting feedback, this amounts to a cyclic formation in which activity will travel around the cycle from one component to the next. Another common pattern is a single component (root) influencing a set of components (branches) each of which influences only the root. Several possibilities for interaction of the root with the branches can result in admissible compositions. For example, the root might select a single branch to activate to satisfy the single active component requirements. Alternatively, it might activate all branches and count the number of branches as they successively go passive, becoming active when all the branches have become passive. These alternative configuration require knowledge of the protocol underlying the interaction but this only means that proofs of emergent well-defined resultants will be conditioned to narrower sub-classes of systems rather than to broad classes such as all differential equation systems.

After much progress in understanding how dynamic structure can be managed in DEVS-based simulations [14–16], a framework that incorporates several insights has recently been developed [17]. This framework, formulated in the DEVS formalism, can be extended to iterative specifications and investigated for conditions of existence as done here. The extension will probably increase the complexity required to state and establish the conditions but would, if successful, bring us closer to understanding the emergence of systems from assemblages of components in the real world.

Appendix A. Turing Machine as a DEVS

Appendix A.1. Tape System

A state is a triple $(tape, pos, mv)$ where $tape : \mathbb{I} \to \{0,1\}$, $pos \in \mathbb{I}$, $mv \in \{-1,1\}$ and $\mathbb{I}$ is the integers; in other words, the tape is an infinite sequence of bits, where *pos* represents the position of the head,

and *mv* is a specification for moving left or right. An internal transition moves the head as specified; an external transition accepts a symbol, move pair, stores the symbol in the current position of the head and stores the move for subsequent execution. The slot for storing the move can also be null, which indicates that a move has taken place.

Each function can be described as follows:

$$\begin{aligned}
&\delta_{int}(tape, pos, mv) = (tape, move(pos, mv), null)\\
&\delta_{ext}((tape, pos, null), e, (sym, mv)) = (store(tape, pos, sym), pos, mv)\\
&ta(tape, pos, mv) = 1\\
&ta(tape, pos, null) = \infty\\
&\lambda(tape, pos, mv) = getSymbol(tape, pos)
\end{aligned}$$

where

$$\begin{aligned}
&move(pos.mv) = pos + mv\\
&store(tape, pos, sym) = tape' \text{ where } tape'(pos) = sym,\ tape'(i) = tape(i)\\
&getSymbol(tape, pos) = tape(pos)
\end{aligned}$$

Appendix A.2. TM Control

A state is a pair (st, sym) where st is a control state and sym is a stored symbol. An internal transition applies the TM transition table to the (st, sym) pair and transitions to the specified control state. An external transition stores the received symbol for subsequent use.

Each function can be described as follows:

$$\begin{aligned}
&\delta_{int}(st, sym) = (TMState(st, sym), null)\\
&\delta_{ext}((st, null), e, sym) = (st, sym)\\
&ta(st, sym) = 1\\
&ta(st, null) = \infty\\
&\lambda(st, sym) = TMOutput(st, sym)
\end{aligned}$$

where

$$\begin{aligned}
&TMState(st, sym) = st'\\
&TMOutput(st, sym) = sym'
\end{aligned}$$

Appendix B. Iterative Specification of Systems

The following is extracted from [12] Chapter 5.

Appendix B.1. Generator Segments

Consider (*Z*,*T*) the set of all segments $\{\omega :< 0, t_1 > \rightarrow Z \mid t_1 \in T\}$, which is a semigroup under concatenation. For a subset Γ of (*Z*,*T*), we designate by Γ^+ the *concatenation closure* of Γ (also called the *semigroup generated* by Γ). Example generators are bounded continuous segments, and constant segments of variable length generating bounded piecewise continuous segments piecewise constant segments, respectively. Unfortunately, if Γ generates Ω (a subset of segments), we *cannot* expect each $\omega \in \Omega$ to have a unique decomposition by Γ; that is, there may be distinct decompositions $\omega_1, \omega_2, \ldots, \omega_n$ and $\omega'_1, \omega'_2, \ldots, \omega'_n$ such that $\omega 1 \bullet \omega 2 \bullet \cdots \bullet \omega_n = \omega$ and $\omega 1' \bullet \omega'_2 \bullet \cdots \bullet \omega'_n = \omega$.

A single representative, or *canonical* decomposition can be computed using a *maximal length segmentation.* First, we find ω_1, the longest generator in Γ that is also a left segment of ω. This process is repeated with what remains of ω after $\omega_1$is removed, generating ω_2, and so on. If the process stops after n repetitions, then $\omega = \omega_1\omega_2 \ldots \omega_n$.

It is not necessarily the case that MLS decompositions exist (i.e., that the just-mentioned processes will stop after a finite number of repetitions). Thus, we are interested in checkable conditions on a set of generators that will guarantee that each segment generated has an MLS decomposition. Fortunately, a segment can have at most one MLS decomposition. We say that Γ is an *admissible* set of generators

for Ω if Γ generates Ω and for each $\omega \in \Omega$, a unique MLS decomposition of ω by Γ exists. (We also say Γ admissibly generates Ω.) The following is proved in [12]:

Theorem B1. *Sufficient Conditions for Admissibility. If Γ satisfies the following conditions, it admissibly generates Γ^+:*

1. *Existence of longest initial segments: $\omega \in \Gamma^+ \Rightarrow max\{t \mid \omega_{t>} \in \Gamma\}$ exists*
2. *Closure under right segmentation: $\omega \in \Gamma \Rightarrow \omega_{<t} \in \Gamma$ for all $\tau \in dom(\omega)$*

Appendix B.2. Generator State Transition Systems

Having established a terminating process for obtaining MLS decompositions, we wish to use these decompositions to help us generate a transition function given only that function's action on the generators. In other words, let Ω_G be an admissible generating set for Ω and suppose we have defined a function $\delta_G : Q \times \Omega_G \to Q$, which we call a *single segment transition function.*

Let $\omega_1, \omega_2, \ldots, \omega_n$ be the MLS decomposition of ω. Having δ defined for each segment ω_i, we wish to piece together these parts to obtain compound transition associated with ω itself.

An *iterative specification* of a system is a structure

$$G =< T, X, \Omega_G, Y, Q, \delta, \lambda >,$$

where T, X, Y, and Q have the same interpretation as for I/O systems; Ω_G is the set of input segment generators; $\delta_G : Q \times \Omega_G \to Q$ is the single segment state transition function; $\lambda : Q \times X \to Y$ is the output function, with the restriction that $\Omega_G \subseteq (X, T)$; and, most important, Ω_G is an admissible set of generators and $\delta_G^+ : Q \times \Omega_G^+ \to Q$ has the composition property.

An iterative specification $G =< T, X, \Omega_G, Y, Q, \delta_G, \lambda >$ specifies a time invariant system, $S_G =< T, X, \Omega_G^+, Y, Q, \delta_G^+, \lambda >$ where δ_G^+ is the extension of δ_G which is well-defined according to the following:

Theorem B2. *Sufficient Conditions for Iterative Specification. Let $G =< T, X, \Omega_G, Y, Q, \delta_G, \lambda >$ be a structure as just defined. Then, if the following conditions hold, G is an iterative specification and $S_G =< T, X, \Omega_G^+, Y, Q, \delta_G^+, \lambda >$ is a system.*

1. *Existence of longest prefix segments: $\omega \in \Omega_G^+ \Rightarrow max\{t \mid omega_{t>} \in \Omega_G\}$ exists*
2. *Closure under right segmentation: $\omega \in \Omega_G \Rightarrow \omega_{<t} \in \Omega_G$ for $t \in dom(\omega)$*
3. *Closure under left segmentation: $\omega \in \Omega_G \Rightarrow \omega_{t>} \in \Omega_G$ for $t \in dom(\omega)$*
4. *Consistency of composition: $\delta_G^+(q, \omega_1 \bullet \omega_2 \bullet \cdots \bullet \omega_n) = \delta_G(\delta_G(\ldots \delta_G(\delta_G(q, \omega_1), \omega_2), \ldots), \omega_n)$*

Appendix C. DEVS Atomic Model Simulation of an Iterative Specification

Theorem C1. *A DEVS atomic model can simulate an Iterative Specification.*

Proof. As shown in Figure 3 and in Figure 5, given an iterative specification $G =< T, X, \Omega_G, Y, Q, \delta_G, \lambda >$, we construct a DEVS model $M =< X, S, Y, \delta_{int}, \delta_{ext}, \lambda, ta >$, such that

$$\begin{aligned}
&g : \Omega_G \to \Omega_{DEVS} \text{ using mls} \\
&g(\omega) = [\omega]_{l(\omega)>} = [\omega]\, \phi_{l(\omega)>} \\
&h : Q \to Q \times \Omega_G \\
&h(q) = (q, dummy)
\end{aligned}$$

According to the definition of the simulating DEVS:

$$\begin{aligned}
&S = Q \times \Omega_G \\
&\delta_{int}(q, \omega) = (\delta_G(q, \omega), dummy) \\
&ta(q, \omega) = l(\omega) \\
&\delta_{ext}((q, \omega), e, \omega') = (\delta_G(q, \omega_{e>}), \omega') \\
&\lambda(q, dummy) = \lambda_G(q)
\end{aligned}$$

In the Iterative Specification:

$$\delta_G^+(q, \omega\omega') = \delta_G(\delta_G(q, \omega), \omega')$$

In the DEVS, let δ_{DEVS} be the transition function of the system specified by the DEVS. We want to show that $h(\delta_G^+(q, \omega\omega')) = \delta_{DEVS}(h(q), g(\omega\omega'))$, i.e.,

$$\begin{aligned}
&\delta_{DEVS}(h(q), g(\omega\omega')) \\
&= \delta_{DEVS}((q, dummy), g(\omega)g(\omega')) \\
&= \delta_{DEVS}((q, dummy), [\omega]_{l(\omega)>}, [\omega']_{l(\omega')>}) \\
&= \delta_{DEVS}(\delta_{ext}((q, dummy), [\omega]), \phi_{l(\omega)>}), \omega'_{l(\omega')>}) \\
&= \delta_{DEVS}((q, [\omega]), \phi_{l(\omega)>}), \omega'_{l(\omega')>}) \\
&= \delta_{DEVS}(\delta_{int}(q, [\omega]), \phi_{l(\omega)>}), \omega'_{l(\omega')>}) \\
&= \delta_{DEVS}((\delta_G(q, \omega), dummy), \omega'_{l(\omega')>}) \\
&= (\delta_G(\delta_G(q, \omega), \omega'), dummy)
\end{aligned}$$

Thus,

$$\begin{aligned}
&h(\delta_G^+(q, \omega\omega')) \\
&= h(\delta_G(\delta_G(q, \omega), \omega')) \\
&= (\delta_G(\delta_G(q, \omega), \omega'), dummy) \\
&= \delta_{DEVS}(h(q), g(\omega\omega'))
\end{aligned}$$

□

Appendix D. Coupled Iterative Specification at the I/O System level

Theorem D1. *An admissible Coupled Iterative Specification specifies a well-defined Iterative Specification at the I/O System level.*

Proof. Given two iterative specifications, G_i and their state sets, Q_i $i = 1, 2$, let $Q = Q_1 \times Q_2$, eventually the state set of the iterative specification to be constructed. For any pair ($q1$,$q2$) in Q, let $(\rho 1, \rho 2)$ be a consistent output trajectory for $(q1, q2)$, i.e., $\rho 1 = \beta(q1, \rho 2)$ and $\rho 2 = \beta(q2, \rho 1)$ and $F(q1, q2) = (\rho 1, \rho 2)$.

Define an autonomous Iterative Specification $< Q, \delta_G, \lambda >$ by

$$\delta_G(q1, q2) = (\delta(q1, \rho 2_{t^*(q1,q2)>}), \delta(q2, \rho 1_{t^*(q1,q2)>}))$$

where $t^*(q1, q2) = \min\{t_1^*, t_2^*\}$ and t_1^*, t_2^* are the times of the MLS for $\rho 1$ and $\rho 2$.

In other words, since each of the component iterative specifications have maximum length segmentations we take the time of next update of the constructed specification to be determined by earliest of the times of these segmentations for the consistent pair of output trajectories. This allows us to define a step-wise transition for the constructed transition function. Closure under composition for this transition function can be established using induction on the number of generators in the segments under consideration. Similarly, the output function is defined by: $\lambda(q1, q2) = (\lambda(q1), \lambda(q2))$. □

Theorem D2. *The set of Iterative Specifications is closed under admissible coupling.*

Proof. This theorem is a corollary of the previous theorem. □

Appendix E. A Statistical Experiment Sampling from 2-Symbol, 3-State TMs

We performed a statistical experiment sampling from two-symbol, three-state TMs. Each sampled TM fills in the xs in Table E1 with values from the sets shown, where one of the rows is chosen at random and its next state change to the halt state (e.g., the first row can be (A,−1,0) meaning jump to state A, move back, and print 0). The number of TMs in this class is $(2 \times 2 \times 3)^{2\times 3} = 12^6$. We generated 1000 samples randomly and simulated them until they halted or for a maximum of 1000 time steps.

Table E1. Template for two-symbol, three-state TMs.

State	Symbol	Next State {A, B, C}	Move {1, −1}	Print Symbol {0, 1}
A	0	x	x	x
A	1	x	x	x
B	0	x	x	x
B	1	x	x	x
C	0	x	x	x
C	1	x	x	x

The results are shown in the first two columns of Table E2. They seem to show that TMs break into two classes—those that halt within a small number of steps (e.g., 16) and those that do not halt. Indeed, the data in the table suggest that approximately one-third of the sampled TMs halt, while two-thirds do not. For those that halt, the probability of halting at the first step is greatest and then decreases exponentially (cf. Figure E1). The third row shows the probabilities computed from a geometric success model using probability of success as 0.161. The agreement of the observed and predicted distributions are in close agreement. This suggests that for the set that will eventually halt, the probability of a TM halting at each step is approximately 1/6, given that it has not halted earlier.

Table E2. Empirical Frequency Distribution for halting step vs. Geometric Model.

Halts at N	Frequency	Geometric Model
1	0.161	0.161
2	0.1	0.09
3	0.042	0.039
4	0.023	0.022
5	0.016	0.015
6	0.005	0.00488
7	0.004	0.00390
11	0.001	0.000990
16	0.001	0.000985
10000	0.647	

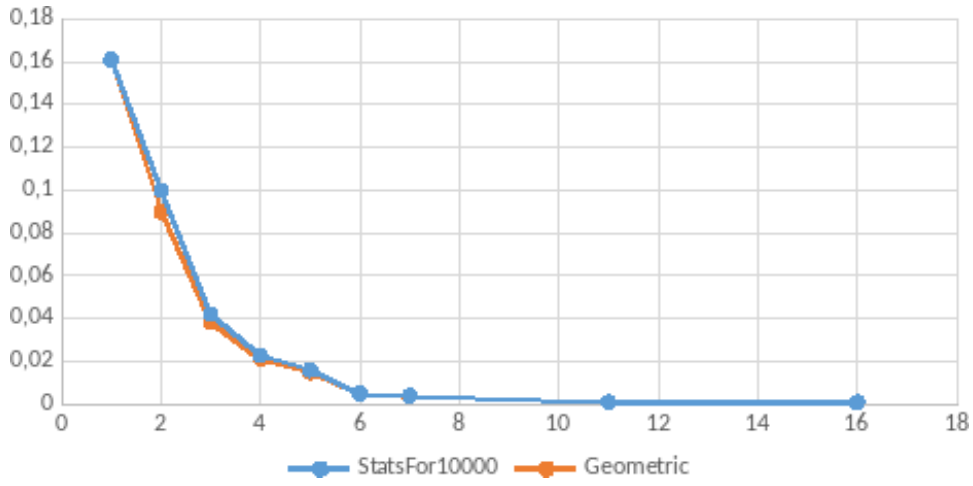

Figure E1. Empirical frequency vs. geometric model.

Summarizing, the existence of an iteratively specified system is not algorithmically decidable but there may be useful probabilistic formulations that can be applied to sub-classes of such specifications. For example, it seems to be that with high probability the halting of a two-symbol, three-state TM

can be decided within 16 simulation steps. Therefore, it might be tractable for an assemblage of components to find a way to solve its composition problems in a finite time for certain classes of components couplings.

References

1. Mittal, S. Emergence in stigmergic and complex adaptive systems: A formal discrete event systems perspective. *Cogn. Syst. Res.* **2013**, *21*, 22–39.
2. Mittal, S.; Rainey, L. Harnessing emergence: The control and design of emergent behavior in system of systems engineering. In Proceedings of the Conference on Summer Computer Simulation, SummerSim'15, Chicago, IL, USA, 26–29 July 2015; pp. 1–10.
3. Ashby, W. *An Introduction to Cybernetics*; University Paperbacks: Methuen, MA, USA; London, UK, 1964.
4. Foo, N.Y.; Zeigler, B.P. Emergence and computation. *Int. J. Gen. Syst.* **1985**, *10*, 163–168.
5. Kubik, A. Toward a formalization of emergence. *Artif. Life* **2003**, *9*, 41–65.
6. Szabo, C.; Teo, Y.M. Formalization of weakemergence in multiagent systems. *ACM Trans. Model. Comput. Simul.* **2015**, *26*, 6:1–6:25.
7. Ören, T.I.; Zeigler, B.P. System Theoretic Foundations of Modeling and Simulation: A Historic Perspective and the Legacy of A. Wayne Wymore. *Simul. Trans. Soc. Model. Simul.* **2012**, *88*, 1033–1046.
8. Zeigler, B.; Muzy, A. Some Modeling & Simulation Perspectives on Emergence in System-of-Systems. In Proceedings of the SpringSim2016, Virginia Beach, VA, USA, 23–26 April 2016; pp. 11:1–11:4.
9. Wikipedia. Available online: https://en.wikipedia.org/wiki/Turing_machine (accessed on 2 November 2016).
10. Uhrmacher, A.M. Dynamic structures in modeling and simulation: A reactive approach. *ACM Trans. Model. Comput. Simul.* **2001**, *11*, 206–232.
11. Nutaro, J. *Building Software for Simulation: Theory and Algorithms with Applications in C++*; Wiley: Hoboken, NY, USA, 2011.
12. Zeigler, B.; Praehofer, H.; Kim, T. *Theory of Modeling and Simulation: Integrating Discrete Event and Continuous Complex Dynamic Systems*; Academic Press: New York, NY, USA, 2000.
13. Giambiasi, N.; Escude, B.; Ghosh, S. GDEVS: A generalized discrete event specification for accurate modeling of dynamic systems. In Proceedings of the Autonomous Decentralized Systems, Dallas, TX, USA, 26–28 March 2001; pp. 464–469.
14. Barros, F.J. Modeling and Simulation of Dynamic Structure Heterogeneous Flow Systems. *SIMULATION Trans. Soc. Model. Simul. Int.* **2002**, *78*, 18–27.
15. Barros, F.J. Dynamic Structure Multi-Paradigm Modeling and Simulation. *ACM Trans. Model. Comput. Simul.* **2003**, *13*, 259–275.
16. Steiniger, A.; Uhrmacher, A.M. Intensional couplings in variable-structure models: An exploration based on multilevel-DEVS. *ACM Trans. Model. Comput. Simul.* **2016**, *26*, 9:1–9:27.
17. Muzy, A.; Zeigler, B.P. Specification of dynamic structure discrete event systems using single point encapsulated control functions. *Int. J. Model. Simul. Sci. Comput.* **2014**, *5*, 1450012.