BioCro as a Dynamical System

Contents

1 Dynamical Systems: Real and Mathematical

In his book *Computation, Dynamics, and Cognition* (Giunti 1997), Giunti distinguishes between real dynamical systems and mathematical dynamical systems:

> A real dynamical system is any real system that changes over time. Therefore, since any real system can be thought to change in time (in some respect), any real system is a real dynamical system. A mathematical dynamical system, on the other hand, is an abstract mathematical structure that can be used to describe the change of a real system as an evolution through a series of states.

It should be emphasized that when we create a mathematical system to model a real one, we are doing more than just quantifying attributes of the real system; we are also selecting which attributes to incorporate into our model and which ones to ignore. For there are an endless variety of attributes which could describe the state of a real system, and we can't even begin to hope to be able to model all of them.

As stated in the Giunti quote, a mathematical dynamical system will describe the change of a real system as an evolution through a series of states. (For now, we will interpret "series of states" loosely, so as to encompass models that describe this change as a continuous evolution of the system state as well as those that describe change in terms of a discrete sequence of states.) The real usefulness of such a mathematical structure, however, comes when it goes beyond merely describing the series of states: the real power comes when we are able to derive a complete picture of the evolution of a system from only partial knowledge of the system, knowledge possibly consisting of, for example, the state of the system at some particular time, the environment in which the system is operating, and some general knowledge of the processes that determine how the system behaves.

1.1 An example: the falling-body problem

A classic example from physics is the falling-body problem: Given a stationary, compact, relatively dense object dropped from a height h_0 above the surface of the earth, what will be its height after a duration of time t? The partial knowledge we have of this system consists of:

1. the initial height h_0 of the object
2. the initial velocity v_0 of the object (In this system, we'll assume $v_0 = 0$: the object is stationary to begin with.)
3. the magnitude, which we'll call g, of the downward acceleration of the body caused by the earth's gravitational field

If we use the function $h(t)$ to embody the complete description of the evolution of the system—that is $h(t)$ tells the height of the body after an elapsed time of t—then our initial knowledge of the system consists of a system of equations putting certain constraints on the function h:

$$\begin{aligned} h(0) &= h_0 \\ h'(t)|_{t=0} &= 0 \\ h''(t) &= -g \end{aligned} \tag{1.1}$$

(The third constraint would more accurately be written as

$$h''(t) = -g \quad \textit{if } \text{h(t)} > 0,$$

but for simplicity, we will only consider the system over durations sufficiently small that the object has not yet hit the ground.)

Anyone who has studied differential equations will immediately recognize this as an initial-value problem, one whose unique solution is

$$h(t) = h_0 - 1/2\, gt^2.$$

Thus, from knowing only the initial height and velocity of the object and some basic principles of physics, we are able to obtain a complete description of the evolution of this "falling object" system over time.

1.2 Some comments on mathematical abstraction

As we mentioned, any mathematical dynamical system that purports to model a real system will necessarily leave much out. In choosing what attributes to retain in our abstract model, there are two main considerations: First, which attributes of the system are of most interest to us? For a model of plant growth, this might include, for example, the rate of growth (mass accumulation) of a plant; the nutrient or energy content of the growing plant; the impact of a group of plants on the surrounding environment, including the temperature and CO_2 content of the surrounding air, or the rate of erosion of the soil substrate; or the resilience of the plant in drought conditions.

Second, there are attributes that may not be of particular intrinsic interest but may help in predicting the behavior of those attributes that *are* of interest.

Returning to the falling-object model for a moment, if the primary object of interest in the *real* falling object system is the height of the object at any given time, then there are certain things about the system we may safely ignore: the color of the object or the time of day, for example, will probably have no bearing on the trajectory of object's motion. On the other hand, knowing the velocity of the object at any given time is crucial to predicting its height over time, even if we may have no intrinsic interest in knowing the velocity.

Other attributes that *could* have some bearing on the motion of the object are (to give a very few examples)

- the size and shape of the object
- the mass of the object
- the air currents in the vicinity of the object's path

But as the Italian experimenters of the 16th century demonstrated, the weight of a compact and relatively dense object has little impact upon the rate at which it falls. It turns out, as a matter of fact, that the predictive accuracy of our model, in which we look only at the height and downward velocity of the object (and the ambient gravitational field) and ignore all other attributes of the system, is rather good in the case of compact, relatively-dense objects. (It's not so good for feathers.)

2 Continuous time versus discrete time

In the system just shown—a mathematical model of a real-world dynamical system—the differential equation constraining the solution has an exact solution as an easily-computable function. Most often, however, we will not be able to find an exact solution, and so we will have to settle for a numerical solution.

We will show how to model the falling-object system numerically, even though this is one case where we don't really need to resort to such methods.

2.1 Euler's method

Euler's method, the most basic of methods for numerical integration of ordinary differential equations, may be applied to any system in which the current rate of change of each of the state variables may be expressed as a function of the current state. Here, we use *state variable* to mean any quantifiable attribute of a system; the *state* of a system is then the conglomeration of the values of all of these variables at any particular time. Euler's method makes the assumption that, given a known state of a system at some particular time, the state of the system at some very small interval of time later can be closely approximated by assuming that the rate of change of each state variable remains essentially constant over that very small time interval.

If $\mathbf{s}$ denotes the state, with $\mathbf{s}(t)$ denoting its value at time t, and if x is one of the state variables, with $x(t)$ denoting its value at time t, then, given

$$dx/dt = f(\mathbf{s}),$$

we assume that for any sufficiently small interval of time Δt,

$$x(t + \Delta t) \approx x(t) + f(\mathbf{s}(t)) \cdot \Delta t.$$

2.2 Applying Euler's method to the falling-body problem

In the system of equations at (1.1), h is the only system variable. But there is no valid equation that gives dh/dt as a function of the state when the state is represented by h alone.

To solve this problem, we will also consider the velocity to be a part of the state. If we think of the state of a system as a snapshot of the system at a particular time, this may seem somewhat counter-intuitive. While it is clear how we might consider the height of the object as part of that snapshot, if we take the notion of "snapshot" in its literal sense, it may seem odd to consider the velocity, the rate of change of position, as part of a snapshot: when we take a literal snapshot, we can't know how fast the objects in the snapshot are moving, at least not if the snapshot is instantaneous. We won't dwell on this philosophical problem: if it bothers you, imagine that the object has a speedometer attached and that our snapshot shows the position of its needle as the object is falling.

And so now, the states of our system have two components, height and velocity, and we can think of each state $\mathbf{s}$ as a point in a 2-dimensional Euclidean space, that is,

$$\mathbf{s} = (s_0, s_1).$$

We will identify the height h with the first component s_0 and the velocity v with the second component s_1. We will consider v to be the velocity in the upward direction so that when the object is falling, $v < 0$. We write $v(t)$ to denote v as a function of time.

Since $v = dh/dt$, we can rewrite the system (1.1) as

$$\begin{aligned} dh/dt &= v \qquad (2.1)\\ dv/dt &= -g \qquad (2.2)\\ h(0) &= h_0\\ v(0) &= 0 \end{aligned}$$

Now we can use equations (2.1) and (2.2) to obtain the Euler method formulas for estimating the state at time $t + \Delta t$ from the state at time t:

$$\begin{aligned} h(t + \Delta t) &= h(t) + v(t) \cdot \Delta t \\ v(t + \Delta t) &= v(t) - g \cdot \Delta t \end{aligned}$$

Let us consider this system over some sequence of times $0 = t_0, t_1, t_2, t_3, ...$ where for each i, $t_{i+1} = t_i + \Delta t$. Further, let write δ for Δt, and let π_0 and π_1 denote the projection of the state $\mathbf{s}$ onto its components, that is

$$\pi_0(\mathbf{s}) = s_0 \tag{2.3}$$

and

$$\pi_1(\mathbf{s}) = s_1. \tag{2.4}$$

Now we can write a recursive definition for the state $\mathbf{s}$ as a function of t:

$$\begin{aligned} \mathbf{s}(t_0) &= (h_0, 0) \\ \mathbf{s}(t_{i+1}) &= (\pi_0(\mathbf{s}(t_i)) + \delta \cdot \pi_1(\mathbf{s}(t_i)), \pi_1(\mathbf{s}(t_i)) - \delta \cdot g) \quad \text{for i}\geq 0. \end{aligned} \tag{2.5}$$

2.3 Note about abstraction and recursive systems

We have just performed the following abstraction to arrive at a recursively-defined function giving the state of a system as a function of time:

$$\begin{tikzcd} \text{REAL SYSTEM} \arrow[d] \\ \text{CONTINUOUS MATHEMATICAL SYSTEM (ODE SYSTEM)} \arrow[d] \\ \text{DISCRETE-TIME APPROXIMATION (RECURSIVE EQUATIONS)} \end{tikzcd}$$

It should be pointed out, however, that not all discrete-time abstract dynamical systems arise as abstractions of real systems or even as abstractions of continuous-time abstract systems.

Consider, for example, a system with a state space $\mathbf{Z}^2$ consisting of all ordered pairs $\mathbf{v} = (v_0, v_1)$ of integers, and a transition rule

$$\mathbf{v}(t_{i+1}) = (\pi_1(\mathbf{v}(t_i)), \pi_0(\mathbf{v}(t_i)) + \pi_1(\mathbf{v}(t_i))).$$

Given an initial state $\mathbf{v}(t_0)$, we now have a way to compute the state $\mathbf{v}(t_i)$ at any time t_i. This abstract dynamical system may not have any relationship to any real dynamical system we might imagine, but it is an abstract dynamical system nevertheless.

(If we take $\mathbf{v}(t_0) = (0, 1)$, by the way, the function $F : \mathbf{N} \to \mathbf{Z}$ defined by the rule

$$i \mapsto \pi_0(\mathbf{v}(t_i))$$

defines the Fibonacci sequence $0, 1, 1, 2, 3, 5, 8, 13, \dots$.)

Another class of discrete-time abstract dynamical systems are the cellular automata. These, however, may have some value in modeling real-world phenomena. (See, for example, Deutsch (2005).)

3 An overview of some abstract dynamical system formulations

This section will provide a short survey of some formulations of abstract dynamical system, along with discussion of how BioCro's terminology may differ in sometimes unexpected ways from terminology the reader may be used to.

(For a insightful and thoroughly abstract mathematical study of the theory of general systems, dynamical and otherwise, see Mesarović and Takahara (1975).)

3.1 Some notational preliminaries

As is common, we will take $f : C \to B$ to mean "f is a function with domain C taking values in the set B." Usually, this means $f(c)$ is defined for every $c \in C$, but in what follows, we occasionally won't always be entirely strict about this. Following Vaught (1985), we write B^C to denote the set of all such functions f.[1] We use $\mathbf{R}$, $\mathbf{Z}$, and $\mathbf{N}$ to denote the real numbers, the integers, and the natural numbers (the finite ordinals, including zero), respectively. $\mathbf{Z}^+$ denotes the *positive* integers.

Following von Neumann, it will sometimes be convenient to identify each natural number n with the set of all of its predecessors. For example, $5 = \{0, 1, 2, 3, 4\}$. This is particularly useful when speaking of Euclidian spaces. For example, $\mathbf{R}^3$, Euclidean 3-space, is usually thought of as the set of all 3-coordinate vectors (x, y, z). But we can equally well consider it to be the set of all mappings $v : \{0, 1, 2\} \to \mathbf{R}$, which, using the set-of-functions notation given above, can be denoted $\mathbf{R}^{\{0,1,2\}}$. (Or, using von Neumann's notion of ordinals, this, too, is denoted $\mathbf{R}^3$, since $3 = \{0, 1, 2\}$!) Thus, we can identify a 3-tuple (x, y, z) with a mapping $v : \{0, 1, 2\} \to \mathbf{R}$, where $v(0) = x$, $v(1) = y$, and $v(2) = z$. We often write v_i in place of $v(i)$ and identify a mapping $v : \{0, 1, 2, \dots, n-1\} \to \mathbf{R}$ with an n-tuple or n-coordinate vector $\mathbf{v} = (v_0, v_1, v_2, \dots, v_{n-1})$. Often, however, there will be some level of indirection involved in our use of the notation v_i for a coordinate of $\mathbf{v}$. For example, if $\mathbf{U}$ is a proper subspace of $\mathbf{R}^n$, so that $\mathbf{U} = \mathbf{R}^U$ for some proper subset U of $n = \{0, 1, \dots, n-1\}$, then we may regard u_i as the value taken by the ith member of U in some arbitrary but fixed ordering of the members of U. We even allow the case where the function domain U isn't a set of intergers at all but just some finite collection of objects. In this case, in the context of considering vectors (*qua* mappings) $\mathbf{v} \in \mathbf{R}^U$, v_i may denote alternately the i'th member of U in some fixed enumeration; the name of a variable associated with that member; or the value of the i'th coordinate of some particular vector $\mathbf{v}$. In the later case, v_i doesn't abbreviate $v(i)$ for some function $v \in \mathbf{R}^U$. Rather, it stands for $v(u_i)$, where u_i denotes "the i'th member of U."

Any function $f : C \to B$ may be identified with the set $\{(c, f(c)) : c \in C\}$. Thus, the target set B is not an intrinsic part of the function f. But, defining the *image set* of f as

$$\operatorname{Im} f = \{b : \text{there exists } c \in C \text{ such that} (c, b) \in f\},$$

we can at least say $\operatorname{Im} f \subseteq B$.

Given any function $f : C \to B$ and any subset $C_0 \subseteq C$, we can define $f|C_0$, the *restriction of f to C_0*, as

$$f|C_0 := \{(c, b) \in f : c \in C_0\}$$

We shall be particularly interested in restrictions of functions specifying points in Euclidean space. Suppose $\mathbf{x} \in \mathbf{R}^n$, and let W be an arbitrary subset of $n = \{0, 1, 2, \dots, n-1\}$. Then $\mathbf{x}|W$ will be a member of $\mathbf{R}^W$, the set of functions that assign a real number to each member of W. We regard R^W as a subspace of R^n. Moreover, if W has k members, R^W will be isomorphic to, but not necessarily equal to, the Euclidean space R^k. (R^W and R^k are equal iff $W = k$ (that is, iff $W = \{0, 1, \dots, k-1\}$).) We define the projection mapping $\pi^{n \to W} : R^n \to R^W$ by the rule

$$v \mapsto v|W.$$

[1]This notation is meant to be suggestive of exponentiation. In fact, if B and C are both finite sets having cardinalities $\|B\|$ and $\|C\|$ respecively, then the cardinality $\|B^C\|$ of B^C will be $\|B\|^{\|C\|}$. For example, if $C = \{\text{apple}, \text{pepper}\}$ and $B = \{\text{red}, \text{yellow}, \text{green}\}$, then there are $\|B\|^{\|C\|} = 3^2 = 9$ possible mappings of the two food items to the three colors.

More generally, given any two finite sets $W \subseteq U$ (not necessarily sets of integers), we may define a projection mapping $\pi^{U \to W} : R^U \to R^W$ by the rule

$$v \in R^U \mapsto v|W.$$

Just as we can restrict the domain of a function, we can expand it as well.

Suppose we have two functions $f \in \mathbf{R}^X$ and $g \in \mathbf{R}^Y$, where either X and Y are disjoint, or $f(z) = g(z)$ for all $z \in X \cap Y$. We define the *union* $f \cup g$ of f and g by the rule

$$(f \cup g)(z) = \begin{cases} f(z) & \text{if } z \in X, \\ g(z) & \text{if } z \in Y \setminus X. \end{cases}$$

Note that this is exactly the same function as that we get by regarding functions as sets of ordered pairs and then taking the literal union of f and g. Also, clearly,

$$f = (f \cup g)|X$$

and

$$g = (f \cup g)|Y.$$

Lastly, given any two sets A and B, we define the *Cartesian product* of A and B to be a set of ordered couples:

$$A \times B = \{(a, b) : a \in A \text{ and } b \in B\}$$

If $A = \mathbf{R}^X$ and $B = \mathbf{R}^U$ with $X \cap U = \emptyset$, then there is a natural isomorphism between $\mathbf{R}^X \times \mathbf{R}^U$ and $\mathbf{R}^{X \cup U}$ given by

$$(\mathbf{x}, \mathbf{u}) \mapsto \mathbf{x} \cup \mathbf{u}.$$

(The inverse mapping is given by $\mathbf{v} \mapsto (\mathbf{v}|X, \mathbf{v}|U)$ for $\mathbf{v} \in \mathbf{R}^{X \cup U}$.) Where convenient and warranted, we will consider $\mathbf{R}^X \times \mathbf{R}^U$ and $\mathbf{R}^{X \cup U}$ identical.

The notion of Cartisean product can be extended to three or more sets. For example, since there is a natural isomophism between $(A \times B) \times C$ and $A \times (B \times C)$, we can just write the product as $A \times B \times C$ and write its members as ordered triplets (a, b, c) (instead of $((a, b), c)$ or $(a, (b, c))$.

3.2 The Khalil model

The first model we consider is that described by Khalil (Khalil 2002). This model is both expressive and flexible, and we believe it is the most intuitively natural way to view the sort of systems BioCro deals with at the systems level.

In the opening chapter, the author introduces dynamical systems as a finite collection of coupled first-order ordinary differential equations

$$\begin{aligned} \dot{x}_0 &= f_0(t, x_0, \dots, x_{n-1}, u_0, \dots, u_{p-1}) \\ \dot{x}_1 &= f_1(t, x_0, \dots, x_{n-1}, u_0, \dots, u_{p-1}) \\ &\vdots \\ \dot{x}_{n-1} &= f_{n-1}(t, x_0, \dots, x_{n-1}, u_0, \dots, u_{p-1}). \end{aligned}$$

This is somewhat more general than the system we considered in section 1.1 in that the derivatives depend not only upon the state variables $x_0, x_1, \dots, x_{n-1}$, but also upon time t and what Khalil refers to as the *input* variables $u_0, u_1, \dots, u_{p-1}$.

Defining

$$\mathbf{x} = \begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ \vdots \\ x_{n-1} \end{bmatrix}, \quad \mathbf{u} = \begin{bmatrix} u_0 \\ u_1 \\ \vdots \\ \vdots \\ u_{p-1} \end{bmatrix}, \quad \mathbf{f}(t, \mathbf{x}, \mathbf{u}) = \begin{bmatrix} f_0(t, \mathbf{x}, \mathbf{u}) \\ f_1(t, \mathbf{x}, \mathbf{u}) \\ \vdots \\ \vdots \\ f_{n-1}(t, \mathbf{x}, \mathbf{u}) \end{bmatrix}, \tag{3.1}$$

the state equation may be written more succinctly as the vector equation

$$\dot{\mathbf{x}} = \mathbf{f}(t, \mathbf{x}, \mathbf{u}). \tag{3.2}$$

(Note that Khalil actually uses 1-based indexing of vector coordinates in his exposition, so the vector $\mathbf{x}$, for example, is defined by

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ \vdots \\ x_n \end{bmatrix}.$$

Here we use 0-based indexing instead in order to be consistent with other parts of this article.)

3.2.1 The notion of state in the Khalil and BioCro models

Khalil remarks that the state variables "represent the memory that the dynamical systems has of its past." As Laplace (Dale 1995) remarks, "We ought then to consider the present state of the universe as the effect of its previous state and as the cause of that which is to follow." The inputs in the model, on the other hand, are in general completely arbitrary. They are not determined by the state or by their own past or future values. They help determine, but are not determined by, the evolution of the state of the system. In a sense, they are like the hand of the experimenter-god touching and influencing this otherwise mechanistically-determined system.

In BioCro, the inputs are referred to as *drivers*. It should also be remarked that in the terminology used to discuss BioCro, and in the terminology of the software itself, the driver variables are considered a part of the state. This is partly for expedience: in certain contexts, it is convenient to treat all quantities that vary with time (and sometimes even those that don't) uniformly.

But convenience aside, there is also some philosophical justification for this: In many systems, the inputs may clearly be thought of as somehow external to the system. When studying an electrical circuit, for example, the experimenter may apply electrical inputs to the system and see how the system responds. Even in a controlled plant-growth experiment conducted in a climate-controlled greenhouse, the environmental inputs may be applied somewhat arbitrarily. In BioCro, by contrast, the inputs (the *drivers*) are usually related to weather and other aspects of the environment—temperature, humidity, radiation, and so on. Unlike in a controlled experiment, these environmental variables evolve according to their own laws; they are not under the control of the experimenter. In a truly comprehensive model, their laws of evolution would be included right alongside the laws determining plant growth. But generally, to do so would overly complicate the model, and by and large, given the inherently chaotic nature of meteorological processes and the vast amounts of additional data that would be required, it isn't even feasible to do so. So in BioCro, we regard them as a part of the system—as part of the *state* of the system—but as a part that is taken as given rather than as a part that is to be derived from some general rules governing the evolution of natural systems.

None of this is to say, of course, that BioCro can't also be used to model controlled experiments, such as those carried out in a green house; or to model "thought experiments" or "what if" scenarios: *What would happen if we used the weather data from 2005 but assumed a much higher CO_2 concentration?*

3.2.2 Variants of Khalil's model

At this point, it is worth bringing up two restricted versions of the Khalil model.

The first is when equation (3.2) can be written as

$$\dot{\mathbf{x}} = \mathbf{f}(t, \mathbf{x}). \tag{3.3}$$

Khalil refers to this as the *unforced* state equation: it lacks any explicit mention of inputs. But, he points out, if the input can be specified as an explicit function of time,

$$\mathbf{u} = \gamma(t), \tag{3.4}$$

an explicit function of the state,

$$\mathbf{u} = \gamma(\mathbf{x}), \tag{3.5}$$

or an explicit function of both,

$$\mathbf{u} = \gamma(t, \mathbf{x}), \tag{3.6}$$

then an equation of the form (3.2) can always be reduced to an equation of the form (3.3).

Khalil goes on to mention one particular special case of the class of systems described by equation (3.3): namely, those that are *autonomous* or *time-invariant*. A system is *autonomous* if the function f does not depend explicitly on t, that is, if

$$\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}). \tag{3.7}$$

The behavior of an autonomous system is invariant to shifts in time origin. (The falling body system considered above was autonomous: it doesn't matter precisely when the object is released, its subsequent motion will follow the same pattern.)

With the exception of certain "toy" examples, such as those based upon the harmonic oscillator module, the systems with which BioCro deals will, on a conceptual level, almost invariably be non-autonomous or time-varying. This is largely due to the time-dependence of the drivers: it matters whether we start our plants growing in March or in May. (Formally, however, as we shall see, a non-autonomous system can be made into an automous one by introducing a time-related variable into the state.)

We will discuss the Khalil model further we discuss the systems model used by BioCro.

3.3 The Giunti-Mazzola model

The model due to Giunti and Mazzola is a further generalization of the autonomous version of the Khalil model, though cast in a somewhat different form. Being autonomous, it is in some respects more restrictive than the general model given in (3.2); but in other respects it is considerably more general.

We highlight this model for two reasons: First, it is mentioned in the supplementary materials to Lochocki et al. (2022). Second, it generalizes the concept of time used for dynamical system from the real numbers (which the Khalil model assumes) to any monoid. In particular, we may consider time domains consisting of the non-negative integers, or some fixed multiple of the same, such as the non-negative even integers. This is one of the natural ways to view time in BioCro systems: in an idealized view of BioCro-modeled systems, time is continuous, but on a practical level, we compute the behavior of such systems in discrete jumps.

We quote Giunti and Mazzola's definition of a dynamical system verbatim (*Definition 1* in Giunti and Mazzola (2012)):

DS_L is a dynamical system on L iff DS_L is a pair $(M, (g^t)_{t \in T})$ and L is a pair $(T, +)$ such that

(i) $L = (T, +)$ is a monoid. Any $t \in T$ is called a *duration* of the system, T is called its *time set*, and L its *time model*;

(ii) M is a non-empty set. Any $x \in M$ is called a *state* of the system, and M is called its *state space*;

(iii) $(g^t)_{t\in T}$ is a family indexed by T of functions from M to M. For any $t \in T$, the function g^t is called the *state transition of duration t* (briefly, *t-transition*, or *t-advance*) of the system;

(iv) for any $v, t \in T$, for any $x \in M$,

  (a) $g^0(x) = x$, where 0 is the unity of L;

  (b) $g^{v+t}(x) = g^v(g^t(x))$.

Notice that not only can the time model now be any monoid, the state space can now be any non-empty set: it is no longer required to be a subset of a Euclidean space. It needn't even be a continuum! A fortiori, there is no longer any requirement that the state transitions be differential equation based. (In BioCro, however, no use is made of this generalization: in BioCro, the state space always *will* be a subset (in fact, a *connected* subset) of a Euclidean space, and state transitions (using *state* in Khalil's sense) will always be differential equation based.)

Instead of differential equations, we have condition (iv.b), sometimes called the *semi-group* property, which relates the structure of the time model to that of the class of state transitions. Just as T is a monoid with operation $+$ and additive identity 0, so too is the collection $(g^t)_{t\in T}$ of state transitions, with the monoid operation being function composition and the identity element being the identity function. Condition (iv) asserts that the mapping from $(T, +)$ to $((g^t)_{t\in T}, \circ)$ whereby $t \mapsto g^t$ is a monoid homomorphism.

Condition (iv) is in fact the crux of this definition of a dynamical system. Without it, there is no structure to the way in which such a system evolves: the system may pass from one state to the next willy-nilly without any constraint on the relationship between states over time.

Giunti and Mazzola's model definition was cited in the supplementary materials to Lochocki et al. (2022) as justification for considering all quantities involved in a system (except for time) as part of the state. Whatever the merits of that argument, in retrospect, this possibly amounts to a sort of cherry-picking of the Giunti-Mazzola definition because it is not altogether clear whether BioCro dynamical systems, as envisioned in that paper, have, in general, a collection of well-defined transition functions $(g^t)_{t\in T}$. Whether they do or do not turns on the question of how we interpret the stipulation, given in the supplement, that "the term state is used to refer to all quantities involved in the system, except time." We will elaborate on this question further in what follows.

3.4 The Barreira-Valls model

We briefly mention one further model for the discrete time case, mainly because the formulation is the epitome of simplicity. *Definition 1.1* in Barreira (2019) states simply

> *A map $f : \mathbf{X} \to \mathbf{X}$ is called a* dynamical system *with discrete time.*

The definition goes on to define higher-order mappings:

> *We define recursively*
>
> $$f^n = f \circ f^{n-1}$$
>
> *for each $n \in \mathbf{Z}^+$, with the convention that $f^0 = \mathrm{id}$. When f is invertible, we also define $f^{-n} = (f^{-1})^n$ for each $n \in \mathbf{Z}^+$.*

This is entirely homologous to the Giunti-Mazzola model for the case where the monoid chosen for the time model is either $\mathbf{Z}$ or $\mathbf{N}$.

Note that the recursion equation (2.5) derived from applying Euler's method to the falling body problem fits nicely into this model: take $\mathbf{X}$ to be Euclidean 2-space and the function f to be defined by the rule

$$(x, y) \mapsto (x + \delta y, y - \delta g).$$

(Barreira and Valls go on to define "a dynamical system *with continuous time*" (Barreira 2019, Definition 1.7) in precisely the same way as Giunti and Mazzola (for the case where the time model is $\mathbf{R}$ or $\mathbf{R}_{\geq 0}$)—that is, as a family of mappings indexed by time satisfying the semi-group property.)

4 The BioCro model

A BioCro system is determined by the specification of five entities:

1. A set of initial values
2. A set of (constant) parameter values
3. A set of drivers
4. A set of direct modules
5. A set of differential modules

These five entities tell us everything about the dynamics of the system that we need in order to "solve" it. (How, precisely, it will be solved is determined by specifying a solver.)[2]

Aside from differences in the concept of *state*, the Khalil model fits very well with an idealized version of the BioCro model in which time is considered to be continuous. We will see this in the Section 4.2, where we discuss the two models side-by-side. But first, we will elaborate a bit further on the Khalil model.

4.1 Elaboration on the Khalil model

Recall that the Khalil model expresses the derivative $d\mathbf{x}/dt$ of the state as a function of time t, the state $\mathbf{x}$, and the input $\mathbf{u}$:

$$\dot{\mathbf{x}} = \mathbf{f}(t, \mathbf{x}, \mathbf{u}).$$

Let us denote the domains of $\mathbf{x}$ and $\mathbf{u}$ (vis-à-vis the function $\mathbf{f}$) by $\mathbf{X}$ and $\mathbf{U}$. $\mathbf{X}$ and $\mathbf{U}$ are vector spaces over the reals, and following the conventions set out in Section 3.1, we may view them as sets of mappings from finite *index* sets into the reals. Thus

$$\mathbf{X} = \mathbf{R}^X \tag{4.1}$$

and

$$\mathbf{U} = \mathbf{R}^U \tag{4.2}$$

for some finite sets X and U, where we assume X and U are disjoint.[3]

[2]A less rigorous but much more succinct overview of the BioCro model is given in Appendix 2 of Lochocki et al. (2022).

[3]If $\|X\| = n$ and $\|U\| = p$, then $\mathbf{R}^X$ is isomorphic to the Euclidean space $\mathbf{R}^n$ and $\mathbf{R}^U$ is isomophic to $\mathbf{R}^p$. In many contexts, we could just go ahead and consider them not just isomophic but identical. But here we want to be more careful because we want to be able to simultaneously consider functions into $\mathbf{R}$ having non-overlapping domains.

Note that in practice, in a model of a real-world system, $\mathbf{f}$ may not be defined for all $\mathbf{x} \in \mathbf{X}$ and all $\mathbf{u} \in \mathbf{U}$. A coordinate corresponding to temperature in degrees Kelvin, for example, can not meaningfully take on values less than zero. In general, however, $\mathbf{f}$ will be defined for all $\mathbf{x} \in \mathbf{X}'$ and all $\mathbf{u} \in \mathbf{U}'$ where $\mathbf{X}'$ and $\mathbf{U}'$ are connected subsets of $\mathbf{X}$ and $\mathbf{U}$. We will reconsider this issue in Section 4.4.2.

Furthermore, recall that in the Khalil model, the value of $\mathbf{u}$ may given by some function of time and/or the state:

$$\mathbf{u} = \gamma(t),$$
$$\mathbf{u} = \gamma(\mathbf{x}),$$

or

$$\mathbf{u} = \gamma(t, \mathbf{x}).$$

In terms of the individual components of $\mathbf{u}$, each component u_i of $\mathbf{u}$ can be expressed as a function γ_i of t or $\mathbf{x}$ or both:

$$u_i = \gamma_i(t),$$
$$u_i = \gamma_i(\mathbf{x}),$$

or

$$u_i = \gamma_i(t, \mathbf{x}).$$

It is also possible that some u_i doesn't actually depend on either time or state, that it is in fact constant:

$$u_i = k \quad \text{for some } k \in \mathbf{R}. \tag{4.3}$$

We can, in fact, partition the variables $u_0, u_1, \dots, u_{p-1}$ comprising the varying input $\mathbf{u}$ into three groups:

1. Let u_i be in group K if the value of u_i depends on neither t nor $\mathbf{x}$; that is, it always has the same value, no matter what the state or the time.
2. Let u_i be in group D if the value of u_i depends on t alone.
3. Let u_i be in group W otherwise, that is if the value of u_i depends on the value of $\mathbf{x}$ (and possibly also on t).

Thus $U = K \cup D \cup W$, allowing us to partition the vector space $\mathbf{U}$ into corresponding sub-vector spaces $\mathbf{K}$, $\mathbf{D}$, and $\mathbf{W}$; that is,

$$\mathbf{U} = \mathbf{K} \times \mathbf{D} \times \mathbf{W},$$

where $\mathbf{K} = \mathbf{R}^K$, $\mathbf{D} = \mathbf{R}^D$, and $\mathbf{W} = \mathbf{R}^W$. Each input $\mathbf{u}$ may now be written as a triplet $(\mathbf{k}, \mathbf{d}, \mathbf{w})$ where $\mathbf{k} \in \mathbf{K}$, $\mathbf{d} \in \mathbf{D}$, and $\mathbf{w} \in \mathbf{W}$.[4] Moreover, there exist functions $\gamma^{\mathbf{D}} : T \to \mathbf{D}$ and $\gamma^{\mathbf{W}} : T \times \mathbf{X} \to \mathbf{W}$ and a constant function $\gamma^{\mathbf{K}}$ with codomain $\mathbf{K}$ such that

$$\mathbf{k} = \gamma^{\mathbf{K}}(), \tag{4.4}$$
$$\mathbf{d} = \gamma^{\mathbf{D}}(t), \tag{4.5}$$

and

$$\mathbf{w} = \gamma^{\mathbf{W}}(t, \mathbf{x}) \tag{4.6}$$

at any moment in the life of the system.

Since $\mathbf{u} = (\mathbf{k}, \mathbf{d}, \mathbf{w})$, we can rewrite the state equation (3.2) as

$$\dot{\mathbf{x}} = \mathbf{f}(t, \mathbf{x}, \mathbf{k}, \mathbf{d}, \mathbf{w}). \tag{4.7}$$

[4]Note that any (or all!) of the sets K, D, or W may be empty, in which case the corresponding vector space is zero dimensional and the corresponding vector argument can be eliminated from equation (4.7). (If all three sets are empty, then of course we have no inputs and the system is automatically described by the unforced state equation (3.3).)

But using equations (4.4), (4.5), and (4.6), we can eliminate $\mathbf{k}$, $\mathbf{d}$, and $\mathbf{w}$ to get $\mathbf{f}$ as a function of t and $\mathbf{x}$ alone:

$$\dot{\mathbf{x}} = \mathbf{f}(t, \mathbf{x}, \gamma^{\mathbf{K}}(), \gamma^{\mathbf{D}}(t), \gamma^{\mathbf{W}}(t, \mathbf{x})). \tag{4.8}$$

In other words,

$$\dot{\mathbf{x}} = \mathbf{f}^*(t, \mathbf{x}) \tag{4.9}$$

for some suitable function $\mathbf{f}^*$, so that we have now an unforced state equation as in equation (3.3).

4.2 BioCro viewed in terms of the Khalil model

In BioCro, as noted above, a system is determined when we specify its *initial values*, *parameters*, *drivers*, *direct modules*, and *differential modules*. How do these relate to the version of the Khalil model just discussed?

- The *initial values* correspond to the state $\mathbf{x}$ at some initial time t_0, which for convenience, we will always take to be 0. (Thus *time* is always the amount of time that has elapsed since the start of the simulation.) We will denote this initial state as $\mathbf{x}_0 = (x_{0,0}, x_{1,0}, \ldots, x_{n-1,0})$. In BioCro, it is the *initial values* specification that determines (in Khalil's terminology) which variables comprise the state, and the dimension of the state space $\mathbf{X}$ is equal to the number of variables in $\mathbf{x}_0$.

For now, we shall consider the initial values as part of the definition of a system only in so far as they determine the set of variables comprising the state space for the system. The specification of what values these variables have at time t_0 will be considered to be something associated with a particular run of a system and not something inherent in the system itself. This will make comparison with the Khalil and Giunti models easier, since those models don't specify anything analogous to an initial state.

In what follows, when we need to make this distinction, we shall refer to a dynamical system together with a specified initial state as a *run* of a system.

(Note that in most of the documentation and other writing about BioCro, *all* quantities are considered to be a part of the state, and the variables whose values are specified when providing the set of initial values to a BioCro simulation are referred to as *differential variables.* As such, when we are referring to things using this terminology, we will refer to $\mathbf{X}$ not as the state space, but as the *differential component of the state space.* In BioCro, it is the *initial values* parameter (and not, for example, the set of differential modules) that determines which variables belong to the differential component.)

- The *parameters* correspond to the sole value in the codomain of the constant function $\gamma^{\mathbf{K}}$. This will be a q-tuple of values $\mathbf{k} = (k_0, k_1, \ldots, k_{q-1})$, where q is the number of parameters, the dimension of the vector subspace $\mathbf{K}$.
- The *drivers* correspond to the function $\gamma^{\mathbf{D}} : T \to \mathbf{R}^r$, giving the value of $\mathbf{d}$ as a function of time. Writing $\mathbf{d}$ as $(d_0, d_1, \ldots, d_{r-1})$, where r is the number of driver variables (the dimension of $\mathbf{D}$), we can decompose $\gamma^{\mathbf{D}}(t)$ into scalar-valued functions $\gamma_0^{\mathbf{D}}(t)$, $\gamma_1^{\mathbf{D}}(t)$, ..., $\gamma_{r-1}^{\mathbf{D}}(t)$.

It should be noted that the driver functions $\gamma_i^{\mathbf{D}}$ are rarely functions that can easily be specified by and computed from some formula. In example shown in Appendix 1 of Lochocki et al. (2022), the function giving the value of the driver variable Q corresponding to photosynthetic photon flux density is based on the function

$$Q = \sin(\frac{t}{12 \cdot 3600}\pi) \cdot 2000 \times 10^{-6}. \tag{4.10}$$

Here, t is meant to represent the elapsed time in seconds, and in this example, the actual values fed into the BioCro system constructor are only the values of Q for a set of integral values of t, namely, $t = 0, 1, 2, \ldots, 43200$. As acknowledged in the appendix, this is a highly artificial example.

Usually, a driver variable function can only be defined via an equation[5] of the form

$$\gamma_i^{\mathbf{D}}(t) = \begin{cases} 0.046 & \text{if } t = t_0, \\ 0.023 & \text{if } t = t_1, \\ ... & \\ ... & \\ 1151.541 & \text{if } t = t_{4000}, \\ 747.040 & \text{if } t = t_{4001}, \\ ... & \\ ... & \\ 0.621 & \text{if } t = t_{8758}, \\ 0.874 & \text{if } t = t_{8759}. \end{cases} \tag{4.11}$$

Here, $t_0, t_1, ..., t_{8759}$ is a sequence of times representing the amount of time elapsed since the beginning of the simulation, with $t_0 = 0$, and for some fixed positive value Δt, $t_{j+1} = t_j + \Delta t$ for $0 \leq j < 8759$. In practice, in BioCro, this equation is usually specified implicitly via an R data frame: $\gamma_i^{\mathbf{D}}$ corresponds to some column of the data frame, and the value of that column in row j is that value for $\gamma_i^{\mathbf{D}}$ at time t_{j-1}.

(For some numerical methods, we need to know values of $\gamma_i^{\mathbf{D}}(t)$ for values of t *between* the time points given in this definition. This can be done by adding an additional case to the above rule; namely,

$$\gamma_i^{\mathbf{D}}(t) = \frac{t_{j+1} - t}{\Delta t}\, \gamma_i^{\mathbf{D}}(t_j) + \frac{t - t_j}{\Delta t}\, \gamma_i^{\mathbf{D}}(t_{j+1}) \quad \text{if } t_j < t < t_{j+1} \tag{4.12}$$

This makes $\gamma_i^{\mathbf{D}}(t)$ into a continuous piece-wise linear function.)

Now to the modules. As we will show below, any BioCro system is equivalent to a BioCro system having only a single module of each type, but having the same initial values, parameters, and drivers.[6] Thus, for simplicity, we will here consider only the case where there is a single module of each type.[7] Later, in the section on modularization, we will consider how a single module may be broken up into multiple modules.

- The *direct module* corresponds to the function $\gamma^{\mathbf{W}} : T \times \mathbf{X} \to \mathbf{W}$. In point of fact, we usually think of the direct module as corresponding to a function ${}^*\gamma^{\mathbf{W}} : \mathbf{X} \times \mathbf{K} \times \mathbf{D} \to \mathbf{W}$, but using equations (4.4) and (4.5) and substituting, we can derive a function $\gamma^{\mathbf{W}}$ that gives the value of $\mathbf{w}$ as a function of t and $\mathbf{x}$ alone.

  Note in particular that the number of output variables of this direct module gives the dimension of the $\mathbf{W}$ component of $\mathbf{U}$. We call these variables the *direct* quantities of the system (for lack of a better term) since they are the outputs of the direct module.

Two observations should be made here.

First, in the general case, where multiple direct modules $\mathcal{M}_1, \mathcal{M}_2, ...$ are used in the construction of our system, some of those modules may depend on the outputs of other direct modules. In this case, each module $\mathcal{M}_i$ corresponds to a function of the form

$$\gamma^{\mathbf{W}_i} : \mathbf{X} \times \mathbf{K} \times \mathbf{D} \times \overline{\mathbf{W}}_i \to \mathbf{W}_i, \tag{4.13}$$

[5]This particular function is, in fact, the function you would get if you expressed the total PAR flux density (in μmol$/m^2/s$) measured at the Earth's surface in Champaign, Illinois (per the *weather* data set accompanying BioCro) as a function (taking $t_i = i$) of the number of hours elapsed since midnight on January 1, 2005.

[6]This isn't to say the requisite modules exist. In order to realize this replacement, the user may have to write them!

[7]As an alternative to confining the discussion here to the case where there is only a single module of each type, we could instead make no stipulation about the number of modules and simply substitute the phrase "the direct (differential) module component" in every place where we speak of "the direct (differential) module".

where $\mathbf{W}_i$ is the subspace of $\mathbf{W}$ generated by the variables in the output of module $\mathcal{M}_i$ and $\overline{\mathbf{W}}_i$ is the subspace of $\mathbf{W}$ generated by those inputs to module $\mathcal{M}_i$ that are not in X, K, or D. ($\overline{\mathbf{W}}_i$ is complementary to $\mathbf{W}_i$, with $\mathbf{W}_i \times \overline{\mathbf{W}}_i$ a subspace of $\mathbf{W}$.) We will discuss this further in the section on modularization.

Second, it should be mentioned that the Khalil model allows for the inclusion of an output function

$$\mathbf{y} = \mathbf{h}(t, \mathbf{x}, \mathbf{u}). \tag{4.14}$$

Khalil says that this output vector $\mathbf{y}$ "comprises variables of particular interest in the analysis of dynamical systems …." In the Khalil model, these variables, unlike the variables that comprise $\mathbf{u}$, are not a part of the state equation. They are there for informational purposes only.

The closest analogue to these variables in BioCro are those variables that are outputs of the direct module of the system but are not inputs to the differential module. (An example of such variables in the BioCro library are the *kinetic energy*, *spring energy*, and *total energy* outputs of the harmonic energy module (class `harmonic_energy`). These exist only to give information about the system using this module since (at least of this writing) there are no existing modules that use these as inputs.)

- The *differential module* corresponds to the function $\mathbf{f}$ in equation (4.7). This is the Khalil state equation, but with $\mathbf{u}$ divided into components $\mathbf{k}$, $\mathbf{d}$, and $\mathbf{w}$. The primary constraint on the differential module is that its output variables must all be (in the terminology of Khalil) *state* variables (as determined, you may recall, by which variables are included in the specification of the initial state).

  (BioCro doesn't require that all state variables be included in the differential module outputs: if some state variable x_i is not, it is assumed that $\dot{x}_i = 0$; that is, that component of the state remains constant throughout the life of the system.)

4.3 BioCro's concept of time

In the C++ interface to the BioCro library, there is only one required user-facing time-related variable—namely, the quantity `timestep`, which must be provided as one (and possibly the only) of the parameters when setting up a system. (In the R interface, there are always three additional quantities—`doy`, `hour`, and `time`—which always come into play when we set up or run a system. These are artifacts of certain aspects of the R interface and are subject to revision, and we will mostly ignore them here.)

The `timestep` quantity, however, gives rise to an implicit quantity, the *elapsed time*, that corresponds well with the *time* variable as used in the Khalil and the Giuli-Mazzola models.[8] `timestep`, in fact, denotes the amount of time that elapses between successive values of any of the driver variables.[9]

Time often shows up explicitly in a BioCro system in the form of a specific date and time, and what the value of some driver variable was at that date and time; for example, from the information in the drivers parameter we may able to make assertions such as *the temperature at 3 p.m. on April 15, 2005 was 20.5°C.* But "3 p.m. on April 15, 2005" is not the sort of time with which the Giunti model deals. Times—*durations*—in the Giunti model are members of a monoid, which we can add together to get another time in the monoid. But we can not add *3 p.m. on April 15, 2005* and *7 a.m. on September 22, 2021* in any meaningful way to get some other date-time. We can, however, add durations: we can, for example, look at the state of a system

[8]I shall henceforth refer to the model from Giunti and Mazzola (2012) presented in section 3.3 as simply the *Giunti* model: it is essentially the same model presented fifteen years earlier in Giunti (1997), but generalized to an arbitrary monoid, a generalization we have no need to make use of in BioCro. (This is not entirely true: whereas Giunti (1997) only considers time systems consisting of the integers, the rationals, the reals, or the non-negative members of the same, we allow time systems isomorphic to, but not identical to, the non-negative integers, such as the set of non-negative even integers. But our time systems will always be commutative and linearly ordered: we have no need to consider arbitrary monoids, such as time systems that correspond to cyclic groups, arbitrary n-dimensional vector spaces, or the quaternions under multiplication.)

[9]We could make *elapsed time* an explicit quantity in our systems by writing a differential module with no inputs, one that always returns the value 1 for its one output variable `elapsed_time`. Then, assuming we give it the initial value zero, its value at any time will represent the amount of time that has elapsed since the start of the simulation, in whatever time units `timestep` is in.

one hour after the initial state of that system, and then look at the state two hours later, and the second observation will be three hours after the time corresponding to the system's initial state (since $2 + 1 = 3$).

As hinted above, the `timestep` quantity generates a monoid: if the value of `timestep` is δ, then the members of that monoid are $0, \delta, \delta + \delta, \delta + \delta + \delta, \delta + \delta + \delta + \delta, ...$, ad infinitum.

Of course, in BioCro, we don't let our system simulations run forever, so the set of times dealt with in any given run of a system doesn't really quite form a monoid because if we add δ to the final time point in our simulation, we get a time that is outside the domain of our simulation. Conceptually, we can deal with this problem (of reconciling BioCro's concept of time with that of the Giunti model) by imagining that our system simulations *could* run forever if we let them (and if we had knowledge infinitely far into the future of any driver variables we happened to be using); we imagine that we *could* do this but that instead, we choose to look at the behavior of the system only over some finite period of time.

4.4 BioCro's concept of state

In BioCro, at the level of a module, all input quantities are considered uniformly. There are good reasons for this, reasons that go beyond mere programmatic convenience. For example, the input to some module might in one system be determined mechanistically as the output of some other module; but in a different system, it might come from data observations and thereby be one of the drivers. But the module using that input doesn't care where it comes from.

Once we set up a system, however, each quantity falls neatly into one of four categories: it is either a constant; a driver; a differential quantity; or a derived quantity, that is, a quantity whose value at any given time can be directly computed from the values of other quantities at that same time using some more or less simple formula. (These correspond, respectively, to the subspaces $\mathbf{K}$, $\mathbf{D}$, $\mathbf{X}$, and $\mathbf{W}$ discussed in Sections 4.1 and 4.2 and to the arguments $\mathbf{k}$, $\mathbf{d}$, $\mathbf{x}$, and $\mathbf{w}$ in the state equation in the form given in equation (4.7).)

The uniform treatment of quantities at the module level is reflected in the C++ code used to implement BioCro simulations: each quantity used in a given system is incorporated into a C++ structure of a user-defined type called `state_map`, which maps names of quantities to the value such quantities have at some particular time. This naturally leads to referring to the aggregate of the values of all quantities of the system at some particular point in time as the *state* of the system at that time. This section attempts to reconcile this conception of state with that commonly used in dynamical systems theory, and in particular, with the formulations presented in section 3.

4.4.1 BioCro state and the Giunti model

A BioCro system having drivers but not including a time-like variable in its state does not, in general, conform to the Giunti model.

To see this, let us consider a typical BioCro system in which the driver component consists of the values of a number of weather-related variables over the course of a year, and suppose these variables happen to all have the same values at two different times; for example, suppose the weather at 1 p.m. on April 12, 2008 is exactly the same as the weather at 3 p.m. on May 16, 2008 to the extent *weather* is captured by the attributes in our model. Moreover, suppose our system has what might be a typical array of differential variables—namely, those that describe the state of growth of a plant that is subjected to the environment described by the driver variables in the system.

Consider now two identical states—one, s_1, corresponding to a seedling planted at 1 p.m. on April 12, 2008, and one, s_2, corresponding to an identical seedling planted at 3 p.m. on May 16, 2008. The states are identical because the attributes of the seedlings, described by the differential variables, are identical, and the attributes of the weather, described by the driver variables, are also identical; and because the parameters (being constant) are identical, and the values of the "direct" variables, being functions of the other three components, are also identical. (Recall that we are specifically excluding date and time from our notion of state here.) In other words, $s_1 = s_2$.

Now consider one of the transition functions g^t—say, for example, some function g^u, where u corresponds to a duration of 30 days. Then $g^u(s_1)$ will be the state corresponding the the attributes of the seedling planted on April 12 and its environment one month later, on May 12, 2008. And $g^u(s_2)$ will be the state corresponding the the attributes of the seedling planted on May 16 and its environment approximately one month later, on June 15. Will the weather at 1 p.m. on May 12, 2008 be identical to that at 3 p.m. on June 15, 2008? According to the Giunti model, it should be, since $s_1 = s_2$ implies that state $g^u(s_1)$ equals state $g^u(s_2)$; and if two states are equal, those components of the state that describe the weather should be equal as well.

But we know that something is wrong here, because even if the same weather occurs at two different times, we can't expect the weather patterns going forward to develop in the same way. Moreover, in all likelyhood the identical seedlings planted on April 12 and May 16 will no longer be identical 30 days later because they likely will have been subjected to different weather conditions.

There are two ways (at least) out of this predicament. One is to ensure that the driver component of the state never repeats itself. Any monotonically-increasing driver variable would do the trick, but the most natural way of ensuring no repetitions is probably to include some representation of the time, such as the calendar date and time, Julian date, reduced Julian date, or Unix time as part of the driver component of the state. (The R interface to BioCro in fact requires either both the day-of-year and hour of the day as driver variables or it requires a monotonically-increasing variable called *time*. The C++ interface, however, requires neither of these.)

A second way, one that makes the system formally time-independent, is to modify the driver component in our notion of the state. In this scheme, the driver component of a state is not just an array of values the driver variables happen to have at some particular time. Now, instead, it is an encapsulation of the future of the driver variable values indefinitely far into the future. One way to imagine this, if we are thinking of the drivers as corresponding to the weather, is to think of the driver component of the state at some particular time as a weather prediction giving the weather at that time *and for every future time*, i.e., the weather one day from now, two days from now, and so on; moreover, not just any prediction, but a 100% accurate one. The state now, without having to include the date or time, encapsulates all the information we need to have in order to know what the state will be x amount of time in the future.

I bring this up to show that even in the presence of drivers (*inputs*, in Khalil's terminology), the notion of an autonomous system is possible. And we can have systems that conform to the Giunti model without requiring that states on different dates be distinct. For example, imagine a greenhouse experiment in which the climate conditions in the greenhouse repeat exactly the same pattern from one day to the next. In this system, the driver state at noon—the "weather" prediction for each moment going forward (one hour later, 10 hours later, 10 days later)—will be exactly the same from one day to the next. And so to will the evolution of a generic seedling: the evolution of a seedling planted at noon on one day will be the same as the evolution of an identical seedling planted at noon twenty days later.

In practical terms, however, this is a rather complicated model. The state space will no longer be a Euclidean space, since $\mathbf{D}$, the driver component of the state space, will no longer be the Euclidean space $\mathbf{R}^r$ but will instead be the set $(\mathbf{R}^r)^T$ of all functions $\gamma^{\mathbf{D}} : T \to \mathbf{R}^r$. (Note that the state transition of duration u restricted to the $\mathbf{D}$ component of the state is at least easily defined: if γ is the $\mathbf{D}$ component of some state $\mathbf{s}$, then $g^u(\gamma)$ will be the function defined by the rule $t \mapsto \gamma(u + t)$.)

The upshot is that if we want to keep the Giunti model as a model for BioCro systems, the best plan is to allow time or some proxy for time to be a component variable of the state.

4.4.2 The state space as a manifold

One of the arguments given in the supplementary materials to Lochocki et al. (2022) for considering *all* variables as state variables is that "the division between state and auxiliary variables is arbitrary." Whether or not this is a compelling argument for considering "all quantities equal", this statement is, at least on a formal level, largely true, at least in the case where variables mutually determine one another. As stated at the conclusion of Appendix II in Mesarović and Takahara (1975),

> The starting point for any modeling is the observations and the assumption about the existence of relationships between them. The primary concept of a system ought to be definable just with that much data. Whether such a relationship can be described as a transition in a state space is a point that needs to be proven. Even if this is possible, *a state space is not unique, which indicates the secondary nature of the concept of state* [emphasis mine].

(Here, the authors are presumably using *state* in the less comprehensive sense, where it is distinguished from system inputs and outputs, though conceivably they could simply mean that which attributes we choose to observe and codify into a notion of *state* is not unique; but for the sake of argument, we'll assume they mean at least the former.

But the authors, in fact, hint at an altogether different view of what is meant by the state of a system. In this view, everything that can be observed about a system is encompassed by the system's inputs and outputs: the system is essentially a black box, and how it responds to given inputs at any particular time is not always the same. This is because some unobservable aspects of the system come into play. These unobservables constitute the *state* of the system, and how it responds at any given time to given inputs depends on what *state* it happens to be in. A helpful analogy here might be the notion of a person's *state of mind* as a determinant of how they might react to a particular event.)

However expansive we choose to make our notion of *state*, one thing is clear: if we choose to regard the parameters, drivers, and the relationship between quantities embodied in the direct modules as constraints on the state space of our systems, then *given* that a state lies in this state space, we can fully specify the state using only the values of the so-called *differential* variables (plus time); the values of all of the other quantities can be determined from these. Thus, if the total number of quantites in the system (including time) is n, and the number of differential variables is k, then the state space may be viewed as a $k+1$-dimensional manifold embedded in Euclidean n-space $\mathbf{R}^n$. Put another way, no matter how many variables we use to describe the state of the system, there are still only $k+1$ degrees of freedom: the parameters can only take one value, the time variable determines the values of all the driver variables, and the values of these together with the values of the differential variables determine the values of the remaining variables, the direct variables.

An analogy may help make this clearer: Say we wish to consider all points on earth. If we don't limit ourselves to points on the surface, then we could specify such points with three coordinates—longitude, latitude, and altitude. Large values of *altitude* will correspond to points above the earth's surface, and negative values will usually correspond to points in the earth's interior. We are considering arbitrary points in a three-dimensional space, and so it makes sense that we need all three coordinates to fully specify such a point.

But suppose we now say that we only want to consider points on the earth's surface. Given this constraint, this understanding that we are only going to try to describe points on the surface of the earth, we can get by with only two coordinates: we need only specify the longitude and the latitude. We *could* include the third number, the altitude, as well (provided we know it), but it is now not necessary, because it is understood that the point lies on the earth's surface. If we consider a system whose *state* comprises the location of some given object on the surface of the earth, then the state space *is* that surface, a two-dimensional space embedded in Euclidean 3-space.

Note that in this example, it matters which two coordinates we choose for describing states. Generally, it will not, for example, suffice to know only the altitude and the longitude, since, given some choice of altitude and longitude, there may be many points of various latitudes that match. (There are exceptional cases, of course: if we specify the altitude as 8848.86 (meters), even without having specified a longitude or latitude (let alone both), we know the object is at the top of Mount Everest.[10])

[10]Here, I am using *altitude* to mean "height above sea level", or what is, for points on the earth's surface, usually called *elevation*.

4.5 Modularization in BioCro

As mentioned in Section 4.2, any BioCro system can be replaced by an equivalent system using only a single module of each type. We merely have to write one direct module and one differential module, where each (respectively) combines the effects of all the individual direct and differential modules that were used in the original system.

I use "merely" advisedly here, because one of the main strengths of BioCro is the ability to modularize, so that once we have a wide repertoire of modules to choose from, we can choose and combine them in whatever way is useful, without having to write a new module each time we want to tweak some aspect of the system as a whole.

In this section, we look at this combining of modules on a formal level, delineating the requirements for using two or more modules in place of one and the effects of doing so. We start with the differential module case since it is the simpler of the two.

4.5.1 Modularization of the derivative function

As mentioned in Section 4.2, when a BioCro system uses only a single differential module, that module corresponds to the function $\mathbf{f}$ in Khalil state equation (4.7). We shall henceforth refer to $\mathbf{f}$ as the *derivative* function for the system.

As it turns out, in BioCro, the derivative function never depends on t directly; if there is any temporal dependence in function $\mathbf{f}$, it is always via some driver variable or differential variable. Therefore, we can rewrite equation (4.7) as

$$\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}, \mathbf{k}, \mathbf{d}, \mathbf{w}). \tag{4.15}$$

Thus, $\mathbf{f} : \mathbf{X} \times \mathbf{K} \times \mathbf{D} \times \mathbf{W} \to \mathbf{X}$. From here on out, we shall adopt BioCro's notion of state space and denote it as $\mathbf{S}$, so that

$$\mathbf{S} = \mathbf{X} \times \mathbf{K} \times \mathbf{D} \times \mathbf{W}.$$

$\mathbf{f}$ is then a function $\mathbf{f} : \mathbf{S} \to \mathbf{X}$, and the state equation, which we shall now refer to as the *derivative* equation, is just

$$\dot{\mathbf{x}} = \mathbf{f}(\mathbf{s}), \tag{4.16}$$

where $\mathbf{s}$ denotes a state in the state space $\mathbf{S}$.

In general, we can write any state $\mathbf{s}$ in terms of the coordinate variables describing each of the component spaces; that is,

$$\mathbf{s} = (x_0, x_1, \dots, x_{n-1}, k_0, k_1, \dots, k_{q-1}, d_0, d_1, \dots, d_{r-1}, w_0, w_1, \dots, w_{s-1})$$

where n, q, r, and s are the dimensions of the component spaces $\mathbf{X}$, $\mathbf{K}$, $\mathbf{D}$, and $\mathbf{W}$, respectively.

Before we talk about decomposing the derivative function of a system, we will first describe what we mean by a valid differential module for a BioCro system.

Let $X = \{x_0, x_1, \dots, x_{n-1}\}$ be the set of differential variables of the system, and let

$$S = \{x_0, x_1, \dots, x_{n-1}, k_0, k_1, \dots, k_{q-1}, d_0, d_1, \dots, d_{r-1}, w_0, w_1, \dots, w_{s-1}\}$$

be the set of *all* the coordinate variables needed to specify a state in the state space of the system. Then $\mathcal{M}$ is a valid differential module for the system if the input variables are contained in S and the output variables are contained in X.

Let $\mathbf{M}_{\text{in}}$ be the vector subspace of $\mathbf{S}$ generated by the input variables of $\mathcal{M}$ and let $\mathbf{M}_{\text{out}}$ be the vector subspace of $\mathbf{X}$ generated by the output variables of $\mathcal{M}$. Then the *derivative function* for $\mathcal{M}$ is some function

$$\hat{\mathbf{f}}_{\mathbf{M}} : \mathbf{M}_{\text{in}} \to \mathbf{M}_{\text{out}}.$$

To each such function, we may associate a unique function $\mathbf{f}_{\mathbf{M}} : \mathbf{S} \to \mathbf{X}$ as follows:

Let $\boldsymbol{\pi}$ be the projection of $\mathbf{S}$ onto $\mathbf{M}_{\text{in}}$, and let $\boldsymbol{\iota}$ be the injective function of $\mathbf{M}_{\text{out}}$ into $\mathbf{X}$ that assigns each coordinate in X that is not an output variable of $\mathcal{M}$ the value zero. Then we define

$$\mathbf{f_M} = \boldsymbol{\iota} \circ \hat{\mathbf{f}}_{\mathbf{M}} \circ \boldsymbol{\pi} : \mathbf{S} \xrightarrow{\boldsymbol{\pi}} \mathbf{M}_{\text{in}} \xrightarrow{\hat{\mathbf{f}}_{\mathbf{M}}} \mathbf{M}_{\text{out}} \xrightarrow{\boldsymbol{\iota}} \mathbf{X}.$$

We shall call the function $\mathbf{f_M}$ the *system-complete derivative function for $\mathcal{M}$.*

Now suppose we have a collection $\{\mathcal{M}_1, \mathcal{M}_2, \ldots, \mathcal{M}_m\}$ of differential modules assumed to be consistent with (the rest of) our system, and let $\{\mathbf{f}_{\mathbf{M_1}}, \mathbf{f}_{\mathbf{M_1}}, \ldots, \mathbf{f}_{\mathbf{M_m}}\}$ be their corresponding system-complete derivative functions. Then the *combined derivative function* for $\{\mathcal{M}_1, \mathcal{M}_2, \ldots, \mathcal{M}_m\}$ is the function

$$\mathbf{f} = \sum_{i \in \{1,2,\ldots,m\}} \mathbf{f}_{\mathbf{M_i}},$$

defined by the rule

$$\mathbf{s} \mapsto \sum_{i \in \{1,2,\ldots,m\}} \mathbf{f}_{\mathbf{M_i}}(\mathbf{s}).$$

If $\{\mathcal{M}_1, \mathcal{M}_2, \ldots, \mathcal{M}_m\}$ comprise the differential modules for a system, then $\mathbf{f}$ is that system's derivative function.

We could always write a single differential module $\mathcal{M}$ that has $\mathbf{f}$ as its system-complete derivative function and then use it in place of the collection of modules $\{\mathcal{M}_1, \mathcal{M}_2, \ldots, \mathcal{M}_m\}$ in any system that uses them. But this module will likely combine several mechanistic bio-systems concepts, and one of the strengths of BioCro is the ability to tweak one mechanistic model without having to rewrite multiple modules that might use it.

4.5.2 Decomposing the direct module function

Let $\mathcal{S}$ be a BioCro system, and let $\mathbf{S} = \mathbf{X} \times \mathbf{K} \times \mathbf{D} \times \mathbf{W}$ be the state space of $\mathcal{S}$. As mentioned in Section 4.2, the direct module component of a BioCro system $\mathcal{S}$ corresponds to a function

$$\gamma^{\mathbf{W}} : \mathbf{X} \times \mathbf{K} \times \mathbf{D} \to \mathbf{W} \tag{4.17}$$

that determines the value of the "direct variable" component of a state from the value of the other components. For convenience in what follows, we shall write $\mathbf{H}$ to abbreviate the cross product $\mathbf{X} \times \mathbf{K} \times \mathbf{D}$. Thus we may write (4.17) as

$$\gamma^{\mathbf{W}} : \mathbf{H} \to \mathbf{W}.$$

In general, however, the direct module component of a system will be subdivided into two or more submodules, and, as also mentioned in Section 4.2, when a system has more than one direct module, each constituent module $\mathcal{M}_i$ corresponds to a function

$$\gamma^{\mathbf{W}_i} : \mathbf{X} \times \mathbf{K} \times \mathbf{D} \times \overline{\mathbf{W}}_i \to \mathbf{W}_i, \tag{4.18}$$

or, using the abbreviation used above,

$$\gamma^{\mathbf{W}_i} : \mathbf{H} \times \overline{\mathbf{W}}_i \to \mathbf{W}_i. \tag{4.19}$$

Letting H denote the set of variables corresponding to $\mathbf{H}$ and denoting the input and output variables of module $\mathcal{M}_i$ as $\mathbf{In}\,\mathcal{M}_i$ and $\mathbf{Out}\,\mathcal{M}_i$ respectively, we can write

$$\mathbf{W}_i = \mathbf{R}^{\mathbf{Out}\,\mathcal{M}_i}$$

and

$$\overline{\mathbf{W}}_i = \mathbf{R}^{\mathbf{In}\,\mathcal{M}_i \setminus H},$$

where $\mathbf{In}\,\mathcal{M}_i$ and $\mathbf{Out}\,\mathcal{M}_i$ are disjoint, since direct modules never share inputs and outputs. Note that it may be the case that $\mathbf{In}\,\mathcal{M}_i \subseteq H$; in this case $\overline{\mathbf{W}}_i$ is 0-dimensional and 4.19 reduces to

$$\gamma^{\mathbf{W}_i} : \mathbf{H} \to \mathbf{W}_i.$$

H corresponds to the union of all the differential variables, parameters, and driver variables of the system. Usually, it will be the case that any given direct module in a system will not use all of the variables in H: not all of the variables in H will affect the value of the output, nor will they even be listed in the list returned by the module's `get_inputs()` function. But they are all always *potentially* available for use by any direct module, and in what follows, it will be convenient to assume that the direct module inputs include all the variables of H; that is, $\mathbf{In}\,\mathcal{M}_i \supseteq H$, for all direct modules $\mathcal{M}_i$. This way, the only thing that changes about the domain of the module function between various direct modules is the $\overline{\mathbf{W}}_i$ component of $\mathbf{H} \times \overline{\mathbf{W}}_i$. This will simplify the exposition of what follows.

(To take a simple example of formal dependence versus actual dependence, consider a two-variable function $f(x, y)$ defined by the rule $(x, y) \mapsto x^2$. Formally, this is a function of two variables x and y. But the value of the function never actually depends on the value of y.)

The ordered sum of two direct modules Suppose that $\mathcal{M}_i$ and $\mathcal{M}_j$ are two direct modules having disjoint sets of output variables (that is, $\mathbf{Out}\,\mathcal{M}_i \cap \mathbf{Out}\,\mathcal{M}_j = \emptyset$), and suppose also that none of the outputs of $\mathcal{M}_j$ are inputs for $\mathcal{M}_i$; that is, $\mathbf{In}\,\mathcal{M}_i \cap \mathbf{Out}\,\mathcal{M}_j = \emptyset$. Let f and g be their corresponding functions. For convenience, we put

$$\begin{aligned} A &= \mathbf{In}\,\mathcal{M}_i \\ B &= \mathbf{Out}\,\mathcal{M}_i \\ C &= \mathbf{In}\,\mathcal{M}_j \end{aligned}$$

and

$$D = \mathbf{Out}\,\mathcal{M}_j,$$

so that

$$f : \mathbf{R}^A \to \mathbf{R}^B$$

and

$$g : \mathbf{R}^C \to \mathbf{R}^D,$$

with $A \cap B = C \cap D = A \cap D = B \cap D = \emptyset$.

Then we define $\mathcal{M}_i + \mathcal{M}_j$, the *ordered sum of $\mathcal{M}_i$ and $\mathcal{M}_j$*, to be the direct module whose corresponding function

$$f * g : \mathbf{R}^{A \cup (C \setminus B)} \to \mathbf{R}^{B \cup D} \tag{4.20}$$

is defined by

$$f * g = (f \circ \pi^{A\cup(C\setminus B)\to A}) \cup (g \circ (\pi^{A\cup(C\setminus B)\to C\setminus B} \cup (\pi^{B\to C\cap B} \circ f \circ \pi^{A\cup(C\setminus B)\to A}))). \tag{4.21}$$

(Note that if $B \cap C = \emptyset$, then $C \setminus B = C$, and 4.21 reduces to

$$f * g = (f \circ \pi^{A\cup C\to A}) \cup (g \circ (\pi^{A\cup C\to C})). \tag{4.22}$$

In this case, the ordering is immaterial, and $\mathcal{M}_j + \mathcal{M}_i = \mathcal{M}_i + \mathcal{M}_j$, with $g * f = f * g$.)

Recalling that the inputs and outputs of a direct module function must be disjoint, we can check that this is indeed the case for the sum. First we note that whenever we can take the ordered sum of two modules $\mathcal{M}_i$ and $\mathcal{M}_j$,

$$\mathbf{In}(\mathcal{M}_i + \mathcal{M}_j) = \mathbf{In}\,\mathcal{M}_i \cup (\mathbf{In}\,M_j \setminus \mathbf{Out}\,\mathcal{M}_i) \tag{4.23}$$

and

$$\mathbf{Out}(\mathcal{M}_i + \mathcal{M}_j) = \mathbf{Out}\,\mathcal{M}_i \cup \mathbf{Out}\,\mathcal{M}_j. \tag{4.24}$$

(The fact that the set of inputs for the ordered sum is $\mathbf{In}\,\mathcal{M}_i \cup (\mathbf{In}\,M_j \setminus \mathbf{Out}\,\mathcal{M}_i)$ is readily apparent from 4.20 and 4.21: the domain of $f * g$ being $\mathbf{R}^{A\cup(C\setminus B)}$ corresponds to the inputs for the corresponding module being $A \cup (C \setminus B)$, which is just $\mathbf{In}\,\mathcal{M}_i \cup (\mathbf{In}\,M_j \setminus \mathbf{Out}\,\mathcal{M}_i)$. Similarly for the outputs.)

Using this, we then have that

$$\begin{aligned}
\mathbf{In}(\mathcal{M}_i + \mathcal{M}_j) \cap \mathbf{Out}(\mathcal{M}_i + \mathcal{M}_j) &= (\mathbf{In}\,\mathcal{M}_i \cup (\mathbf{In}\,M_j \setminus \mathbf{Out}\,\mathcal{M}_i)) \cap (\mathbf{Out}\,\mathcal{M}_i \cup \mathbf{Out}\,\mathcal{M}_j) && \text{by 4.23 and 4.24} \\
&= (\mathbf{In}\,\mathcal{M}_i \cap \mathbf{Out}\,\mathcal{M}_i) \\
&\quad \cup (\mathbf{In}\,\mathcal{M}_i \cap \mathbf{Out}\,\mathcal{M}_j) \\
&\quad \cup ((\mathbf{In}\,M_j \setminus \mathbf{Out}\,\mathcal{M}_i) \cap \mathbf{Out}\,\mathcal{M}_i) \\
&\quad \cup ((\mathbf{In}\,M_j \setminus \mathbf{Out}\,\mathcal{M}_i) \cap \mathbf{Out}\,\mathcal{M}_j) && \text{by distributivity of } \cap \text{ over } \cup \\
&= \emptyset \cup \emptyset \cup \emptyset \cup \emptyset \\
&= \emptyset
\end{aligned}$$

That each of the intersections in the distributive expansion is empty is easily verified: $\mathbf{In}\,\mathcal{M}_i \cap \mathbf{Out}\,\mathcal{M}_i$ and $(\mathbf{In}\,M_j \setminus \mathbf{Out}\,\mathcal{M}_i) \cap \mathbf{Out}\,\mathcal{M}_j)$ are both empty as a consequence of direct modules having non-overlapping inputs and outputs. $(\mathbf{In}\,M_j \setminus \mathbf{Out}\,\mathcal{M}_i) \cap \mathbf{Out}\,\mathcal{M}_i$ must be empty since a member of $\mathbf{In}\,M_J$ that is *not* in $\mathbf{Out}\,\mathcal{M}_i$ can't also be *in* $\mathbf{Out}\,\mathcal{M}_i$. Finally, $\mathbf{In}\,\mathcal{M}_i \cap \mathbf{Out}\,\mathcal{M}_j = \emptyset$ was a stipulation made when defining the ordered sum of $\mathcal{M}_i$ and $\mathcal{M}_j$.

Equation 4.21 perhaps requires a little explication in order to be comprehended.

Suppose we are given some value x in $\mathbf{R}^{A\cup(C\setminus B)}$, the domain of $f * g$. We can describe $(f * g)(x) \in \mathbf{R}^{B\cup D}$ by describing the way to compute how $(f * g)(x)$ maps each $y \in B \cup D$ into $\mathbf{R}$.

First suppose $y \in B$. Then we need only look at the first component in the union on the right-hand side of 4.21—namely, $f \circ \pi^{A\cup(C\setminus B)\to A}$. The projection $\pi^{A\cup(C\setminus B)\to A}(x)$ of x to $\mathbf{R}^A$ tells us that we need consider only the coordinates of x that correspond to members of A. Once we have a vector in $\mathbf{R}^A$, we can apply the function f to obtain a value in $\mathbf{R}^B$. This is all we need, since y is in B.

Now suppose $y \in D$. Here we need to look at the somewhat more complicated second component of the right-hand side of 4.21, that is, $g \circ (\pi^{A\cup(C\setminus B)\to C\setminus B} \cup (\pi^{B\to C\cap B} \circ f \circ \pi^{A\cup(C\setminus B)\to A}))$. Since $g : \mathbf{R}^C \to \mathbf{R}^D$, we need to feed g some value in $\mathbf{R}^C$ to obtain a mapping to $\mathbf{R}$ of values (such as y) in D. But x is in $\mathbf{R}^{A\cup(C\setminus B)}$, so x only tells how values in C that aren't also in B are mapped. The mapping for these values corresponds to the projection $\pi^{A\cup(C\setminus B)\to C\setminus B}(x)$, a value in $\mathbf{R}^{C\setminus B}$. To find the portion of the mapping we need that belongs to $\mathbf{R}^{C\cap B}$, we look at $\pi^{B\to C\cap B} \circ f \circ \pi^{A\cup(C\setminus B)\to A}$. As we have just seen, $f \circ \pi^{A\cup(C\setminus B)\to A}$ maps x to a member of $\mathbf{R}^B$. Then we can apply the projection $\pi^{B\to C\cap B}$ to obtain a member of $\mathbf{R}^{C\cap B}$. Taking the union of the components in $\mathbf{R}^{C\setminus B}$ and $\mathbf{R}^{C\cap B}$ gives us a value in $\mathbf{R}^C$, to which we can apply function g. The result is a function in $\mathbf{R}^D$ telling how all values (such as y) in D are mapped.

This is all perhaps easier to understand if we think in terms of module inputs and outputs: Given mappings for all values in A and all values in C that aren't in B, we can obtain mappings for all values in B and D as follows: Since we know all the inputs A to module $\mathcal{M}_i$, we can use the module to obtain all the outputs B. Now we know all the inputs C to module $\mathcal{M}_j$—both those that *aren't* in B (which were given at the outset), and those that *are* in B (which were obtained by applying module $\mathcal{M}_i$). This yields all the outputs D of module $\mathcal{M}_j$.

General ordered sum We now generalize the notion of an ordered sum of two direct modules to the ordered sum of any finite number of direct modules.

Suppose we have an ordered collection $(\mathcal{M}_1, \mathcal{M}_2, \dots, \mathcal{M}_n)$ of direct modules. As is the case with all direct modules, $\mathbf{In}\,\mathcal{M}_i \cap \mathbf{Out}\,\mathcal{M}_i = \emptyset$ for all i. Further, assume that $\mathbf{Out}\,\mathcal{M}_i \cup \mathbf{Out}\,\mathcal{M}_j = \emptyset$ for all $i \neq j$, and that $\mathbf{In}\,\mathcal{M}_i \cap \mathbf{Out}\,\mathcal{M}_j = \emptyset$ whenever $i < j$. Then we define the ordered sum $\sum_{i=1}^{n} \mathcal{M}_i$ recursively as follows:

$$\begin{aligned} \sum_{i=1}^{k} \mathcal{M}_i &= M_1 && \text{for } k = 1 \\ \sum_{i=1}^{k} \mathcal{M}_i &= \sum_{i=1}^{k-1} \mathcal{M}_i + \mathcal{M}_k && \text{for } 1 < k \leq n \end{aligned} \tag{4.25}$$

Things are not quite as simple as this, however, since we must show that the ordered sum given on the right-hand side of 4.25 is always defined. Specifically, we must show that

$$\mathbf{Out} \sum_{i=1}^{k-1} \mathcal{M}_i \cap \mathbf{Out}\,\mathcal{M}_k = \emptyset \tag{4.26}$$

and

$$\mathbf{In} \sum_{i=1}^{k-1} \mathcal{M}_i \cap \mathbf{Out}\,\mathcal{M}_k = \emptyset. \tag{4.27}$$

But it easily follows by induction from equation 4.24 that

$$\mathbf{Out} \sum_{i=1}^{k-1} \mathcal{M}_i = \bigcup_{i=1}^{k-1} \mathbf{Out}\,\mathcal{M}_i,$$

and 4.26 easily follows from this, the distributivity of $\cap$ over $\cup$, and the assumption that the outputs of the modules are pairwise disjoint.

To prove 4.27, we first observe that it follows immediately from 4.23 that $\mathbf{In}(\mathcal{M}_i + \mathcal{M}_j) \subseteq \mathbf{In}\,\mathcal{M}_i \cup \mathbf{In}\,M_j$, and from this it is easy to show by induction that

$$\mathbf{In} \sum_{i=1}^{k-1} \mathcal{M}_i \subseteq \bigcup_{i=1}^{k-1} \mathbf{In}\,\mathcal{M}_i. \tag{4.28}$$

Since a stipulation in defining the ordered sum of modules was that output of each module in the ordered collection is disjoint from the inputs of each module occuring earlier in the ordering, in light of the distributivity of $\cap$ over $\cup$, the desired result 4.27 immediately follows.

Appendix: Degenerate BioCro systems

This appendix is meant to demonstrate certain edge cases and "off-label" uses of BioCro systems. All of these systems are set up using the R interface. A similar set of systems that use the C++ library directly could be written in C++.

A minimal system

This system contains the absolute minimum number of quantities. Since it has only a single time point, `timestep` is present only to satisfy a formal requirement of the validity checker; it is otherwise meaningless.

A formal requirement of the R interface (but not of the C++ interface) is that the set of driver variables either contains `time` or contains both `doy` and `hour`. All three variables show up in the output.

```
library(BioCro)
run_biocro(parameters = list(timestep=1), drivers = data.frame(time=45.625))
```

```
##   doy hour ncalls   time
## 1  45   15      1 45.625
```

```
run_biocro(parameters = list(timestep=1), drivers = data.frame(doy=80, hour=14.25))
```

```
##   doy  hour ncalls     time
## 1  80 14.25      1 80.59375
```

Note that `ncalls` always shows up in the output data frame, even though it is constant and even though it is not a system variable.

Note also that if `time` is a driver, it dominates: `doy` and `hour` (if present) are overwritten. If `time` is not present, both `doy` and `hour` must be; if only one is, we get obscure error:

```
  Error in floor(result$time) :
    non-numeric argument to mathematical function
```

A system having a differential variable but no differential module

As noted in Section 4.2, it is the `initial_values` parameter that determines which variables are differential variables. Usually, each differential variable will be an output of one or more differential modules, but this is not required. Differential variables that are *not* in the output of any differential module are assumed to have a derivative of zero; that is, they are constant. This system exhibits such a case.

```
run_biocro(initial_values = list(x = 52),
           parameters = list(timestep=1),
           drivers = data.frame(time=0:4))
```

```
##   doy hour ncalls time  x
## 1   0    0      5    0 52
## 2   1    0      5    1 52
## 3   2    0      5    2 52
## 4   3    0      5    3 52
## 5   4    0      5    4 52
```

An *off-label* use of `run_biocro`

Here is an example of what might be called an "off-label" use of a BioCro system. This system really doesn't deserve to be called a dynamical system at all. Although the *drivers* parameter contains five rows of temporal and spacial data (each row specifies a time and a place), the rows have no inherent relationship to one another: they do not represent any sort of evolution of a system over time. The times specified by the rows aren't even in chronological order: although the `timestep` variable is *supposed* to indicate the temporal relationship between successive rows of the `drivers` parameter value, this is a convention only, and it is not enforced.

Nevertheless, this system is useful: it uses the `BioCro:solar_position_michalsky` module to compute the cosine of the zenith angle of the sun at noon in various terrestrial locations on various days of the year. We could have gotten the same information using five calls to `run_biocro` with drivers having a single row, but doing it in one call is more convenient.

```
result <- run_biocro(parameters = list(timestep=1),
                     drivers = data.frame(doy = c(355, 172, 80, 80, 80),
                                          hour = 12,
                                          time_zone_offset = -6,
                                          year = 2022,
                                          lat = c(40, 40, 40, 0, 89),
                                          longitude = -88),
                     direct_module_names = 'BioCro:solar_position_michalsky')
result[c('lat', 'longitude', 'doy', 'hour', 'cosine_zenith_angle')]
```

```
##   lat longitude doy hour cosine_zenith_angle
## 1  40       -88 355   12          0.44655908
## 2  40       -88 172   12          0.95824629
## 3  40       -88  80   12          0.77093280
## 4   0       -88  80   12          0.99996308
## 5  89       -88  80   12          0.02509952
```

A system having only drivers (and the obligatory `timestep` parameter)

Like the minimal system shown in the first example, this system has no differential variables and no modules. But the drivers include some driver variables that aren't time related. Like all systems not having any modules, it doesn't really *do* anything.

```
result <- run_biocro(parameters = list(timestep=1),
                     drivers = weather$`2005`[1000:1010,])
result[c('year', 'doy', 'hour', 'precip', 'rh', 'solar', 'temp', 'windspeed')]
```

```
##    year doy hour     precip      rh   solar  temp windspeed
## 1  2005  42   15 0.01058333 0.73690 755.964 4.530     7.475
## 2  2005  42   16 0.01058333 0.70095 421.429 5.085     7.445
## 3  2005  42   17 0.01058333 0.73080 101.775 4.375     5.935
## 4  2005  42   18 0.01058333 0.80135   1.081 2.465     4.785
## 5  2005  42   19 0.01058333 0.83150   0.345 1.650     4.115
## 6  2005  42   20 0.01058333 0.84020   0.299 1.210     3.565
## 7  2005  42   21 0.01058333 0.87250   0.115 0.635     3.255
## 8  2005  42   22 0.01058333 0.86045   0.184 0.550     3.760
## 9  2005  42   23 0.01058333 0.84600   0.138 0.890     4.815
## 10 2005  43    0 0.00000000 0.82435   0.138 1.250     4.955
## 11 2005  43    1 0.00000000 0.83260   0.414 1.025     5.675
```

The weather information this run displays could just as easily be displayed using

```
weather$`2005`[1000:1010,
               c('year', 'doy', 'hour', 'precip', 'rh', 'solar', 'temp', 'windspeed')]
```

References

Barreira, Luís. 2019. *Dynamical Systems by Example*. 1st ed. 2019. Problem Books in Mathematics. Cham: Springer International Publishing.

Dale, Andrew I. 1995. *Philosophical Essay on Probabilities / Pierre-Simon Laplace.* 1st ed. 1995. Sources in the History of Mathematics and Physical Sciences; 13. New York: Springer.

Deutsch, Andreas. 2005. *Cellular Automaton Modeling of Biological Pattern Formation: Characterization, Applications, and Analysis.* Modeling and Simulation in Science, Engineering and Technology. Boston: Birkhäuser.

Giunti, Marco. 1997. *Computation, Dynamics, and Cognition.* Oxford University Press.

Giunti, Marco, and Claudio Mazzola. 2012. "Dynamical Systems on Monoids: Toward a General Theory of Deterministic Systems and Motion." In *Methods, Models, Simulations & Approaches Towards A General Theory of Change - Proceedings of the Fifth National Conference of the Italian Systems Society*, 173–85. https://www.researchgate.net/publication/272943599_Dynamical_Systems_on_Monoids_Toward_a_General_Theory_of_Deterministic_Systems_and_Motion.

Khalil, Hassan K. 2002. *Nonlinear Systems.* 3rd ed. Upper Saddle River, N.J: Prentice Hall.

Lochocki, Edward B, Scott Rohde, Deepak Jaiswal, Megan L Matthews, Fernando Miguez, Stephen P Long, and Justin M McGrath. 2022. "BioCro II: a software package for modular crop growth simulations." *In Silico Plants* 4 (1). https://doi.org/10.1093/insilicoplants/diac003.

Mesarović, Mihajlo D., and Yasuhiko Takahara. 1975. *General Systems Theory: Mathematical Foundations.* Mathematics in Science and Engineering, v. 113. New York: Academic Press.

Vaught, Robert L. 1985. *Set Theory: An Introduction.* Boston: Birkhäuser.