Iterative specification as a modeling and simulation formalism for I/O general systems

Alexandre Muzy*, Bernard P. Zeigler~, Franck Grammont†

Abstract—**Beyond functional systems, a vast class of models do not exhibit analytical solutions. Simulation is often used to approximate these solutions in a computational manner. An alternative is to include the simulation structure as a part of a modeling formalism. This article aims at exploiting iterative system specification as a modeling formalism. The systems specified are dynamic with inputs/outputs (I/O) that can be coupled in a modular way. The iterative specification consists of a decomposition of the I/O behavior of a system into trajectory segments. We prove that specific decompositions can be combined while ensuring that the overall I/O behavior is correctly represented. While generic in nature, the approach is verified against leaky integrate and fire neuronal modeling applications.**

I. Introduction

The increasing role of simulation as a fundamental tool for scientists, raises more intensive questions at the theoretical level. In this context, simulation is more than a model implementation (or experimentation) on a computer. Indeed, some of the simulation mechanisms may tend to become part of the model structure itself. However, it is important not to conflate the separation between model (real system referent) and simulator (computational substrate). A proper accounting can unify both static and dynamical system specifications.

In this context, theory of modeling and simulation [1], [2] has been grounded on the iterative specification of general systems (for references on the mathematical foundations of general system theory, see [3], [4], [5], [6], [7], [8], [9], [10]). The iterative system specification, very general in its principle, can be intuitively conceived as a mapping between the continuous trajectories of the general system and its state-based representation. At a basic cognitive level, states consist of constant values of the descriptive variables of a model. Then, the fundamental idea is to focus on the time intervals where these variable values do not change thus *segmenting* the system behavior into variable length parts. In this way, the trajectory of a dynamic system is cut into segments. These segments, subsegments of the original segment, are the objects exchanged and computed by model components [11]. In this way, iterative specification has been used as an intermediate specification between Discrete Event System Specification (DEVS) at the lower (computational) level and general system behavior at the higher level. Since DEVS has been proved to be equivalent to such iterative specification, DEVS can be used to specify general system structure and compute its behavior. Based on the segmentation of a general system trajectory, DEVS achieves a state based realization of an input-output relation [12], [13]. Although the iterative system specification principle has been used to ground the theory, it has never been exploited as a modeling formalism ***per se***. Hence, the main goal of this article is to formulate the iterative system specification as a modeling formalism for dynamic systems. To achieve this goal, we will show how to combine specific segmentations to build a proper set of trajectories at the system level and choose between different specifications. The general segmentation approach is applied to discrete-event trajectories as particular objects of interest. Finally to confirm its utility, this generic modeling formalism will be tested in the domain of neuronal networks.

This article is organized as follows. In Section 2, a new compact reformulation of iterative specification of systems is proposed. In Section 3, new methods for segmentation of trajectories are derived from the standard segmentation theory and applied to discrete-event trajectories. Specific segmentations are defined that can be combined in a modular way. In Section 4, this extended modeling framework is used to build neuronal models. In Section 5, a discussion links segmentation and a family of system specifications using neuron parameters. Finally, a conclusion closes this article with some new perspectives implied by our introduction of iterative specification as a modeling formalism.

II. Mathematical framework

The dynamics of systems can be iteratively specified by segmenting their trajectories (i.e., cutting the trajectories into subsegments (cf. Definition 1)). This segmentation has to ensure certain properties to preserve the equivalence between the concatenated subsegments and the original trajectories (cf. Definition 2). At the iterative specification level (cf. Definition 4), the concatenated segments need to be properly concatenated to properly specify a general system (cf. Definition 5). More precisely, the segmentation needs to satisfy certain requirements to enable the generated trajectory to properly represent a general system (cf. Theorem 7).

A. Concatenation/composition of segments into higher-level segments

The trajectory of a system usually consists of a *time function* $f : T \rightarrow Z$, with T the *time base* and Z *whatever set* (i.e., of inputs, outputs or state of a model). A restriction of a time function (or trajectory) to a time interval consists of a *segment*.

*Université Côte d'Azur, CNRS, I3S, France, Email: alexandre.muzy@cnrs.fr.

~Chief Scientist, RTSync Corp, 530 Bartow Drive Suite A Sierra Vista, AZ 85635, United-States of America.

†Université Côte d'Azur, CNRS, LJAD, France.

More precisely,

Definition 1. *[Modified from [2]] A segment is a partial function $\omega : [0, t_f) \rightarrow Z$, with $t_f \in \mathbb{R}^+$ the duration of ω, Z a set of values.*

Segments share the same duration but not necessarily the same domain. This allows systems to interact over a hybrid time base. For example, a segment $\omega_X \in \Omega_X$ can be defined over a continuous time base ($T_c = dom(\omega_X) = [0, t_f) \cap \mathbb{R}^+$) while an output segment $\omega_Y \in \Omega_Y$ can be defined over a discrete time base ($T_d = dom(\omega_Y) = [0, t_f) \cap \mathbb{N}$). A resulting hybrid time base then consists of $T = T_c \times T_d$. The domain of each segment is then obtained by projecting the time base.

As shown in Figure 1, segments can be *concatenated* or *composed* together.

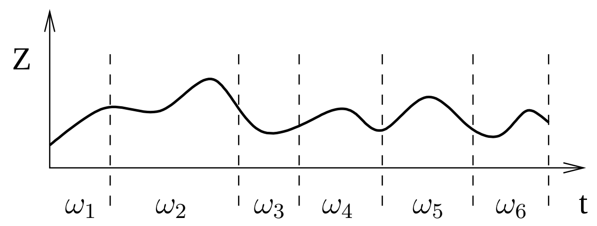


Figure 1. Concatenation of segments.

More precisely, to compose consistently segments together, closure under composition can be formally defined as:

Definition 2. A set of segments Ω is said to be *closed under composition/concatenation* if for every pair of contiguous segments $\omega_{t)} \in \Omega$ and $\omega_{[t} \in \Omega^+$, we have $\omega = \omega_{t)} \bullet \omega_{[t}$, where $\omega \in \Omega$ is a segment (defined over $[0, t_f)$), $\omega_{t)} \in \Omega$ is the *left segment of* ω (defined over $[0, t)$), $\omega_{[t} \in \Omega$ is the *right segment of* ω (defined over $[t, t_f)$), and $\bullet : \Omega \times \Omega \rightarrow \Omega$ is the *concatenation/composition operation*.

B. Unique segmentation/decomposition of segments into segments

Figure 1 can be re-interpreted as considering that a segment of a dynamic system can be cut (segmented) into other segments. This last operation is called a *segmentation/decomposition.*

Considering the set of all segments Ω, whether it is to segment or concatenate segments of this set, we would like to find a subset of segments (the *set of segment generators*), $\Omega_G \subseteq \Omega$, to generate it. The *set of generated segments* (or *composition closure* of Ω_G), $\Omega_G^+ \supseteq \Omega$, is then the set of all possible concatenations of segment generators. If the set of segment generators Ω_G satisfies $\Omega_G^+ = \Omega$ it is said to be the set of segment generators for Ω.

A major difficulty now is that the set of generated segments Ω_G^+ is possibly infinite because of the continuous nature of system's time base. To ensure that a unique decomposition can be selected from the infinite number of possible decompositions that exist, the process of Maximal Length Segmentation (MLS) can be used. Algorithm 1 describes the iterations enabled by an MLS (taken from [14]). In an MLS, a segment $\omega \in \Omega$ is successively decomposed into segment generators taken from Ω_G. Each *cut* decomposes the segment $\omega \in \Omega$ into two subsegments, whereby the left-cut is always from Ω_G, whereas the right-cut may still be from Ω. The time breaking point $t \in \mathbb{R}^+ \cup \{+\infty\}$ is determined to *maximize the length of the left-cut segment* $\omega_{t)}$. The latter constitutes the first element of the generated segments $\omega_1 \in \Omega_G^+$. The remaining segment then consists of the right-cut segment $\omega_{[t} \in \Omega$. The cut is iterated until the right-cut segment $\omega_{[t}$ belong to Ω_G. For recursive validity, an MLS algorithm is required to finish in a finite number of steps.

Algorithm 1 Maximal length segmentation algorithm.

```
Variables:
  $\omega \in \Omega$: Initial segment
  $t$: Time breaking point
  $i$: Segment index
Begin
    function MLS($\omega \in \Omega$)
        repeat
            $t \leftarrow max\{t \,|\, \omega_{t)} \in \Omega_G\}$
            $\omega_{t)} \leftarrow$ cut-left($\omega, t$)
            $\omega_i \leftarrow \omega_{t)}$
            $\omega \leftarrow \omega_{[t}$
            $i$++
        until $\omega_{[t} \in \Omega_G$
        $\omega_n \leftarrow \omega_{[t}$
        return $\omega_1 \bullet ... \bullet \omega_i \bullet ... \bullet \omega_n$
    end function
End
```

MLS is a fundamental mechanism to ensure that segments can be decomposed uniquely. On the other hand, as we will see,

Remark 3. MLS can be used to ensure that segment generators can be composed uniquely. Indeed, considering a set of segment generators, concatenating any two segments should satisfy MLS (existence of longest prefix segments as well as left and right segmentations) to ensure that it results also in a segment generator and then that the set of segment generators is closed under composition.

C. Deterministic iterative system specification

In previous subsections we described how to compose segments into another segment and conversely how to decompose a segment into other segments. Being able to do both composition and decomposition of segments ensures that segment operations are consistent. These operations are described here at the system's level. Three main entities are defined:

1) The iterative system specification manipulating generators,
2) The process concatenating/composing/generating segments together, resulting in
3) A deterministic input-ouput general system [7].

Formally, a system is iteratively specified by

Definition 4. *[Modified from [2]]* A *Deterministic Iterative System Specification (*DetIterSpec*)*

$$\text{DetIterSpec} = (\delta, \lambda)$$

where

- $\delta : Q \times \Omega_G \rightarrow Q$ is the *single segment state transition* function, with Q the *set of states* and $\Omega_G \subseteq X^T$ the set of *admissible input segment generators* (a subset of all functions from *time set* T to *input set* X). A transition from a state $q \in Q$ to another state $q' \in Q$, for an *input segment* $\omega \in \Omega_G$ is unique and consists of $q' = \delta(q, \omega)$.
- $\lambda : Q \rightarrow Y$ is the *output* function.

At the iterative specification level, the concatenation (or composition) closure of segments needs to ensure the composition of the resulting continuously defined segments at the general system level. Based on an iterative specification, the problem is to find a consistent segmentation that is unique and able to produce iteratively the whole system dynamics by concatenating the subsegments into complete segments at the general system level. Making an analogy with automata theory, $(\delta, \lambda) \overset{(+,\lambda)}{\rightarrow} (\Delta, \lambda)$ is a compact notation showing that an iterative specification (δ, λ) is an extension of the concatenation closure process where an alphabet is concatenated until closure is obtained. Such closure puts together letters to make strings (as languages) in a system (Δ, λ) (as defined after and in [7]).

The *DetIterSpec* transition function is extended to account for concatenation/composition process:

Definition 5. *[Modified from [2]] A concatenation/composition process is a structure*

$$\text{COPRO} = (\delta^+, \lambda)$$

where $\delta^+ : Q \times \Omega_G^+ \rightarrow Q$ is the *global state transition function* (the extension of δ to Ω_G^+), and Ω_G^+ is the *set of generated input segments* (or concatenated input segments), and $\lambda : Q \rightarrow Y$ is the *output* function.

Finally the general system structure is obtained:

Definition 6. *[Modified from [2]] A Deteterministic Input-Output General System is a structure*

$$\text{DetIOGSYS} = (\Delta, \lambda)$$

where $\Delta : Q \times \Omega \rightarrow Q$ is the *state transition function*, and $\Omega \subseteq X^T$ is the *set of admissible input segments*, and $\lambda : Q \rightarrow Y$ is the *output* function.

To sum up, following an MLS, an iterative specification allows computing (eventually by simulation) the whole state trajectory along the entire input segment following the requirements:

Theorem 7. *[Modified from [2], p. 121] Sufficient conditions for iterative specification*

An iterative specification $G = (\delta, \lambda)$ can be associated to a system $S_G = (\Delta, \lambda)$ if the following conditions hold:

1) Existence of longest prefix segments*: $\omega \in \Omega_G^+ \implies max\{t \,|\, \omega_{t)} \in \Omega_G\}$ exists*
2) Closure under right segmentation*: $\omega \in \Omega_G \implies \omega_{[t} \in \Omega_G$ for all $t \in dom(\omega)$*
3) Closed under left segmentation*: $\omega \in \Omega_G \implies \omega_{t)} \in \Omega_G$ for all $t \in dom(\omega)$*
4) Consistency of composition*: $\omega_1, \omega_2, \ldots, \omega_n \in \Omega_G$ and $\omega_1, \omega_2, \ldots, \omega_n \in \Omega_G^+$ $\implies \delta^+(q, \omega_1 \bullet \omega_2 \bullet \ldots \bullet \omega_n) = \delta(\delta(\ldots\delta(\delta(q, \omega_1), \omega_2), \ldots), \omega_n)$.*

Requirements 2 and 3 are quite strong. They can be relaxed:

Remark 8. According to requirements 2 and 3, any input generator can be segmented at any time point leading to left and right subsegments. From ([2], p. 123), left and right segmentability can be *weakened* while still enabling Theorem 7 to hold (the stated conditions are *sufficient* but not *necessary*). To do so, both conditions are replaced by

For $\omega \in \Omega_G^+$ if $t^* = max\{t \,|\, \omega_{t)} \in \Omega_G\}$ and $t_k < t^* < t_{k+1}$ then $\omega_{[t_k, t^*)} \in \Omega_G$ and $\omega_{[t^*, t_{k+1})} \in \Omega_G^+$.

For example, this weakened condition does away with the need for right segmentation if there is no remaining right segment at the end of each cycle of the MLS Algorithm 1. A particular example is where all the segments are composed of positive ramps, one ramp starting where the other ends. The corresponding set of input segment generators then consists of $\Omega_G^R = \{\omega \,|\, \exists a > 0 : \omega(t) = at, \, \omega(0) = 0, \, t \in dom(\omega)\}$. The corresponding set of generated segments Ω_G^{R+} is closed under left segmentation *but not* under right segmentation (as a right segment of a ramp does not start from *0*).

III. Specific maximal length segmentations

Previous segmentation requirements from Theorem 7 are extracted to focus on proper generated segment sets obtained by segmentation (cf. Definition 11). MLS, as presented in [2], consists of a general principle to allow a sound/formal building of a system specification hierarchy. It is proposed here to apply this concept to the vast class of computational models. To achieve this goal, specific MLS processes will be proposed (cf. Definition 12). These specific MLS are combined to build *proper* generated segment sets (through the intersection of corresponding generated sets, cf. Theorem 24) thus ensuring iterative specification. However, other kinds of combinations of generated segment sets can also be *proper* (cf. the disjoint union combination in Example 23) or *not necessarily proper* (cf. the union combination in Remark 22). Examples concern discrete-event segments (cf. Definition 14) and corresponding generators (cf. Definition 15) while Theorem 24 is general and applies to the whole set of possible generated segments. Examples of specific MLS applications show different combinations. Besides, the specific segmentations are based on the length or on the number of events in segments, which is as we will see, a very general kind of segmentation.

A. Specific maximal length segmentations

A set of generated segments (which depends on MLS) can be *improper*, i.e., every generated segment $\omega \in \Omega^+$ may not be segmentable to elements of the generator segment set, $\omega \in \Omega_G$. This can help if the segmentation is not in coherence with the generator segment set Ω_G, i.e., the latter is missing generators of the generated segments and conversely, the set of generated segments (and corresponding system) will not be proper. Therefore, requirements need to be set to ensure that the set of generated segments, Ω_G^+, is proper by segmentation. Three definitions of propriety of the set of generated segments, Ω_G^+, are proposed, requiring increasing segmentation constraints.

First, a general definition requires *only* the right segment left over to be recursively *proper.*

Definition 9. *A set of generated segments* Ω_G^+ (resp. *a set of segment generators* Ω_G) is *proper* when, satisfying a maximal length segmentation, it follows the two requirements:

1) *Existence of longest prefix segments*: $\omega \in \Omega_G^+ \implies max\{t \,|\, \omega_{t)} \in \Omega_G\}$ exists, with $\omega_{t)} \in \Omega_G$, the left segment generator,
2) *Propriety of the right segment:* The right segment left over $\omega_{[t} \in \Omega_G^+$, is *proper*.

As in Theorem 7, segmentation closure requirements can be *weak*.

Definition 10. *A set of generated segments* Ω_G^+ (resp. *a set of segment generators* Ω_G) is *weakly proper* when, satisfying a maximal length segmentation, it follows the two requirements:

1) *Existence of longest prefix segments*: $\omega \in \Omega_G^+ \implies max\{t \,|\, \omega_{t)} \in \Omega_G\}$ exists,
2) *Weakened segmentability:* For $\omega \in \Omega_G^+$ *if* $t^* = max\{t \,|\, \omega_{t)} \in \Omega_G\}$ and $t_k < t^* < t_{k+1}$ th*en* $\omega_{[t_k,t^*)} \in \Omega_G$ and $\omega_{[t^*,t_{k+1})} \in \Omega_G^+$.

As in Theorem 7, segmentation closure can be required over both left and right segments.

Definition 11. *A set of generated segments* Ω_G^+ (resp. *a set of segment generators* Ω_G) is *strongly proper* when, satisfying a maximal length segmentation, it follows the two requirements:

1) *Existence of longest prefix segments*: $\omega \in \Omega_G^+ \implies max\{t \,|\, \omega_{t)} \in \Omega_G\}$ exists
2) *Closure under left segmentation*: $\omega \in \Omega_G \implies \omega_{t)} \in \Omega_G$ for all $t \in dom(\omega)$
3) *Closure under right segmentation*: $\omega \in \Omega_G \implies \omega_{[t} \in \Omega_G$ for all $t \in dom(\omega)$

These definitions are fundamental for the system theory of computational systems. They bring a new light and open new theoretical fields. Usual Definition 11 inherited from original modeling and simulation theory [2] requires both closure under left and right segmentation closure. Both are strong but not necessary requirements. Left segmentation closure is a fundamental assumption of the theory stipulating that at any time, the state of a system can be determined. This allows concatenating together generated segments to *compose* trajectories at system level. Relaxing these requirements, as we will see, allows composing higher level segment generators while still leading to unique segmentations.

Different specific MLS processes are then possible, each one having its own *segmentation condition*. Compared with general MLS as defined in subsection II-B, the following derivation is set:

Definition 12. *A specific maximal length segmentation* is based on both generator segment set Ω_G and generated segment set Ω_G^+, fulfilling *segmentation condition* $c : \Omega_G \to \mathbb{B}$, i.e., $\Omega_G = \{\omega \,|\, c(\omega) = true\}$ and $\Omega_G^+ = \{\omega \,|\, c(\omega_{t)}) = true\}$. General function MLS$(\omega \in \Omega)$ of Algorithm 1 is derived as a specific MLS adding: (i) the segmentation condition such that MLS$(\omega \in \Omega,\, c(\omega_{t)}))$, and (ii) the maximum time breakpoint detection such that $t \leftarrow max\{t \,|\, \omega_{t)} \in \Omega_G \wedge c(\omega_{t)}) = true\}$.

As introduced in Remark 3, now disposing of a proper set of segment generators Ω_G allows concatenating segment generators into higher-level segment generators. To ensure this property the closure under composition of an admissible set of segment generators Ω_G can be checked:

Definition 13. *A set of segment generators* Ω_G, admissibly generating a *proper* set of generated segments Ω^+, is *closed under composition* if for every pair of contiguous segment generators $\omega_{t)} \in \Omega_G$ and $\omega_{[t} \in \Omega_G$: $\omega = \omega_{t)} \bullet \omega_{[t}$, where $\omega \in \Omega_G$ is also a segment generator, $\bullet : \Omega_G \times \Omega_G \to \Omega_G$ is the *generator concatenation/composition operation,* and concatenation respects segmentation condition, i.e., $t \leftarrow max\{t \,|\, \omega_{t)} \in \Omega_G \wedge c(\omega_{t)}) = true\}$.

The set of segments Ω is now restricted to a simple (general) case of *discrete-event segments.* Discrete-events consist of discrete values at precise time points and no values at other time points. Therefore, discrete-events are used to focus on value changes in a system (at input, state or output levels). The set of discrete-event segments can be formally defined as:

Definition 14. *[Modified from [2]] The set of all discrete-event segments* consists of $\Omega_E = \{\omega \,|\, \omega : T_{[t_0,t_n)} \to Z \cup \{\phi\}\}$, with ω a *partial function*, Z an *arbitrary set*, ϕ the *nonevent*. There exists a finite set of time points $t_0, t_1, \ldots, t_{n-1} \in [t_0, t_n)$ such that $\omega(t_i) = z_i$ for $i = 0, \ldots, n-1$ and $\omega(t) = \phi$ for all other $t \in [t_0, t_n)$.

Concatenating discrete-event segments requires relying on a set of discrete-event segment generators. Furthermore, defining basic discrete-event segment generators to be used in various segmentations simplifies the search of segmentation solutions and their combination:

Definition 15. *The set of basic discrete-event segment generators* Ω_G^E, corresponding to the set of discrete-event segments Ω_E consists of $\Omega_G^E = \Omega_G^Z \cup \Omega_G^\phi$ with *the set of event segment generators* such that $\Omega_G^Z = \{z_i \,|\, z_i(0) = z\}$, where Z is an *arbitrary set*, and *the set of nonevent segment generators is* $\Omega_G^\phi = \{\phi_i \,|\, \phi_i : [0, t_i) \to \{\phi\},\, t_i \in \mathbb{R}^+\}$, where t_i is *the end time of corresponding nonevent* segment *generator* ϕ_i.

Two specific MLS examples will now be discussed. The first one is based on event counting segmentation while the second one is based on time length segmentation. This is a fundamental aspect of many computational systems that can be modeled either based on state change (e.g., discrete-event models) or on time change (e.g., discrete-time models).

Example 16. Segmentation by counting events to a fixed number

The set of all event segments having a number of events less or equal to N (for $N \geq 0$), is defined as a subset of all discrete event segments as $\Omega_N \subseteq \Omega_E$ (cf. Figure 2).

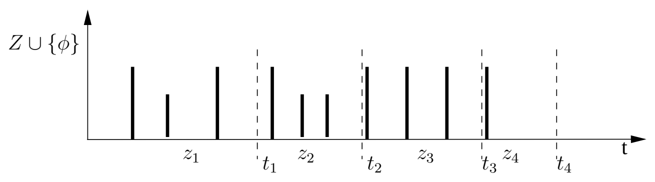


Figure 2. Segmentation by counting the number of events such that $N = 3$.

The generated set Ω_G^{N+} is *strongly proper*, i.e., following Definition 11:

1) Existence of longest initial segments: Considering any discrete-event segment in Ω_N, an initial segmentation of the latter (being a concatenation of the basic discrete-event segment generators of Definition 15) can be set easily based on the generator segment set $\Omega_G^N = \{\omega \,|\, c(\omega) = \mathit{true}\}$, with segmentation condition $c(\omega_{t)}) = \begin{cases} \mathit{true} & \mathit{if}\, n_E \leq N \\ \mathit{false} & \mathit{otherwise} \end{cases}$, where $n_E = \Sigma_{\hat{t}\in[0,t)}\chi_{\omega_{t>}(\hat{t})\neq\phi}$ is the *number of events* in segment $\omega_{t)}$, with χ the *indicator function*.
2) Closure under left and right segmentations: Subsegments of a segment have no more events than the segment itself.
3) Based on both basic discrete-event segment generators of Definition 15, using MLS, and a fixed number segmentation condition c, every possible discrete-event segment $\omega \in \Omega_E$ can be segmented.

Example 17. Segmentation by measuring segment lengths to a fixed length

The set of all event segments having a length less or equal to T (for $T \geq 0$), is defined as a subset of all discrete-event segments as $\Omega_T \subseteq \Omega_E$ (cf. Figure 3). It is obvious that the set of generated discrete-event segments Ω_G^{T+} is also *strongly proper*.

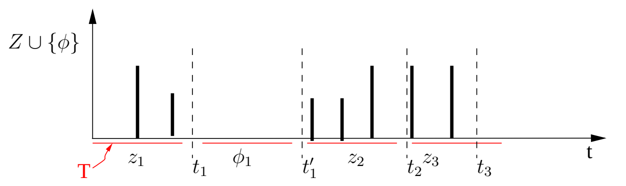


Figure 3. Segmentation by measuring time intervals such that $l(\omega) \leq T$. Notice that only the last generator segment is not exactly of size T.

Example 18. Segmentation by fixed segments

The set of all event segments having a length equal to $\overline{T}$ (for $\overline{T} \geq 0$), is defined as a subset of all discrete-event segments as $\Omega_{\overline{T}} \subseteq \Omega_E$ (cf. Figure 3). The set of generated discrete-event segments $\Omega_G^{\overline{T}+}$ is *proper,* depending on the size of $\overline{T}$ compared with the total size of segment $l(\omega)$. Notice that it cannot be strongly proper as segment generators cannot be split into subsegment generators. Let us see here the closure property of the set of admissible segment generators $\Omega_G^{\overline{T}}$ corresponding to the proper generated set $\Omega_G^{\overline{T}+}$. We prove it by induction. At a first segmentation step 1, it is obtained $\omega = \omega^1_{t)} \bullet \omega^1_{[t}$ with $t = \overline{T}$, $\omega^1_{t)} \in \Omega_G^{\overline{T}}$ and $\omega^1_{[t} \in \Omega_G^{\overline{T}+}$. The same process is achieved until obtaining decomposition $\omega_1 \bullet ... \bullet \omega_n$, with ω_n the last right segment. Then, $\Omega_G^{\overline{T}+}$ is *proper.*

The same proof arguments can be extended to the variable segment length segmentation of Example 17. By induction, at a first segmentation step 1, it is obtained $\omega = \omega^1_{t)} \bullet \omega^1_{[t}$ with $t = \alpha T$ and $\alpha \in (0, 1)$ and $\omega^1_{[t} \in \Omega_G^{T+}$. The same process is achieved until obtaining decomposition $\omega_1 \bullet ... \bullet \omega_n$, with ω_n the last right segment. Considering the total length of *n-1* concatenated segments as $L = \Sigma_{i\in\{1,...,n-1\}} l(\omega_i) \leq T - \alpha T$, this means that the last right segment ω_n can be segmented into $\omega^n_{t)} \bullet \omega^n_{[t}$ (both of length less than T) and previous *n-1* segments concatenated with $\omega^n_{t)}$ while still respecting segmentation condition $\Sigma_{i\in\{1,...,n-1\}} l(\omega_i) + l(\omega^n_{t)}) \leq T$. This means that Ω_G^{T+} is *proper.*

B. *Combination of specific maximal length segmentations*

Definitions 9 to 11 exposed the requirements for a generated segment set Ω_G^+ (resp for a segment generator set Ω_G) to be proper. Here, we explore formally the different combinations/operations (by intersection, disjoint union and union) of proper generated segment sets (resp. segment generator sets) and verify if the combination is also proper. First specific combination examples are shown for each operation to discuss their propriety. After these results are sum up and generalized.

Segmentations use two different sets: the generated segment set Ω_G^+ (a concatenation of segments) and a generator segment set Ω_G (an "alphabet" of segments). The combination of generated segment sets, $\Omega_G^+ \cap {\Omega'_G}^+$, should not be confounded with the combination of generator segment sets, $\Omega_G \cap \Omega'_G$, thus leading to the following remark.

Remark 19. Notice that the intersection of two segment generators sets may be empty, i.e., $\Omega_G \cap \Omega'_G = \{\phi\}$, which does not mean that the corresponding intersection of generated segment sets $\Omega_G^+ \cap \Omega'^+_G$, does not exist and/or is not proper.

As shown previously different specific segmentations lead to different generated segments. Let us see a simple example of combination (intersection) of two specific segmentations and how proper it is.

Example 20. Intersection of two proper generated segment sets that is proper

The *intersection* of two proper generated segment sets, $\Omega_G^+ \cap {\Omega'_G}^+$, obtained by specific MLS decomposition of Example 16, with condition

$$c(\omega_{t>}) = \begin{cases} \mathit{true} & \mathit{if}\, n_E \leq N \\ \mathit{false} & \mathit{otherwise} \end{cases},$$

and by specific MLS decompositions of Example 17, with condition

$c'(\omega_{t>}) = \begin{cases} true & if\, t \leq T \\ false & otherwise \end{cases}$, is also proper. The resulting MLS consists of segmenting when the number of events in the left segment $\omega_{t)} \in \Omega_G \cap \Omega'_G$ reaches N or segmenting when the length of the left segment $\omega_{t)} \in \Omega_G \cap \Omega'_G$ reaches T, whichever occurs first.

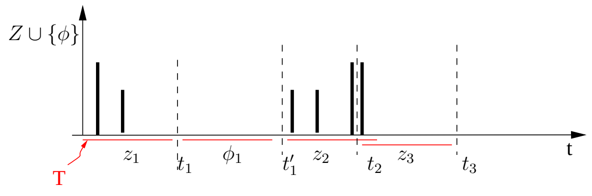


Figure 4. Intersection of segmentation by counting events such that $N \leq 3$ and segmentation by measuring time intervals such that $l(\omega) \leq T$.

However, the intersection of two proper generated segment sets is not always proper as shown in following example:

Example 21. Intersection of two proper generated segment sets that is not proper

The *intersection* of two proper generated segment sets, $\Omega_G^+ \cap \Omega'^+_G$, obtained by specific MLS decomposition of segments having a number of events greater than or equal to one, with condition

$c(\omega_{t>}) = \begin{cases} true & if\, n_E \geq 1 \\ false & otherwise \end{cases}$,

and by specific MLS decompositions of Example 17, with condition

$c'(\omega_{t>}) = \begin{cases} true & if\, t \leq T \\ false & otherwise \end{cases}$, is not proper.

Following the arguments of Example 18 for segments of length less than T, it can be seen that the last right segment ω_n could not be segmented into $\omega^n_{t)} \bullet \omega^n_{[t}$ (both of length less than T) because right segment $\omega^n_{[t}$ could have no events leading second condition *c'* to be false. Notice, that replacing first condition *c* by fixed length condition (and corresponding generator set $\Omega_G^{\overline{T}}$), the intersection is then proper (each segment being of fixed length they are not concatenated together whereas satisfying the intersection condition).

Hence, not all combinations of generated segment sets are necessarily proper, as it is also the case for the union of generated segment sets.

Remark 22. *The union of two proper subsets,* Ω_G^+ and Ω'^+_G, *is not necessarily proper* even if the two subsets are proper. Applying MLS to the union of the two generated segment sets Ω_G^+ and Ω'^+_G, MLS will stop at which ever condition occurs last, i.e., $t^* = max\{t\}$ such that $c_1(\omega_{t)}) \vee c_2(\omega_{t)}) = max\{t_1, t_2\}$. Let us consider a set of segment generators as $\Omega_G = \{\omega, \omega'\}$ with $\omega(t) = \begin{cases} 1 & for\ t = 0 \\ \phi & elsewhere \end{cases}$ defined over $[0, t_f]$ and $\omega'(t) = \begin{cases} 1 & for\ t = t'_f \\ \phi & elsewhere \end{cases}$ defined over $[0, t'_f]$. Concatenating the second generator at the end of the first extends the first except that it leaves the end point without a generator (e.g., the remainder of a concatenation $\omega \bullet \omega'$ is just a single event point).

operation	proper generated sets	proper generator sets
intersection	not nec. proper	proper
union	not nec. proper	not nec. proper
disj. union	at least weakly proper	not nec. proper

Table I
PROPRIETY OF OPERATIONS ON GENERATED AND GENERATOR SEGMENT SETS.

Let us see finally a case of disjoint union of two proper sets:

Example 23. Disjoint union of two proper generated segment sets is weakly proper

Let us consider segmentation condition

$c(\omega_{t)}) = \begin{cases} true & if\, n_E \geq N \wedge t \leq T \\ false & otherwise \end{cases}$, with $N \geq 1$,

and segmentation condition

$c'(\omega_{t)}) = \begin{cases} true & if\, n_E = 0 \wedge t > T \\ false & otherwise \end{cases}$.

The intersection of both corresponding generator segments Ω_G and Ω'_G is empty. Then, each generated segment set Ω_G^+ or Ω'^+_G, being proper, their *disjoint union* $\Omega_G^+ \amalg \Omega'^+_G$, is *weakly proper*. Indeed, condition *c* guarantees closure under left segmentation whereas condition *c'* does not garentee closure under left *and* right segmentations (for some $t \in dom(\omega)$ of segments satisfying second condition, left and right segments do not satisfy anymore this condition). Finally, the generated set are segments with a number of events greater or equal to N within interval T separated by longer nonevent intervals. At the operational level, for any segmentation, only one condition can be true.

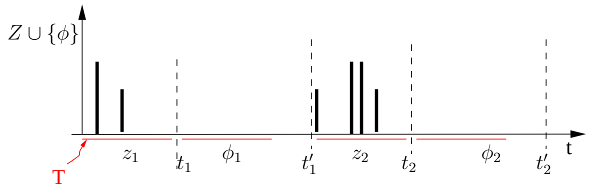


Figure 5. Disjoint union of MLS segmentations with $N = 2$.

Table I sums up previous examples and also adds some information that we will show/discuss immediately.

Concerning the operations on generated segment sets, they can be summarized as follows:

- Intersection: $\Omega_G^+ \cap \Omega'^+_G = \{\omega \,|\, (c(\omega_{t)}) \wedge c'(\omega_{t')})) = true, l(\omega) = min\{t, t'\}, \omega_{t)} \in \Omega_G, \omega_{t')} \in \Omega'_G\}$, with $(c(\omega_{t)}) \wedge c'(\omega_{t')})) = true$ meaning that both conditions *c* and *c'* are true. Example 21 shows that $\Omega_G^+ \cap \Omega'^+_G$ is not necessarily proper: $\Omega_G^{T+} \cap \Omega_G^{N_{sup}+}$, with $\Omega_G^{N_{sup}} = \{\omega \,|\, n_E > 0\}$.
- Union: $\Omega_G^+ \cup \Omega'^+_G = \{\omega \,|\, (c(\omega_{t)}) \vee c'(\omega_{t')})) = true, l(\omega) = max\{t, t'\}, \omega_{t)} \in \Omega_G, \omega_{t')} \in \Omega'_G\}$, with $(c(\omega_{t)}) \vee c'(\omega_{t')})) = true$ meaning that one or both conditions *c* and *c'* are true. Remark 22 shows that $\Omega_G^+ \cup \Omega'^+_G$ is not necessarily proper.

- Disjoint union: $\Omega_G^+ \amalg \Omega_G'^+ = \{\omega \mid (c(\omega_{t)}) \veebar c'(\omega_{t')})) = true, \omega_{t)} \in \Omega_G, \omega_{t')} \in \Omega_G'\}$, with $l(\omega) = \begin{cases} t & if \quad c(\omega_{t)}) = true \\ t' & otherwise \end{cases}$ and $(c(\omega_{t)}) \veebar c'(\omega_{t')})) = true$ meaning that only one condition *c* or *c'* is true. Example 23 shows that $\Omega_G^+ \amalg \Omega_G'^+$ can be weakly proper (we will show after that it is *always* at least weakly proper).

Concerning the operations on generator segment sets, they can be summarized as follows:

- $\Omega_G \cap \Omega_G'$ (as defined for generated sets) is proper, the latter intersection of the two generator segments being a strong requirement determining the minimum size segment being also proper.
- $\Omega_G \cup \Omega_G'$ (as defined for generated sets) can be improper, following Remark 22, it can be seen that the union of the two segments is improper.
- $\Omega_G \amalg \Omega_G'$ (as defined for generated sets) can be improper, following also Remark 22, it can be seen that the disjoint union of the two segments is improper.

Let us see that the union of two generated segment sets Ω_G^+ and $\Omega_G'^+$ is always at least weakly proper by following theorem:

Theorem 24. The disjoint union of two proper generated segment sets, $\Omega_G^+ \amalg \Omega_G'^+$, is *at least weakly proper* .

Proof: Let us consider the set of segment generators $\Omega_G = \{\omega \mid c(\omega) = true\}$, corresponding to the first specific MLS and $\Omega_G' = \{\omega' \mid c'(\omega') = true\}$ corresponding to the second specific MLS. Remember now that an MLS should find the longest generator (of length $l(\omega) = t^*$) in a generated segment set Ω_G^+. Applying MLS to the disjoint union of both generated segments, only one condition can be true, i.e, $t^* = \begin{cases} t & if \quad c(\omega_{t)}) = true \\ t' & otherwise \end{cases}$. Each generated set being proper, accounting for Example 23, their disjoint union is then at least weakly proper. ■

IV. System specification

Conventional discrete-event models of neurons do not rely on a mathematical framework that allows precise and safe (mathematically speaking) modeling of the dynamics of a neuron. This section first briefly introduces a biological neuron and the usual trajectories (and possible segmentation) of the neuron's potentials that can be mapped to an input-output system. A corresponding informal description of a discrete-event model of a leaky integrate and fire neuron is provided. After that, two iterative specifications are derived from these models: one at spike level (cf. Definition 26), the other one at burst level (cf. Definition 37 using bursty segments Definition 29). Each specification proves to specify a well-defined general system (cf. Theorems 27 and 38). While spiky neurons operate at the basic level of single spikes (or discrete-events), bursty neurons allow aggregating these spikes into packets of spikes, using previous given generator-based definitions.

A. Generating systems

Based on previous propriety definitions and examples, Theorem 7 can be generalized as follows:

Theorem 25. *Sufficient conditions for iterative specification of a general system,* $(\delta, \lambda) \overset{(+,\lambda)}{\rightarrow} (\Delta, \lambda)$*:*

1) *The set of generated input segments* Ω_G^+ *is proper (cf. Definitions 9 to 11),*
2) Consistency of composition*:* $\omega_1, \omega_2, \ldots, \omega_n \in \Omega_G$ *and* $\omega_1, \omega_2, \ldots, \omega_n \in \Omega_G^+$.

Proof: Replacing condition 1) by strong propriety definition 11 is equivalent to Theorem 7. Strong propriety can be weakened as shown at the end of subsection II-C. Weak and strong propriety can be generalized as simple propriety as shown in Example 18 by induction. ■

B. Neurons as dynamic systems

Figure 6 describes the continuous potential propagation through the dendrite (input), the soma (locus of potential integration) and the axon (output) of a neuron. The latter can be described as an input-output system with dendrite PostSynaptic Potential (PSP) as input, soma membrane potential as state and axon potential as output. Segmentation is represented by vertical lines.

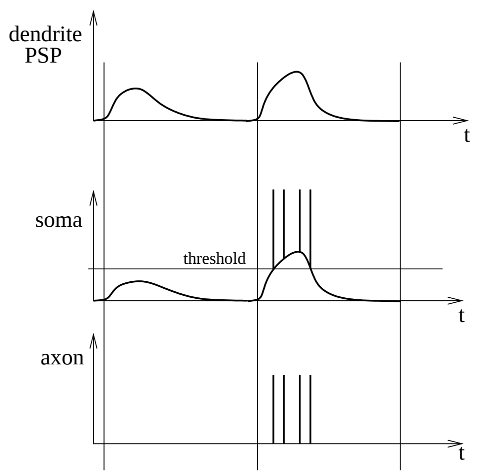


Figure 6. Continuous potential propagation through the dendrite, the soma and the axon of a neuron.

The use of *dynamical systems* in neurosciences [15] concerns mainly continuous generations of spikes [16], [17], [18], [19]. The iterative specification of systems aims at being more general applying to any discrete or continuous system.

C. Discrete-event model of a leaky integrate and fire neuron

Discrete-event spiking neurons have been widely implemented in several software environments [20]. From the neuronal nets perspective, the Discrete Event System Specification (DEVS) formalism has been used mainly for proposals of novel neuron models [21], the specification of dynamic structure neurons [22], the abstraction of neural nets [23] and for the specification of continuous spike models [19], [18]. In addition to DEVS representations, discrete events have been used successfully in neuronal nets for modeling [24], [20] and simulation [25], [26], [20], [27].

The goal of the discrete-event model hereafter is to propose a model, as simple as possible, capturing the essence of discrete-event characteristics in a leaky integrate and fire (LIF) neuron application. Spikes here do not model explicitly potential variations rather they are abstracted in event segments. A LIF neuron consists of a memory-based model storing the potentials previously received in the membrane (especially here in the soma, thus simply refering to the "soma membrane potential" as the "membrane potential" hereafter). An interesting aspect is that the remaining potentials decrease in time due to leakage leading to nonlinearities in neural behaviors. Corresponding simple discrete-event models discussed here provide a basis that can be easily extended to deal with further details (such as multiple inputs, synaptic weights, etc.). Figure 7 depicts the dynamics of a basic discrete-event model of a leaky integrate and fire neuron.

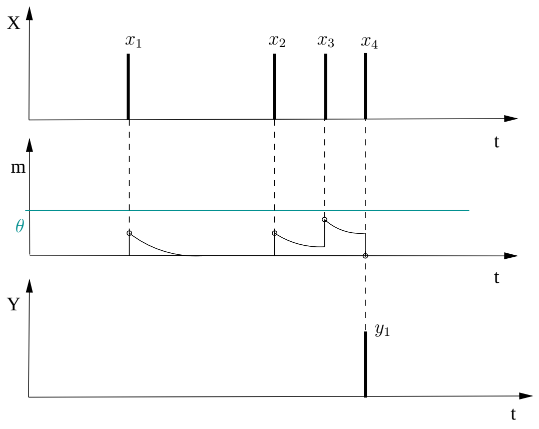


Figure 7. Discrete-event model of a leaky integrate and fire neuron.

A discrete-event input is a couple (x_i, t_i), with $x_i = 1$ and $t_i \in \mathbb{R}^+_\infty$ respectively the value and the time stamp of the discrete-event. Discrete-event inputs occur in a sequence $(x_0, t_0), (x_1, t_1), \ldots, (x_i, t_i), \ldots, (x_n, t_n)$. The same characterization holds for each discrete-event output (y_i, t_i).

Based on the current value of membrane potential m, the new value m' consists of:

$$m' = \begin{cases} r^e m + 1 & if\, m < \theta \\ 0 & otherwise \end{cases} \tag{1}$$

with $e \in \mathbb{R}^+_\infty$ the *elapsed time since the last transition*, *initial membrane potential* is $m = 0$, *new membrane potential* m' depending on: *remaining potential* $r^e m$, with $r \in [0, 1]$ the *remaining coefficient*

[1], and $\theta \in \mathbb{R}^+$ the *firing threshold.*

Discrete-event neuron models consider only membrane potential transitions when receiving an external event (an input spike) or when having scheduled an internal event (the membrane potential going to zero because of the potential leak). Also, notice that both change of potential membrane value and firing are immediate (without the commonly employed "artificial" discrete-time delay of one unit). For membrane potential, this means that membrane potential is not updated (and considered to be constant) between two discrete-events.

Spike value emission y depends on *threshold* $\theta \in \mathbb{R}^+$:

$$y = \begin{cases} 1 & if\, m \geq \theta \\ \phi & otherwise \end{cases} \tag{2}$$

D. Iterative specification at spike level

The previous discrete-event model provides an introduction to the discrete-event dynamics of a LIF neuron. The model is considered here from an iterative system specification perspective, ensuring the global system properties as defined in Section II. Figure 8 depicts the segment-based description. Notice that due to exponential decay, a spiky segment lasts infinitely until a next spiky segment.

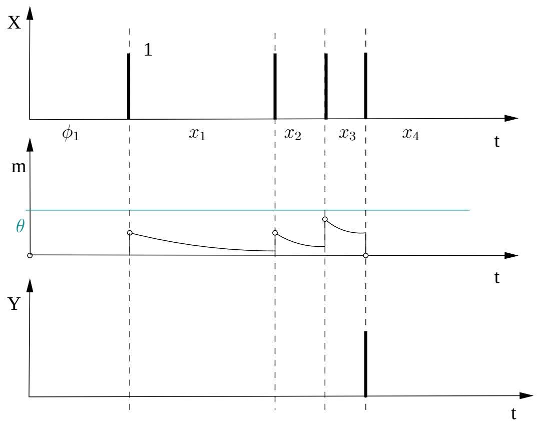


Figure 8. Discrete-event model of a leaky integrate and fire neuron. Notice that state computations occur only at the beginning and at the end of segments.

Definition 26. The translation of a LIF model M_S into a *spiky iterative specification* $G(M_S)$ of a system consists of:

$$G(M_S) = (\delta_S, \lambda_S)$$

where $dom(\omega_{X_S}) = [0, t] \cap \mathbb{R}$, $X_S = Y_S = \{1, \phi\}$, $Q_S = \{m\}$, $\Omega^S_G = \{x_i \,|\, x_i(0) = \{0, 1\} \wedge x_i(t) = \phi \, for\, t \in [0, t_i)\}$ is the *set of spiky input segment generators* (null value input events are used for initialization as it will be seen), both *single segment state transition* δ_S and *output function* λ_S are defined in following theorem.

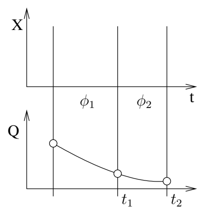
Now that a LIF neuron has been iteratively specified, the question is whether this specification leads to a well-defined general system, i.e., does the specification satisfy the requirements exposed in Theorem 7?

Theorem 27. *A spiky iterative specification* $G(M_S) = (\delta_S, \lambda_S)$ *can be associated to a system* $S_{G(M_S)} = (\Delta_S, \lambda_S)$ *through a concatenation process* $COPRO_{G(M_S)} = (\delta^+_S, \lambda_S)$.

Proof: Sufficient conditions for iterative specification (cf. Theorem 25) can be checked as:

[1]When *remaining coefficient* value $r = 1$, there is *no leak*, all the potential received remains in the membrane. When $r = 0$, all the potential received at time $t - 1$ is lost (the model is then equivalent to McCulloch & Pitts' model).

1) Considering spiky input segments as a subset of all discrete-event segments, i.e., $\Omega_G^S \subset \Omega_G^E$, where each input segment starts with initial value zero or one, the set of generated input segments Ω_G^{S+} is *strongly proper* being based on condition $c(\omega_{t)}) = \begin{cases} true & if\, n_E \leq 1 \\ false & otherwise \end{cases}$, for left segment $\omega_{t)} = x_i(t) = \begin{cases} 1 & for\ t=0 \\ \phi & elsewhere \end{cases}$ or $\omega_{t)} = x_i(t) = \begin{cases} 0 & for\ t=0 \\ \phi & elsewhere \end{cases}$, leading to the set of input segment generators $\Omega_G^S = \{x_i \,|\, x_i(0) = \{0,1\} \wedge x_i(t) = \phi\ elsewhere\}$.
2) *Consistency of composition*: The *single segment state transition function* for a spiky input segment $x_i \in \Omega_G^S$ is defined as $\delta_S(m, x_i) = \begin{cases} 0 & if\, m + x_i \geq \theta \\ (m+x_i).r^{t_i} & otherwise \end{cases}$, where the top part of the equation corresponds to a spike emission and the bottom part corresponds to an update of membrane potential with input spike addition. The output function consists of $\lambda_S(m) = \begin{cases} 1 & if\, m+1 \geq \theta \\ \phi_i & otherwise \end{cases}$. Now consider that each spiky generator $x_i \in \Omega_G^S$ can be decomposed into basic discrete event generators (as defined in Definition 15): (i) an event input generator $z_i \in \Omega_G^Z$ such that $\Omega_G^Z = \{z_i \,|\, z_i(0) = \{0,1\}\}$, and (ii) a *nonevent segment generator* $\phi_i \in \Omega_\phi$ *such that* $\Omega_G^\phi = \{\phi_i \,|\, \phi_i : [0, t_i) \rightarrow \{\phi\}\}$. It can be shown that these atomic generators can be *concatenated* to define recursively the single segment state transition function δ_S of the spiky iterative specification $G(M_S) = (\delta_S, \lambda)$, and then the extended state transition function δ_S^+ of the concatenation process $COPRO_{G(M_S)} = (\delta_S^+, \lambda)$:
   a) For a nonevent input generated segment $\omega \in \Omega_S^+$, e.g.:



   considering an exponential remaining coefficient $r^t = exp(-\alpha t)$, with α a constant, nonevent input segment generators, $\phi_1, \phi_2 \in \Omega_G^E$ can be concatenated while membrane potential computation remains true, i.e., $\delta_S^+(m, \phi_1 \bullet \phi_2) = \delta_S(\delta_S(m, \phi_1), \phi_2)$ or through membrane potential $m' = \delta_S^+(m, exp(-\alpha(t_1 + t_2))) = \delta_S(\delta_S(m, exp(-\alpha t_1)), exp(-\alpha t_2))$.
   b) For each spiky segment $x_i \in \Omega_G^S$, $\delta_S^+(m, x_1 \bullet x_2) = \delta_S(\delta_S(m, x_1), x_2)$, as the single segment state transition function δ_S is recursively defined for a spiky input event segment as $\delta_S^+(m, x_i) = \begin{cases} \delta_S(0, \phi_i) & if\, m + z_i \geq \theta \\ \delta_S(m + z_i, \phi_i) & otherwise \end{cases}$, where the top part of the equation corresponds to a spike emission and the bottom part corresponds to an update of membrane potential with input spike addition.
   c) Finally, the output function consists of $\lambda_S(m) = \begin{cases} 1 & if\, m \geq \theta \\ \phi_i & otherwise \end{cases}$.

■

In conclusion, this theorem shows that basic atomic discrete-event generators can be concatenated in a compact iterative specification of spiky neurons, $G(M_S) = (\delta_S, \lambda_S)$. Here the input segment generators $\Omega_G^S = \{x_i \,|\, x_i(0) = \{0,1\} \wedge x_i(t) = \phi\ elsewhere\}$ are composed of only two types of input segment generators (the ones starting with input value 0 and the ones starting with input value 1, both being null elsewhere.) Moreover, the dynamics are provided by the single segment state transition function $\delta_S(m, x_i) = \begin{cases} 0 & if\, m + x_i \geq \theta \\ (m+x_i).r^{t_i} & otherwise \end{cases}$ and output function $\lambda_S(m) = \begin{cases} 1 & if\, m+1 \geq \theta \\ \phi_i & otherwise \end{cases}$.

E. Iterative specification at burst level

Neurons can exchange spikes in bursts. Figure 9 describes an experiment that caused a single neuron to generate spikes in burts. More precisely, bursts of spikes can be defined as follows.

Definition 28. [Modified from [28]] A "*burst firing* (...) [consists] of trains of two or more spikes occurring within a relatively short interval and followed by a [longer] period of inactivity" (cf. Figure 9).

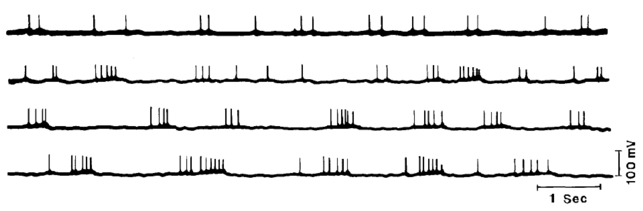


Figure 9. Burst firing: "Effect of intracellular injection on the firing pattern of nigral dopamine (DA) cells. In the first few minutes following impalement with a calcium-containing electrode, the stabilized DA cell demonstrates its typical slow, single spike firing pattern (top trace). As calcium leaks from the electrode into the cell, the pattern slowly changes over the next 10 to 20min into a burst-firing pattern (second through fourth trace)" (from [28]).

Based on bursts and on the previous definition of basic discrete-event generators (cf. Definition 15), we aim to define *bursty LIF neurons* as having the usual LIF properties while being able to exchange, through input-output interfaces, and to compute, bursty segments. A bursty segment consists of packet segments followed by longer null segments. Until now only *packets* (and their duration) have been used formally. Before being able to segment bursty segments we need to precisely define what is a bursty segment in order to be able to identify each subsegment.

Definition 29. Bursty segments.

Following Definition 28, a bursty segment $\omega \in \Omega_B$, can be segmented alternatively into an active segment (with events) followed by an inactive segment (with no events), followed by an active segment,..., leading to a generated segment $\omega^+ \in \Omega_G^{B+}$ such that $\omega^+ = P_1 \bullet \phi_1 \bullet P_2 \bullet \phi_2 \bullet ...$, where ϕ_i *are nonevent segments* and $P_j = x_{j1} \bullet \ldots$ *are packet segments*: a concatenation of *spicky segments* $x_{jk} \in \Omega_G^S$. Bursts follow also two *requirements*: (i) each packet duration $l(P_j)$ is less than a length T_p, i.e., $l(P_j) \leq T_p$, and (ii) each inactivity duration $l(\phi_i)$ is greater than a packet duration, i.e., $l(\phi_i) > T_p$. Notice that active segments are called *packets* to reflect the discrete nature of spikes.

Figure 10 presents an example of bursty segmentation in a bursty neuron. An application of this algorithm is presented in Figure 10. The algorithm starts the segmention either with a burst segment or an empty segment. If it is a packet segment, the algorithm will concatenate all spicky segments until a last spike. If it is an empty segment, the algorithm will search for the first next spike. We will see in details how the segmentation can be properly achieved.

Lemma 30. The set of generated nonevent segment $\Omega_G^{\phi+}$ is proper and based on generators $\Omega_G^{\phi} = \{\omega | \, c_\phi(\omega) = true\}$, with $c_\phi(\omega) = \left\{ \begin{array}{ll} true & if \, n_E = 0 \\ false & otherwise \end{array} \right.$, indeed $\Omega_G^{\phi+} = \Omega_G^{\phi}$.

Proof: The concatenation of two nonevent segments is another nonevent segment. The left and right segments of a null event segment are also a null event segments. Therefore the set of nonevent generated segments, $\Omega_G^{\phi+}$, is *strongly proper*. Then the set of nonevent segment generators Ω_G^{ϕ} is closed under concatenation. This means that the concatenation of a finite number of non event segments is also a non event segment. Therefore the MLS always results in termination with the input segment $\omega \in \Omega$ as the result. ■

Now that nonevent segments can be detected and concatenated, let us detect packets. To do so, the density of segments is defined. This density can be *low* (being lower than a threshold D) or *high* (being above a threshold D). More precisely:

Definition 31. *Density* of a segment $\omega \in \Omega$ consists of $Density(\omega) = \frac{n_E(\omega)}{l(\omega)}$ with $n_E(\omega)$ as defined in Example 16. *Generator segment set* $\Omega_G^{>D} = \{\omega | \, c_{>D}(\omega) = true\}$, with $c_{>D}(\omega) = \left\{ \begin{array}{l} true \quad if \, Density(\omega) > D \\ false \quad otherwise \end{array} \right.$, is *lower bounded*. *Generator segment set* $\Omega_G^{<D} = \{\omega | \, c_{<D}(\omega) = true\}$, with $c_{<D}(\omega) = \left\{ \begin{array}{l} true \quad if \, Density(\omega) < D \\ false \quad otherwise \end{array} \right.$, is *upper bounded*. Generator segment set Ω_D is said to be *density bounded* if it is either upper or lower bounded.

As for the set of nonevent segment generators, the set of density bounded generators is closed under concatenation.

Remark 32. Density bounded generator sets are *closed under concatenation*. This can be easily shown considering all pairs of segment generators $\omega_1, \omega_2 \in \Omega_G^D$, such that $n_E(\omega_1) < (or \;\; >) \, Dl(\omega_1))$ and $n_E(\omega_2) < \;(or \;\; >) \, Dl(\omega_2)$, leading to $n_E(\omega_1) + n_E\,(\omega_2) \; < \; (or \;\; >) \, Dl(\omega_1) + Dl(\omega_2)$. Density bounded generator sets are closed under concatenation.

The set of nonevent segment generators, Ω_G^{ϕ}, was strongly proper. Let us see now the properness of the set of density bounded generators Ω_G^D.

Lemma 33. The set of generated density bounded segments Ω_G^{D+} is *weakly proper*, indeed $\Omega_G^{D+} = \Omega_G^D$.

Proof: By closure under concatenation, the concatenation of a finite number of density bounded segments is also density bounded. Therefore the MLS always results in termination with the input segment as the result. Closure under left and right segmentation is not necessarily true (depending on the uniformity of spike distribution). ■

Having shown that $\Omega_G^{D+} = \Omega_G^D$ and $\Omega_G^{\phi+} = \Omega_G^{\phi}$, both sets of concatenated generators are finite and thus respects MLS definition. We will employ lower bounded density bounded generators as the obvious means to represent bursts which are groups of spikes with greater density than their surrounding. Now, the question remains concerning MLS' distinction between both non event and packet generators. This requires adding a new condition:

Remark 34. In the following proposition we also require that such generators start and end with a spike to facilitate boundary detection in the MLS process.

More formally, let $\widetilde{\Omega_G^{>D}} = \Omega_G^{>D} \cap \Omega_G^{LIM}$ be the *set of packet generators* with $\Omega_G^{LIM} = \{\omega \,|\, c_{LIM}(\omega) = true\}$ and $c_{LIM}(\omega) = \left\{ \begin{array}{l} true \quad if \, \omega(0) = 1 \wedge \omega(l(\omega)) = 1 \\ false \quad otherwise \end{array} \right.$, then

Proposition 35. $\widetilde{\Omega_G^{>D}} \bigcup \Omega_G^{\phi}$ is a *weakly proper* generator set.

Proof: In view of the Lemmas 30 and 33, it is enough to consider heterogeneous segments of the form, $\omega = \phi_1\omega_1 \bullet \phi_2\omega_2 \bullet ... \bullet \phi_n \bullet \omega_n$. MLS will produce a left segment that ends at the first spike of the first density-bounded segment. The remaining right segment will then have the density-bounded segment as its MLS left segment. The subsequent remaining segment is of the same form subject to repeated MLS. The process terminates because the remaining segment always gets shorter. Note that the requirement for boundary spikes greatly simplifies the MLS segmentation. Without this requirement MLS can stop in the middle of a segment generator, between two spikes. ■

Notice that Ω_G^{LIM} is consistently *weakly proper* (starting with spike is closed under left-segmentation, similarly ending is...).

The proposition gives us the means to generate and uniquely decompose segments containing high density subsegments. However it does not assure that high density segments are separated by long null event segments. One solution to do so is to synthesize generators that have encapsulated high density segments and nonevent segments of equal lengths. Let $\Omega_G^B = \Omega_G^{\phi}\widetilde{\Omega_G^{>D}} = \{\omega \,|\, \omega = \phi \bullet \omega', \phi \in \Omega_G^{\phi}, \omega' \in \widetilde{\Omega_G^{>D}}, \, c_L(\phi, \omega') = true\}$ be the set of bursty generators, with $c_L(\phi,\omega') = \begin{cases} true & if\, l(\phi) = l(\omega') \\ false & otherwise \end{cases}$, a generator set of paired concatenated nonevent and spike-enclosed density lower-bounded segments (or *packets*).

Notice that $\omega = \phi \bullet \omega'$ with is a higher-level segment generator composed of two concatenated segment generators. Also having both nonevent and packet segments of equal length has been set for sacke of simplicity. This assumption can be easily extended using both length conditions $c_L(\phi) = \begin{cases} true & if\, l(\phi) > T_p \\ false & otherwise \end{cases}$ and $c_{L'}(\omega') = \begin{cases} true & if\, l(\omega') \leq T_p \\ false & otherwise \end{cases}$ with T_p an arbitrary packet length.

Lemma 36. Ω_G^B is a *weakly proper* generator set.

Proof: The MLS process will detect the first nonevent segment and continue to the following density-bounded segment, stopping at its end, which marks the end of the first paired nonevent-density-bounded segment. The remaining right segment, if not a nonevent, is of the same form as the original (thus leading to a weak proper set). The process terminates because remaining segment always gets shorter. ■

With all the previous apparatus, it is now easy to detail the structure of a bursty neuron model as

Definition 37. The translation of a bursty model M_B into an *iterative specification* $G(M_B)$ of a system consists of:

$$G(M_B) = (\delta_B, \lambda_B)$$

where $dom(\omega_{X_B})$, X_B, Y_B, Q_B are defined as for the spike model, Ω_B^G is the *set of input segment bursty generators* defined with both *single segment state transition* δ_B and *output function* λ_B in following theorem.

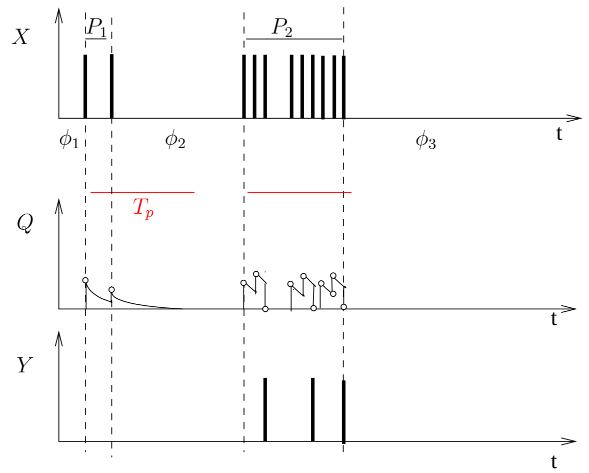


Figure 10. Bursty segmentation at input, state and output levels.

Again we need to prove now that the bursty specification leads to a well-defined system.

Theorem 38. *A bursty iterative specification* $G(M_B) = (\delta, \lambda)$ *can be associated to a system* $S_{G(M_B)} = (\Delta_B, \lambda_B)$ *through a concatenation process* $COPRO_{G(M_B)} = (\delta_B^+, \lambda_B)$.

Proof: Sufficient conditions for iterative specification (cf. Lemma 25) can be checked as:

1) Considering bursty input segments as a subset of all discrete event segments, i.e., $\Omega_G^B \subset \Omega_G^E$, they can be properly segmented based on Ω_G^B the *weakly proper generator segment set* as previously defined (cf. previous lemmas and proposition).
2) *Consistency of composition*: Considering a generated bursty input segment $\omega \in \Omega_G^{B+}$ as a concatenation of spicky segments such that $\delta_B{}^+(m,\omega) = \delta_B{}^+(m, x_1 \bullet x_2 \bullet ... \bullet x_n) = \delta_S^+(\delta_S^+(\ldots \delta_S^+(\delta_S^+(m,x_1),x_2),\ldots),x_n)$. As spicky transitions proved to be composable (cf. Theorem 27), i.e., $\delta_S^+(m,x_i) = \delta_S(m+z_i,\phi_i)$, the final membrane potential value is equal to $\delta_B{}^+(m,\omega) = \Sigma_{i=1}^n \delta_S^+(m,x_i)$, with generator spicky transition $\delta_S^+(m,x_i)$ described in Theorem 27. Finally, *output function* consists of $\lambda_B(m) = \begin{cases} 1 & if\, m \geq \theta \\ \phi_i & otherwise \end{cases}$.

■

V. Conclusion and perspectives

A new compact reformulation of iterative specification of systems has been proposed. Iterative system specification structure has been considered here as the main structure for the computational modeling of dynamic systems with memory. Iterative system specification as defined is abstract enough to be applicable to most (if not all) dynamic computational models. To establish this approach, an approach called Maximal Length Segmentation has been developed for discrete-event segmentats. Defining new basic discrete-event generators, specific segmentations (depending on segment metrics conditions, e.g., state and time changes) can be defined and combined in a modular way. That is, conditions of specific MLS can be logically combined. In future work we will show that at the network level as well, structure-conditions can be combined by iteratively employing the network components' MLS. This is consistent with the iterative definition at the basic dynamic level of components.

Using this extended computational modeling framework, spiky neuronal models have been formally mapped to iterative system specification structures using discrete-event segments. Bursty MLS has been obtained by the combination of three specific MLS processes (based on event counting, time intervals and non event detection). Corresponding to the bursty MLS, bursty neurons have been defined as exchanging bursty generator segments. At the input/output relation level of system specification, packets encapsulating high density spiky intervals and nonevent periods alternate. This form of input segment can generate the same form of output segment. This analysis is facilitated by the iterative specification approach.

Applicability of the iterative specification framework to other dynamic systems is illustrated in the potential to formalize spike coding by neurons [29]. This can provide new insights by considering neurons as analog-to-digital converters.

Indeed, encoding continuous-time signals into spikes using a form of sigma-delta modulation would fit the DEVS iterative specification framework which accommodates both continuous and discrete event segments. Future research could seek to characterize properties of I/O functions that map continuous segments to discrete event segments. Another application domain is that event-based message transmission for time-varying non-linear systems in sensor networks subject to saturation effects [30].

This approach also provides a formal system specification hierarchy basis for probabililstic state changes and continuous-time Markov models [31]. Relying on both system specification and density segmentation provided here, we aim at modeling probabilistic populations of bursty neurons with respect to their structure/behavior formalizing the criteria for correct realization using system morphisms [2]. In addition application to challenging areas such as evolutionary game theory [32], [33] could prove fruitful since analytic solutions are extremely difficult to obtain and formalized simulation as presented here should provide a more sound way forward. Finally, future work will study the composition of bursty neurons into networks where the possibility of exchanging bursts as unitary generators will be formally characterized.

Acknowledgements

This material is based in part upon work supported by the Centre National de la Recherche Scientifique.

References

[1] B.P. Zeigler. *Theory of Modeling and Simulation*. Wiley, 1976.

[2] B.P. Zeigler, T.G. Kim, and H. Praehofer. *Theory of Modeling and Simulation*. Academic Press, 2000.

[3] Michael A Arbib. *Theories of abstract automata*. 1972.

[4] George J Klir. *Architecture of systems complexity*. Saunders, New York, 1985.

[5] M.D. Mesarovic and Y. Takahara. *Abstract Systems Theory*. LNCIS 116. Springer.

[6] M.D. Mesarovic and Y. Takahara. *General Systems Theory: Mathematical Foundations*. LNCIS 116. Academic Press.

[7] Wayne Wymore. *A mathematical theory of systems engineering*. Wiley, 1967.

[8] André Arnold. Finite transition systems. international series in computer science, 1994.

[9] Michael A Harrison. Lectures on linear sequential machines. Technical report, 1969.

[10] Yu-Chi Ho. *Discrete event dynamic systems: analyzing complexity and performance in the modern world*. IEEE, 1992.

[11] Alexandre Muzy and Bernard P. Zeigler. Activity-based Credit Assignment (ACA) Heuristic for Simulation-based Stochastic Search in a Hierarchical Model-base of Systems. *IEEE Systems Journal*, PP(99):1–12, 2014.

[12] Ryo Sato. Realization theory of discrete-event systems and its application to the uniqueness and universality of devs formalism. *International journal of general system*, 30(5):513–549, 2001.

[13] Takao Asahi and BP Zeigler. Behavioral characterization of discrete event systems. In *AI, Simulation, and Planning in High Autonomy Systems, 1993. Integrating Virtual Reality and Model-Based Environments. Proceedings. Fourth Annual Conference*, pages 127–132. IEEE, 1993.

[14] Hessam Sarjoughian. *Inductive Modeling of Discrete-event Systems: A TMS-based Non-monotonic Reasoning Approach*. PhD thesis, Tucson, AZ, USA, 1995. UMI Order No. GAX95-34670.

[15] Eugene M Izhikevich. *Dynamical systems in neuroscience*. MIT press, 2007.

[16] Eugene M Izhikevich et al. Simple model of spiking neurons. *IEEE Transactions on neural networks*, 14(6):1569–1572, 2003.

[17] Wolfgang Maass. Networks of spiking neurons: the third generation of neural network models. *Neural networks*, 10(9):1659–1671, 1997.

[18] Guillermo L Grinblat, Hernán Ahumada, and Ernesto Kofman. Quantized state simulation of spiking neural networks. *Simulation*, 88(3):299–313, 2012.

[19] Rene Mayrhofer, Michael Affenzeller, Herbert Prähofer, Gerhard Höfer, Alexander Fried, and Er Fried. Devs simulation of spiking neural networks. In *Cybernetics and Systems: Proceedings EMCSR 2002*, volume 2, pages 573–578, 2002.

[20] Romain Brette. Simulation of networks of spiking neurons: A review of tools and strategies. *Journal of Computational Neuroscience*, 23(3):349–398, 2007.

[21] B.P. Zeigler. Discrete event abstraction: an emerging paradigm for modeling complex adaptative system. *Adaptation and evolution (festschrift for John H. Holland)*, 2005.

[22] S. Vahie. *Discrete Event Modeling and Simulation Technologies: A Tapestry of Systems and AI-Based Theories and Methodologies*, chapter Dynamic Neuronal Ensembles: Neurobiologically Inspired Discrete Event Neural Networks. Springer-Verlag, 2001.

[23] Bernard P. Zeigler. Statistical simplification of neural nets. *International Journal of Man-Machine Studies*, 7(3):371–393, 1975.

[24] Arnaud Tonnelier, Hana Belmabrouk, and Dominique Martinez. Event-driven simulations of nonlinear integrate-and-fire neurons. *Neural Computation*, 19(12):3226–3238, 2007.

[25] M.L. Hines and N.T. Carnevale. Discrete event simulation in the NEURON environment. *Neurocomputing*, 58-60(0):1117–1122, 2004.

[26] YuHua Tang, BaiDa Zhang, JunJie Wu, TianJiang Hu, Jing Zhou, and FuDong Liu. Parallel architecture and optimization for discrete-event simulation of spike neural networks. *Science China Technological Sciences*, 56(2):509–517, 2013.

[27] Anthony Mouraud, Didier Puzenat, and Hélène Paugam-Moisy. DAMNED: A Distributed and Multithreaded Neural Event-Driven simulation framework. *Computing Research Repository*, abs/cs/051, 2005.

[28] Anthony Grace and Benjamin Bunney. The control of firing pattern in nigral dopamine neurons: burst firing. *The Journal of neuroscience*, 4(11):2877–2890, 1984.

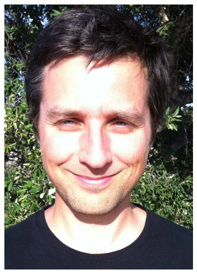
[29] Y. C. Yoon. Lif and simplified srm neurons encode signals into spikes via a form of asynchronous pulse sigma-delta modulation. *IEEE Transactions on Neural Networks and Learning Systems*, PP(99):1–14, 2016.

[30] Guoliang Wei, Shuai Liu, Licheng Wang, and Yongxiong Wang. Event-based distributed set-membership filtering for a class of time-varying non-linear systems over sensor networks with saturation effects. *International Journal of General Systems*, 45(5):532–547, 2016.

[31] Bernard P. Zeigler and Alexandre Muzy. Temporal modeling of neural net input/output behaviors: The case of xor. *Systems*, 5(1), 2017.

[32] Chengjiang Wang, Li Wang, Juan Wang, Shiwen Sun, and Chengyi Xia. Inferring the reputation enhances the cooperation in the public goods game on interdependent lattices. *Applied Mathematics and Computation*, 293:18 – 29, 2017.

[33] Chengyi Xia, Shuai Ding, Chengjiang Wang, Juan Wang, and Zengqiang Chen. Risk analysis and enhancement of cooperation yielded by the individual reputation in the spatial public goods game. *IEEE Systems Journal*, 2016.



Alexandre Muzy is chargé de recherche at CNRS. His research mainly concerns the theory of modeling and simulation at general system level with main applications in biology and neurosciences. He is associate editor of Simulation journal and member of many scientific committees of international conferences.

Bernard P. Zeigler is Emeritus Professor at Arizona Center of Integrative Modeling and Simulation and Chief Scientist at RTSync Corp, Arizona and Maryland. He is internationally known for his seminal contributions in modeling and simulation theory.