System specification

Hans Vangheluwe

When studying existing systems, *observations* (of structure and behaviour) are the only tangible artifacts we have at our disposal [Kli85]. A modeller may, based on observations and/or insight, build progressively more complex models of a system. Here, we present a hierarchy of abstract model structures. Each structure elaborates on the previous one, introducing (and representing) more detailed knowledge about the system. The reverse operation, going from a model containing more information to a less detailed one, must be shown to be possible. This, as some questions about the behaviour and structure of the system are better answered at lower levels in the hierarchy. In particular, explicit behaviour in the form of input and output trajectories, described at the lowest level, is often required.

In object-oriented terminology, a simulation *model* consists of model *objects* (often used to represent real-world objects, entities, or concepts) as well as *relationships* among those objects. In general, a model object is anything that can be characterized by one or more *attributes* to which *values* are assigned. Attributes are either called *indicative* if they describe an aspect inherent to the object or *relational* if they relate the object to one or more other objects. The values assigned to attributes have a *type* in the programming language sense (*i.e.,* , a set of allowed values they must belong to).

Mathematical *sets* and operations defined on those sets are the starting point for abstract system representation or modelling. Simple finite sets of numbers $\{1, 2, \ldots, 9\}$, identifiers $\{a, b, \ldots, z\}$, as well as infinite sets such as $\mathbb{N}, \mathbb{N}^+, \mathbb{R}$, and $\mathbb{R}^+$ are typically used. Often, specific *meaning* is given to sets and their members. The set EV for example is a finite set denoting arrival and departure *events* in a queueing system

$$EV = \{ARRIVAL, DEPARTURE\}.$$

As in the *discrete event* abstraction, discussed later in greater detail, only a finite number of events are assumed to occur in a bounded time interval, the non-event symbol ϕ is introduced to denote the absence of events changing the state of the system. The event set is subsequently enriched with ϕ

$$EV^{\phi} = EV \cup \{\phi\}.$$

This demonstrates the use of basic set operations such as $\cup$. To describe multiple attributes of a system, the set product $\times$ is used

$$A \times B = \{(a, b) | a \in A, b \in B\}.$$

1 Time base and Segments

Every simulation model must have an *indexing attribute* which, at some level of abstraction will enable state transitions [Nan81]. *Time* is the most common indexing attribute. Time is special in that it inexorably progresses: the current state and behaviour of a system can only modify its future, never its past. This concept is often called *causality*: a cause must always occur before a consequence. In a simulation context, the indexing attribute is referred to as *system time*. Any set T can serve as a formalisation of time. A

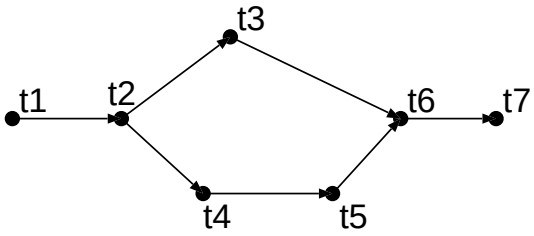

Figure 1: Partially ordered time base

nominal relationship $=$ may be added to T to denote equality. To obtain a usable *time base* however, an *order* relation on the elements of T is needed:

$$TimeBase = \langle T, < \rangle$$

This relation has properties

- transitive: $A < B \wedge B < C \Rightarrow A < C$,
- irreflexive: $A \not< A$,
- antisymmetric: $A < B \Rightarrow B \not< A$.

This formalises the notion of order in time. The ordering relationship may be *total* (linear): each element of T can be related to every other element. A *partial* ordering where not all elements of T can be compared is useful in modelling uncertainty or concurrency. In Figure 1 for example, the nodes denote time instants and the edges denote "precedes in time" ($<$). t_2 precedes both t_3 and t_4 in time, but no information is available about the relative position in time of t_3 and t_4. In case of concurrent behaviour, causality must not be violated within the individual concurrent threads, but the time-ordering between concurrent events may be left unspecified [Mil93]. Mathematically, partial ordering leads to a lattice structure.

For total ordering, it must be possible to compare any two elements of T:

$$\forall t, t' \in T : t < t' \vee t' < t \vee t = t'.$$

In case of total ordering, *intervals* may be defined. With intervals, the past $T_{t[}$ and future $T_{]t}$ of an instant $t \in T$ may be defined

$$T_{t[} = \{\tau | \tau \in T, \tau < t\},$$
$$T_{]t} = \{\tau | \tau \in T, t < \tau\}.$$

Once intervals have been defined,

$$T_{\langle t_b, t_e \rangle}$$

denotes a time interval, where $\langle t$ means $]t$ or $[t$. In many cases, $(T, +)$ is an Abelian group with zero 0 and inverse $-t$. In case $+$ is order preserving

$$t_1 < t_2 \Rightarrow t_1 + t < t_2 + t.$$

Common time bases (with appropriate $<$ and $+$) are

- $T = \{NOW\}$. Models such as algebraic models are *instantaneous*. The time base is a singleton.
- $T = \mathbb{R}$. Models with this time base are called *continuous-time* models. Note how *discrete event models* have $\mathbb{R}$ as a time base. However, only at a finite number of time-instants in a bounded time-interval, an event different from the non-event ϕ occurs.

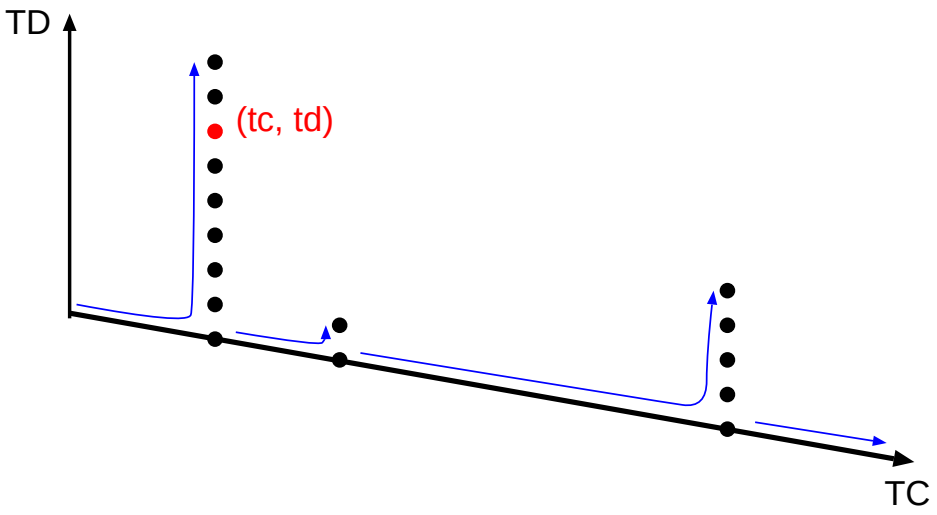

Figure 2: Time base for hybrid system models

- $T = \mathbb{N}$ (or isomorphic). Models with this time base are called *discrete-time* models. Some formalisms such as Finite State Automata (FSA) do not have an explicit notion of time (unlike their extension, timed automata). Hence, they are often called *untimed* models. There is however a notion of progression (from one state to another). According to our general definition, the index of progression, a natural number, is time.

In *hybrid* system models which combine aspects of continuous and discrete models [MB02], a system evolves continuously over time ($\mathbb{R}$) until a certain condition is met. Then, *instantaneously* (the continuous time does not progress), the system may go through a number of discrete states (the index of progression is discrete) before continuing its continuous behaviour. To uniquely describe progression (of generalized time) in this case, a tuple (t_c, t_d) depicted in Figure 2 is needed. Even when a series of discrete transitions keeps returning to the *same* state, the discrete index t_c allows one to distinguish between them. The time base used is

$$T = \{(t_c, t_d) | t_c \in \mathbb{R}, t_d \in \{1, \ldots, N(t_c)\}\}.$$

Here, $N(t_c)$ (≥ 1) describes the number of discrete transitions the system goes through at continuous time t_c. Obviously, only a partial ordering will be defined over T which consists of first testing the relationship between the t_c components, and subsequently (if equal), that between the t_d components.

In case of Partial Differential Equations (PDEs), the time base remains $\mathbb{R}$. The other *independent variables* (often space in the form of some coordinate system) should be seen as infinitely many state-variable *labels* or *generalized coordinates*.

1.1 Behaviour

Given a time base, we wish to formalize *behaviour* over time. This is done by means of a time function, called *trajectory* or *signal*

$$f : T \to A$$

describing, at each time t, the value of the signal. A denotes the set of valid values f can take over T. The time base may be restricted to a subset of T: $T' \subseteq T$. The restriction of f to T' is

$$f|T' : T' \to A,$$

$$\forall t \in T' : f|T'(t) = f(t).$$

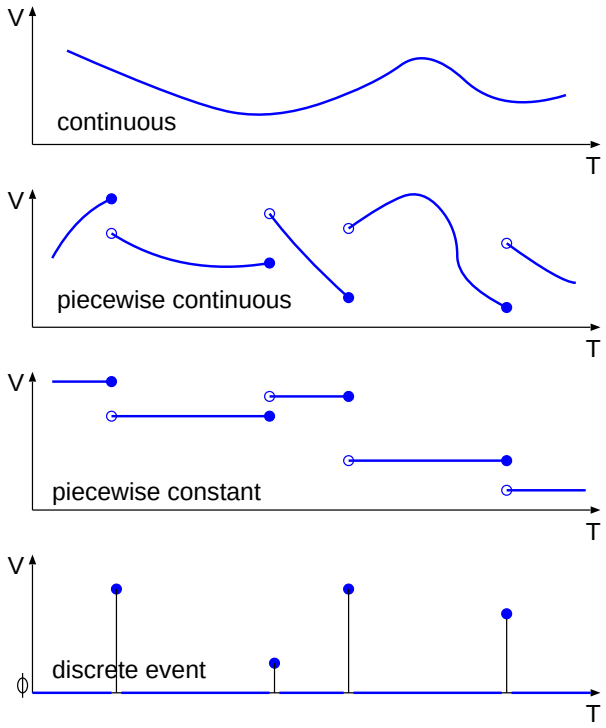

Figure 3: Segment types

The past of f is defined as $f|T_{t\rangle}$. The future of f is defined as $f|T_{\langle t}$.
The restriction of f to an interval is called a *segment* ω

$$\omega : \langle t_1, t_2\rangle \to A.$$

The set of all allowed segments is called Ω. It is a subset of all possible segments (A, T). The length of a segment:

$$l : \Omega \to T_0^+$$
$$\omega \to t_f - t_i, dom(\omega) = \langle t_b, t_e\rangle$$

Segments are contiguous if their domains $\langle t_1, t_2\rangle$ and $\langle t_3, t_4\rangle$ are contiguous: $t_2 = t_3$.
Contiguous segments may be concatenated – $\omega_1 \bullet \omega_2$:

$$\omega_1 \bullet \omega_2(t) = \omega_1(t), \forall t \in dom(\omega_1);$$

$$\omega_1 \bullet \omega_2(t) = \omega_2(t), \forall t \in dom(\omega_2),$$

where $\langle$ and $\rangle$ must denote matching open/closed interval boundaries to ensure the concatenated segment is still a *function* (*i.e.,* has a unique value in each point of its domain).
A desirable property of a set of segments Ω is that it is closed under concatenation $\bullet$: concatenating *any* left and right segment of a segment yields the same segment:

$$\forall t \in dom(\omega) : \omega_{t\rangle} \bullet \omega_{\langle t} = \omega.$$

Figure 3 shows some common segment types: continuous, piecewise continuous, piecewise constant and discrete event. Note how for discrete event systems, inputs and output segments are *event segments*

$$\omega : \langle t_1, t_2 \rangle \rightarrow A \cup \{\phi\},$$

with ϕ the non-event. For such systems, the internal state behaviour is piecewise constant (the internal state only changes at event times).

2 Levels of system specification

With a time base and segments defined, we can build a hierarchy of system specification structures which incorporate progressively more knowledge about the system. All these structures will view the system as a *box* interacting with its environment through a well defined *interface*. The levels presented here elaborate on the hierarchy first proposed by Klir [Kli85] and later modified by Zeigler [ZPK00].

2.1 Observation Frame

At the lowest level, the only knowledge we have of the behaviour of a system is how we wish to observe it: which time base to use and which quantities to observe at instants from the time base. This is represented in the form of an Observation Frame O:

$$O \equiv \langle T, X, Y \rangle.$$

T with appropriate operators forms a time base. X is the input value set. It is a model for the input (influencing the behaviour of the system) variables we consider. Y is the output value set. It is a model for the system response variables.

2.2 I/O Relation Observation

Once the interface variables to observe as well as their value ranges have been determined, all possible *relationships* between input and output segments can be recorded

$$IORO \equiv \langle T, X, \Omega, Y, R \rangle.$$

Here, $\langle T, X, Y \rangle$ is an Observation Frame, and Ω is the set of all possible input segments for this system. Note how Ω allows one to specify how the system's *environment* may influence the system. As such, Ω formalizes the Experimental Frame's generator presented before. Ω is a subset of all mathematically possible segments with T as domain and X as image. R is the *I/O relation*

$$R \subseteq \Omega \times (Y, T),$$

where $\Omega \subseteq (X, T)$, all possible segments with T as domain and X as image. (Y, T) stands for all possible segments with T as domain and Y as image. Input segments ω and output segments ρ are defined as

$$\omega : \langle t_i, t_f \rangle \rightarrow X;$$

$$\rho : \langle t_i, t_f \rangle \rightarrow Y.$$

Though not necessary, it is common to observe input and output segments over the same time domain. The relation R relates input and output segments

$$(\omega, \rho) \in R \Rightarrow dom(\omega) = dom(\rho).$$

As will be discussed further on, general *non-causal* relationships between interface variables, not specifying *a priori* which are input and which are output may be specified by R. Higher levels in the specification hierarchy are explicitly causal.

2.2.1 From I/O Relation Observation to Observation Frame

It is possible to go from an I/O Relation Observation model specification to an Observation Frame level model by merely discarding the Ω and R information at the I/O Relation Observation level.

2.3 I/O Function Observation

At the I/O Relation Observation level, an input segment ω is not necessarily associated with a unique output segment ρ. This is due to a limited knowledge of the internal working of the system. At the I/O Function Observation level, we want to associate a *unique* output segment with every input segment. Therefore, more information needs to be specified about the system. This is done in the form of a set F of *I/O functions* f. This leads to the I/O Function Observation structure

$$IOFO \equiv \langle T, X, \Omega, Y, F \rangle,$$

where $\langle T, X, Y \rangle$ is I/O Relation Observation, Ω is the set of all possible input segments, F is the set of I/O functions:

$$f \in F \Rightarrow f \subset \Omega \times (Y, T);$$
$$dom(f(\omega)) = dom(\omega).$$

f is conceptually equivalent to the system's *initial state*: For each f, an input segment will be transformed into a unique output segment.

2.3.1 From I/O Function Observation to I/O Relation Observation

It is possible to go from an I/O Function Observation to an I/O Relation Observation by constructing R from F:

$$R = \bigcup_{f \in F} f.$$

2.4 I/O System

In some cases, we have some insight into the *internal* working of the system. This insight usually consists of a number of *descriptive variables* and how their values evolve over time. Under certain conditions, these variables are *state variables*. One usually tries to keep the set of state variables as small as possible.

In general systems theory [Wym67], a causal (output is the consequence of given inputs), deterministic (a known input will lead to a unique output) system model SYS is defined. It is a template for a plethora of different formalism such as Ordinary Differential Equations, Finite State Automata, Difference Equations, Petri Nets, *etc.* Its general form is

$$SYS \equiv \langle T, X, \Omega, Q, \delta, Y, \lambda \rangle$$

T	time base
X	input set
$\Omega = \{\omega : T \to X\}$	input segment set
Q	state set
$\delta : \Omega \times Q \to Q$	transition function
Y	output set
$\lambda : Q \to Y$ (or $Q \times X \to Y$)	output function

$$\forall \omega, \omega' \in \Omega, \delta(\omega \bullet \omega', q_i) = \delta(\omega', \delta(\omega, q_i)).$$

The time base T is the formalisation of the independent variable time. The input set X describes all possible allowed input values (possibly a product set). An input segment ω represents input during a time-interval. The history of system behaviour is condensed into a *state* (from a state set Q). The dynamics is described in a transition function δ which takes a current state, and applies an input segment $\omega \in \Omega$ to it to obtain a new state. The system may generate output. This output is obtained as a function λ

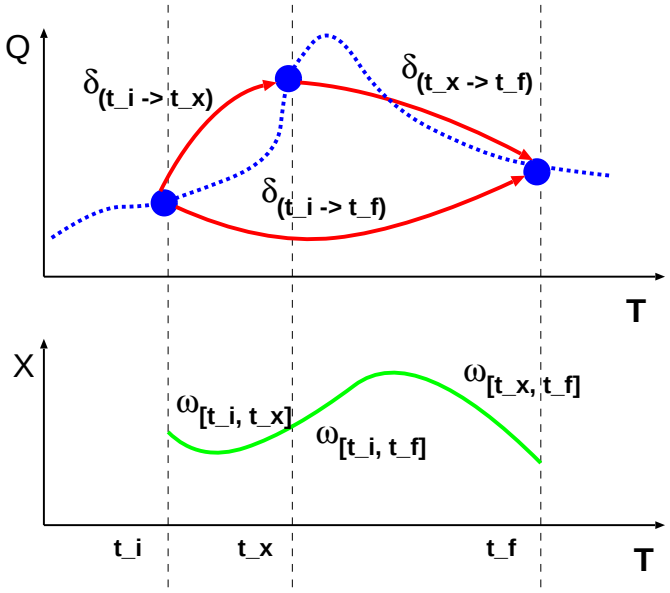

Figure 4: SYS state transition property

of the state (and more generally, of the current input too). State and transition function must obey the composition or semigroup property as shown in Figure 4. This property, whereby a transition over a time interval $[t_i, t_f]$ can always be split into a composition of transitions over arbitrary sub-intervals, is the basis of all model simulators. Obviously, this also requires Ω to be closed under concatenation as well as left segmentation. Closure under concatenation requires that the concatenation of any two contiguous segments in Ω is again in Ω. Left segmentation requires that any left segment of a segment in Ω is an element of Ω in its own right.

As the output function is described separately, efficient simulators will *only* invoke this function (which may be large and compute-intensive) when the user needs to observe output. Note how the *output intervals* (times between outputs) are not part of the model, but rather of the simulation experiment. Figure 5 shows how output need not be produced at each transition time. Even though a model written by a user may not distinguish between δ and λ, a simple *dependency analysis* will identify which variables and expressions are not needed to compute δ. Such variables are output variables $\in Y$ and the expressions belong in λ.

As SYS is a template for a host of causal, deterministic formalisms, it is possible to describe both models of the vessel example presented earlier. In the Ordinary Differential Equation (ODE) case, the time base is continuous ($\mathbb{R}$). The transition function is written in integral form. Different numerical approximations of the integral can be used in the implementation of an abstract simulator.

2.4.1 From I/O System specification to I/O Function Observation

It is possible to go from an I/O System specification to an I/O Function Observation. For a given initial condition q and a given input segment ω, we can define a *state trajectory* $STRAJ_{q,\omega}$ from SYS

$$STRAJ_{q,\omega} : dom(\omega) \rightarrow Q,$$

with

$$STRAJ_{q,\omega}(t) = \delta(\omega_{t\rangle}, \forall t \in dom(\omega).$$

From this state trajectory, an *output trajectory* $OTRAJ_{q,\omega}$ may be constructed

$$OTRAJ_{q,\omega} : dom(\omega) \rightarrow Y,$$

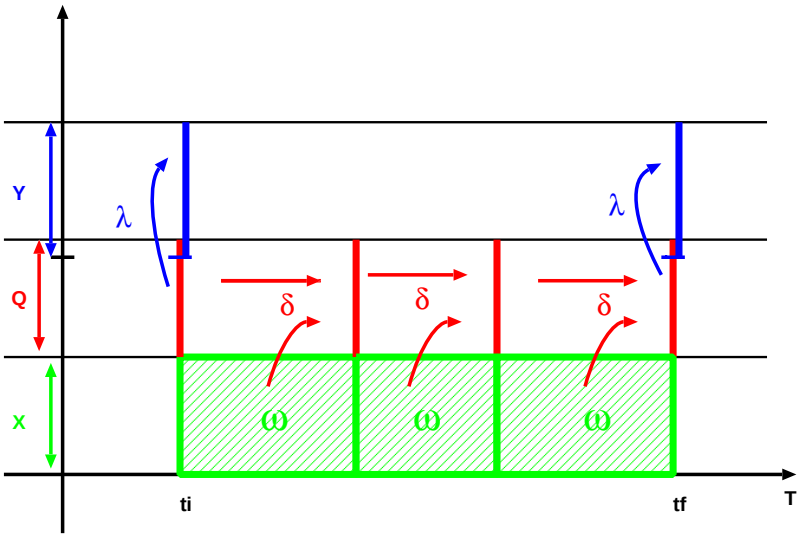

Figure 5: Simulation kernel operation

with

$$OTRAJ_{q,\omega}(t) = \lambda(STRAJ_{q,\omega}(t), \omega(t)), \forall t \in dom(\omega).$$

Thus, for every q (initial state), it is possible to construct

$$\mathcal{T}_q : \Omega \to (Y, T),$$

where

$$\mathcal{T}_q(\omega) = OTRAJ_{q,\omega}, \forall \omega \in \Omega.$$

The I/O Function Observation associated with SYS is then

$$IOFO = \langle T, X, \Omega, Y, \{\mathcal{T}_q(\omega) | q \in Q\}\rangle.$$

Subsequently, we may derive the I/O Relation Observation by constructing the relation R as the union of all I/O functions:

$$R = \{(\omega, \rho) | \omega \in \Omega, \rho = OTRAJ_{q,\omega}, q \in Q\}.$$

In SYS, δ is *deterministic*: applying the same input segment to the same state will always lead to the *same, unique* new state (and output). Often, deterministic simulation kernels are used to simulate non-deterministic models. Two main approaches are possible:

1. A deterministic model is decorated with transition probabilities as shown in Figure 6. The same model is then simulated a number of times, with the same initial conditions and parameters. Whenever a non-deterministic transition is encountered however, a unique, deterministic, transition is chosen by sampling from a stochastic distribution, taking into account the transition probabilities in the model. Thus, from the point of view of the simulation engine, it is simulating a deterministic model. To be able to make meaningful statements about the behaviour of the non-deterministic model, a sufficient number of samples must be simulated to obtain statistically relevant *estimates* of performance metrics (such as average queue lengths in a queueing model). In discrete event simulation in particular, this approach is common and its statistical aspects have been studied in great detail [LK91]. In a slightly modified form, this approach is called Monte Carlo simulation.

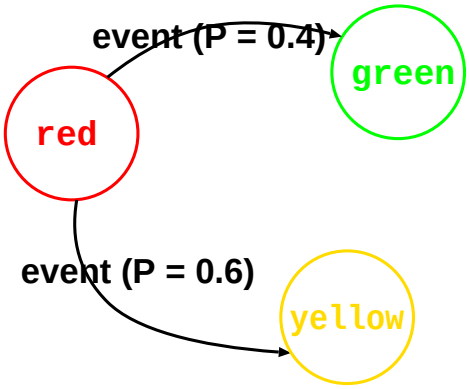

Figure 6: Non-deterministic model with transition probabilities

2. One may wish not to specify any probability distribution but leave the uncertainty of making a transition to more than one new state in the transition function. In case of State Automata, this turns the transition graph into a transition hypergraph [Har88]. Such a specification can always be transformed into a deterministic one by constructing a new state set $Q_{new} = 2^Q$, the set of all subsets (powerset) of Q [Cas93]. A new transition function is constructed describing the –now deterministic– transition to a new state, denoting the set of states from Q to which a non-deterministic transition existed in the old model. It is noted that in quantum physics, evolution over time of a wave function (a distribution interpreted as being probabilistic) is also deterministic.

Zeigler [ZPK00] presents a refinement of SYS in which behaviour is specified as an *iterative* application of *generator segments*. Arbitrary input segments are generated from elementary segments.

It should be noted that though models may be iteratively simulated, this is not necessary per se. If a symbolic (analytical) solution can be found, this is often preferable. Analytical solutions usually describe a (parametrised) class of solutions rather than a single one. Also, accumulation of numerical errors is often avoided. As an example, the following model described in the Difference Equation formalisms ($T = \mathbb{N}, Q = \mathbb{R}$)

$$\begin{cases} x_1 &= 1 \\ x_{i+1} &= ax_i + 1, i > 1 \end{cases}$$

can be re-written as

$$\begin{cases} x_n &= 1 + a + a^2 + \ldots + a^{n-1} \\ ax_n &= a + a^2 + \ldots + a^{n-1} + a^n, \end{cases}$$

subtracting the second from the first equation leads to the instantaneous (no iteration required) solution

$$x_n = \frac{1 - a^n}{1 - a}.$$

References

[Cas93] Christos G. Cassandras. *Discrete Event Systems*. Irwin, 1993.

[Har88] David Harel. On visual formalisms. *Communications of the ACM*, 31(5):514–530, May 1988.

[Kli85] George J. Klir. *Architecture of Systems Problem Solving*. Plenum Press, 1985.

[LK91] Averill M. Law and David W. Kelton. *Simulation Modeling and Analysis*. McGraw-Hill, 1991.

[MB02] Pieter J. Mosterman and Gautam Biswas. A modeling and simulation methodology for hybrid dynamic physical systems. *Transactions of the Society for Computer Simulation International*, 2002.

[Mil93] Robin Milner. Elements of interaction. *Communications of the ACM*, 36(1):70–89, January 1993. Turing Award Lecture.

[Nan81] Richard E. Nance. The time and state relationships in simulation modeling. *Communications of the ACM*, 24(4):173–179, April 1981.

[Wym67] A. Wayne Wymore. *A Mathematical Theory of Systems Engineering – the Elements*. Wiley Series on Systems Engineering and Analysis. Wiley, 1967.

[ZPK00] Bernard P. Zeigler, Herbert Praehofer, and Tag Gon Kim. *Theory of Modelling and Simulation: Integrating Discrete Event and Continuous Complex Dynamic Systems*. Academic Press, second edition, 2000.