Table of Contents

Chapter 3: Parts and Elementary Composition

1. Introduction

A natural cell is an amazing machine with complex behaviors driven by autonomous genetic programs, environmental sensing, and sophisticated processing of these signals into mechanochemical responses. Underlying all of these processes are chemical interactions among canonical sets of biomolecules: DNA, RNA, proteins, a surprising diversity of small molecules and ions, and molecular superstructures, such as membranes, structural elements such as microtubules, and large-scale DNA structures such as chromosomes. Biologists have discovered groups of these chemical interactions that define key cellular functions, including gene expression, signal transduction, metabolism, and secretion. These groups of functions, and the molecules that carry them out, provide a conceptual basis set from which synthetic biologists can configure novel function. One of the tasks of synthetic biology is to develop the informal definitions of encapsulated function used in biology into formal, reusable, reconfigurable "parts" that can be used for synthetic design.

By definition, a part is simply a system element with defined boundaries and behavior. There is no single way to define a biological part, because there are many ways to decompose complex cellular systems into lower-level elements. As a result, there is flexibility in defining the most primitive components available as substrates for bottom-up design. However, for synthetic biology to mature into a more industrial engineering discipline, we must develop a consistent definition for a biological "part."

Our choices about these primitive components, and the decompositions we use, are ultimately driven both by the physical nature of the systems and by our engineering needs. Because such decompositions are in part driven by human choice and engineering need, they are often called "abstractions". Engineers attempt to define abstraction "layers," in which parts are defined at similar levels of generalization away from specific details of implementation. A good abstraction layer allows reuse of parts both by distilling a useful function into its key properties and providing a consistent mathematical framework for modeling systems composed by parts at this level of description. The most primitive layer of abstraction is usually something closest to a physical description of a specific implementation. At this elementary level, the design can be turned into a specification to manufacture the device. For synthetic biology to progress, we must properly define effective layers of abstraction that allow “high-level” specifications to be automatically compiled into primitive layers necessary for manufacture, and specifying a consistent definition of a biological “part” is at the heart of each effective abstraction layer.

It is practically tempting to define biological parts based on their physical instantiations. In the laboratory, the “parts” with which one directly works are generally DNA fragments that encode “functions”; thus, one can imagine defining a part as a DNA sequence. However, this definition does not easily capture the intended behavior of the system or the implication of the composition among the parts, at least partly because DNA itself does not explicitly encode function. While the DNA sequence may be a key piece of information for manufacture, it is not sufficient (or even often necessary) for functional design.

Another option is thinking of the different steps required for the desired design as the “part” of the system. Let’s consider the simple case of designing a system to express a protein at a given level in a specified host. We might break the system into parts that control transcription initiation, mRNA production, translation of mRNA into protein, and protein and mRNA degradation. In stating this decomposition, we have made choices about the “separable” functions of the system and the level of detail at which we are working. One might wonder, however, if it is truly possible to separate these functions. For example, efficient translation in bacteria is correlated with increased mRNA stability, since the presence of ribosomes on the transcript can protect it from RNA cleaving enzymes; thus, at some level of abstraction mRNA translation and degradation cannot be considered in isolation. We have also made assumptions about how these parts connect. For example, it makes sense that the transcription initiation part would “feed” polymerases to the mRNA production part, which would output mRNA, but until we formally state this fact, the possible interaction between the two parts is implicit. Additional questions are also left unresolved. Is the input a concentration of initiated polymerase or a rate of initiated polymerase production? Is the output a concentration of mRNA or a rate of production? Without a precise definition, it is unclear how a set of parts encodes a system’s behavior.

This input/output specification is a description of a more general concept: dependency. In a particular design, one part may depend on input from another, creating a functional dependency. Thus, our gene expression “part” above might depend on the input from a transcription factor produced by another part in the system. There is also the possibility of resource dependency, in which parts rely on a common pool of host resources, like polymerase, nucleic acids, ribosomes, amino acids, and more. In addition, parts may also have a genetic context dependency. For example, the activity of the part might change depending on where it is inserted into the host genome or on the environmental temperature. Specifically, when a synthetic sequence is inserted somewhere downstream of an endogenous promoter, activity by the endogenous promoter may alter gene expression of the synthetic construct in undesirable ways, or, in mammalian cells, epigenetic modifications associated with the endogenous promoter silences expression from the synthetic promoter. There are a number of types of such context dependencies which are an unavoidable consequence of the physical instantiation of the part.

Some dependencies may be in line with a design's desired input/output relationships among synthetic parts and the host, but other undesirable dependencies may arise from the biophysics of the processes involved. Generally we label such undesired dependencies cross-talk and parasitic interactions. Cross-talk arises when the use of a common resource (or functional input) by two or more parts induces an undesired interaction. A well-defined example of this occurs when protein production from one part saturates a common protease, thus slowing the degradation of protein from another part. Parasitic interactions occur when a part possesses an activity other than the desired one that causes unwanted system behavior. Examples of parasites include: lack of complete specificity of a transcription factor for a DNA sequence, allowing it to bind to an unexpected region in the host chromosome and change expression of a gene not considered in the original design; or promiscuous/multifunctional enzymes that possess activities other than those needed for a desired metabolic pathway. (In *E. coli* there are over 100 known multifunctional enzymes, some catalyzing as many as nine distinct reactions.) Later, we will discuss a third dependency type, retroactivity, in which even desired interactions induce unwanted side-effects in circuit function.

With these shortcomings in mind, we must carefully consider our formal definition for a biological part in synthetic biology. Below we introduce and discuss a functional definition of a part, in which the part is actually a process, with specified inputs and outputs, rather than a physical entity. We will argue later that this formal definition will also drive desiderata for how we choose, molecularly design, and characterize the physical implementations of biological function, in the form of classes of protein or RNA regulatory elements, enzymes, and larger-scale devices such as a "nitrogen fixation device".

2. Defining Design Elements

There are a number of reasons we have decided to use a functional definition of a part. First, as designers we think in terms of connecting functional units to obtain a goal. The physical implementation (for example, the specific protein that will carry out the function) is initially less important than its overall behavior. Thus, for design purposes it is best to define the "part" as the function, not the molecule. Second, properly defined functions have formal specifications regarding input and output, which facilitate the design process. Finally, defining parts as functions, rather than as molecules, allows a hierarchical view of design, in which complex functions are compositions of more elementary functions, with the formal, explicit inputs and outputs.

State and State Variables

One way we can think about our definition of a “part” is that they are processes that change the **state** of our system—in this case a cell. The state is defined by the **state variables**, such as the concentration and physical distribution of various molecules, the temperature, and the volume of the system. The state is a static snapshot of the system, defined by the status of the state variables at a given time, but does not reflect the system’s dynamics.

The goal of any design is generally to move a system from an initial state through a programmed sequence of subsequent states via changes to specific state variables. Whether or not a transition between two states is permitted depends on both the current state and external inputs. In other words, there is a *function* that uses the system’s current state and the external variables as inputs to determine whether a specific transition is allowed to occur, and what the resulting outputs will be. We define these functions as **parts**.

A Part is a Process

In our definition, parts are processes that take a subset of state variables as input, like concentration, compartment volume, and temperature, and produce a set of *rates of change* for these state variables as output. These rates of change determine how the state will change over time, based on the parts’ current inputs.

It is important to emphasize here the the parts’ outputs are the rates of change of the state variables, not the physical molecule(s) that actually perform the transformation. For example, consider the following process:

$A + E \rightarrow B + E$

The part is the actual process, the *conversion* of A to B - *not* any or all of the three molecules involved in the process. The part encoding the conversion takes as input the concentrations of A and E and outputs the rates of change of the concentrations of A, B, and E (Figure 1, 2).

Parts can be very simple or quite complex. A single part may actually be composed of many elementary processes, depending on the level of abstraction in the design at any given point. For example, when a cell moves from environment 1 to environment 2, a state change may be triggered in which many genes alter their expression. There is a large “expression network” part that encodes the mechanisms of this change, but there are also many simultaneous individual parts at work. We will need to consider parts at different levels of granularity at different points in the design process.

Compositors Integrate Parts

In most cases, multiple parts interact with a single molecule. **Compositors** must integrate the outputs of individual parts to determine how the overall state of the system should change. We define compositors as the aggregators of the effects of these parts. They are a means to track the implications of multiple processes having their own instantaneous rates.

There is one compositor for each state variable of the system (for example, for each type of molecule). Multiple parts may act on the same state variable, and thus on the same compositor. A compositor takes as input the rates from possibly many parts and produces a change in the relevant state variable. Thus, the state variable is, in a sense, the output of the compositor. The set of outputs from the compositors is the cellular state.

To illustrate, we will return to the simple reaction introduced above, $A + E_1 \rightarrow E_1 + B$. Again, the part is the process: the conversion of A to B as catalyzed by E_1 (Figure 1). The part's inputs are the states of *A* and *E_1 (*their concentrations), and the part's output is the rate of change for each molecule involved in the process. Each of these rates in turn acts as an input for the relevant compositor, which implements the change as defined by the part and produces the modified state as output (Figure 1, 2). There must be compositors for *A*, *B*, and *E1*, even though the state of this last variable isn't changed by this part.

The parts' outputs have a parameter, k_1, representing the intrinsic rate of a reaction, in addition to the concentrations of the relevant molecules. In a physical sense, this means that parts' outputs include a parameter defined by the intrinsic activities of the molecules ultimately responsible for the physical transformation.

Now consider adding a second related process to the same system. (Figure 3). Here we added a new enzyme-catalyzed reaction that converts two B molecules to one C molecule. The inputs to this new part are the concentrations of *B* and *E_2* and the outputs are the rates of change of *B*, *C and E_2*.

Elements of Design	**Definitions**	**Explanation**
Parts	$A + E_1 \rightarrow E_1 + B$	Our system has one part: the enzymatic conversion of A to B by E_1
Rates of change	$\nu_A^1 = -k_1[E_1][A]$, $\nu_B^1 = k_1[E_1][A]$, $\nu_{E_1}^1 = 0$	These are the formal outputs of the part above. This one part implies that three state variables might have to change over time based on, as specified by the reaction mechanism. Here we represent the rates of change as deterministic, mass-action rate laws. The superscript on the rate variable represents the part number associated with this rate; the superscript is the state variable affected by this rate. Note that, because E1 is an enzyme, its rate of change is 0.
Compositors	$\frac{d[A]}{dt} = \nu_A^1$ $\frac{d[B]}{dt} = \nu_B^1$ $\frac{d[E_1]}{dt} = \nu_{E_1}^1$	A compositor is the sum of all rates of change of state variables of the system. In this case there is only one process, so each compositor has a single term, and the result is a system of ordinary differential equations.
States	$[A],[B],[E_1]$	The state of the system, in this case, is the concentration of all chemical species on which part behavior depends. Ultimately, there could be other sorts of state variables such as position, volume, or temperature that might be affected by a part a designer chooses to use. [A] and [E1] are inputs to the part above.

Figure 1: Parts, change, compositors, and states for the system with one part.

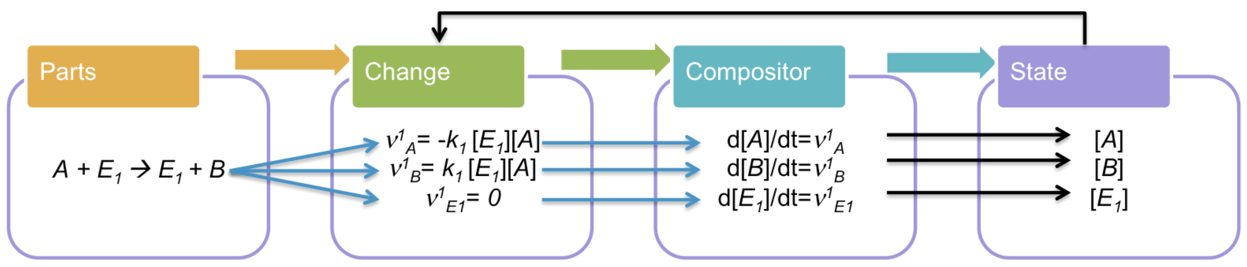


Figure 2. Parts, change, compositors, and states for the system with one part. This figure shows the flow of "information" in the wiring of our system. Once a designer has chosen to use this part the associated "Changes" are implied. The values of the changes are the outputs of the part and the inputs are, in this case, the chemical concentrations of the individual chemical species (A and E_1).

In addition, the two parts are not independent; an interaction has been induced. The first part, "$A + E_1 \rightarrow E_1 + B$", induces changes in [B], which is an input to the second part, "$2\,B + E_2 \rightarrow E_2 + C$". This is reflected in a change to the B compositor (compare Figures 1 & 2 and 3 & 4) which now contains two terms, one derived from each elementary part. The "wiring" of our system, then, explicitly appears as part outputs that appear together in compositors.

Elements of Design	**Definitions**	**Explanation**
Parts	$A + E_1 \rightarrow E_1 + B$ $2\,B + E_2 \rightarrow E_2 + C$	Now there are two parts. An “output” of the first, B, is an input to the second.
Rates of change	$\nu_A^1 = -k_1[E_1][A]$, $\nu_B^1 = k_1[E_1][A]$, $\nu_B^2 = -2\,k_2[E_2][B]^2$, $\nu_C^2 = k_2[E_2][B]^2$, $\nu_{E_1}^1 = 0$, $\nu_{E_2}^2 = 0$	Now we have five state variations, A, B, C, E_1, and E_2 . The first part affects A, B, and E_1. The second part affects B, C, and E_2. E_1 and E_2 are catalysts, so they do not change state when these process execute. Thus their rates of change are zero, but because the parts are dependent on the state of these enzymes, we must keep track of them.
Compositors	$\frac{d[A]}{dt} = \nu_A^1$ $\frac{d[B]}{dt} = \nu_B^1 + \nu_B^2$ $\frac{d[C]}{dt} = \nu_C^2$ $\frac{d[E_1]}{dt} = \nu_{E_1}^1$ $\frac{d[E_2]}{dt} = \nu_{E_2}^2$	The compositors are similar to those in Figure 1. However, now two parts affect the state of B, and therefore its compositor has two terms, one from each part.

Figure 3: Parts, change, compositors, and states for the system with two parts

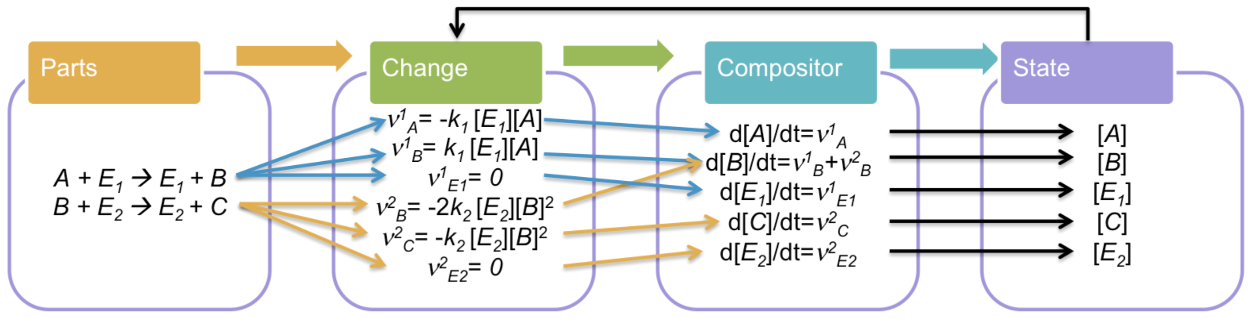


Figure 4: Parts, change, compositors, and states for the system with two parts. This figure shows the flow of "information" in the wiring of our system. Once a designer has chosen to use these two parts the associated "Changes" are implied. The values of the changes are the outputs of the two parts (1 and 2 in the superscripts of the rates) and the inputs are, in this case, the chemical concentrations of the individual chemical species (*A* and E_1 for the first part, *B* and E_2 for the second).

Again we see that each type of molecule has its own compositor, and the single compositor for each molecule includes the rates of change from all relevant parts. In essence, they are the integrator over time of the rates. The output from the compositors represents the state of the system: in this case, the vector of concentrations {[A],[B],[C],[E1], [E2]}.

In summary:

1) The inputs to "parts" are state variables of the system.
2) The output of parts are instantaneous rates of changes of these variables.
3) The inputs to compositors are the outputs of parts.
3) The outputs compositors are instantaneous values of the state variables.

The effect of all this is that, at a moment in time, the inputs and outputs of the system are values of state variables.

Assumptions

There are a number of assumptions in the above that should be pointed out. First, parts as defined above are expected to be encapsulated such that they have well-defined inputs and outputs that are the key determinants of the designed behaviors. All other "contextual" dependencies should either be constant, properly averaged, or have minimal impact on behavior, which may not be realistic. (If they need to be considered, the definition of the part can be extended to incorporate those factors that affect its performance.)

In addition, the definitions developed require that we choose a mathematical representation for the states and rates of change to represent the processes underlying of the functions of interest. Once we choose the appropriate representation, the parts and compositors define a system of equations we can either attempt to solve analytically (which is rarely possible for these complex systems) or for which we can compute numerical solutions. As designers we use the mathematical models both to find designs that meet our specifications (e.g. through searching for part types, their connections, and their quantitative behaviors that together meet some goal) or to optimize a design by pinpointing physical parameters that need to be tweaked to achieve a performance goal, as described in Chapter 1. The choice of mathematical representation, though, introduces implicit assumptions than must be considered carefully. We will address these issues later in the book.

3. Enzymatic Reactions

Most of a cell's physiology and behavior is mediated by the enzymatically driven reactions. Therefore, to use the framework above in a computational design setting, we must define enzymes in terms of parts and compositors and discuss how to represent them mathematically.

A Brief Introduction to Enzymology

Enzymes, composed of proteins and sometimes RNAs, catalyze chemical reactions by operating on some set of substrates that bind in the enzyme's "active site" and are converted to products. Enzymes reduce the inherent energy barrier for a given reaction, catalyzing reactions that could theoretically occur spontaneously in some cases, but are prohibitively slow because the energy barrier is too high. The enzyme provides an "active site" to which the metabolites can associate, which provides an electrostatically favorable environment that effectively lowers the reaction barrier such that the reaction can occur faster. The state of the enzyme in this reaction is not changed.

The mechanistic complexity of enzymatic reactions can vary greatly. For example, enzymes can be composed either of single polypeptides or polyribonucleotides, or complexes of multiple macromolecules, and may or may not depend on addition small molecule co-factors for their activity. Environmental conditions and relative concentrations of all substrates and products can bias the relative rates of each of these activities. They also can couple reactions so energy can be harnessed to drive the reaction preferentially in one direction. Both the degree of substrate binding and the rate of substrate turnover may be regulated, either competitively, by binding of other molecules to the active site, or allosterically, in which molecules binding at distal sites affect the activity. Further, enzymes can have multiple activities and also show promiscuity in which a number of different substrates can be consumed to produce a number of variant products.

Finally, thermodynamically, all reactions are reversible; that is, they can operate in both directions. However, the properties of substrates and products and coupling to other reactions may render the reaction practically unidirectional. We will often present chemical reactions as only proceeding in one direction, only including the back reaction when necessary. In the example above, we asserted a simple mass-action, unidirectional catalysis formulation for an enzymatic conversion. We now explore the limitations of this representation using our parts/compositor framework.

Modeling a Simple Enzymatic Reaction

To use enzymes in a formal design framework, we must develop appropriate computational models. First, consider a simple conversion of metabolites M1 into M2 catalyzed by the enzyme E: $M_1 + E \rightarrow E + M_2$

First, we need to be able to store and retrieve information about parts in a way that is useful to computer programs.

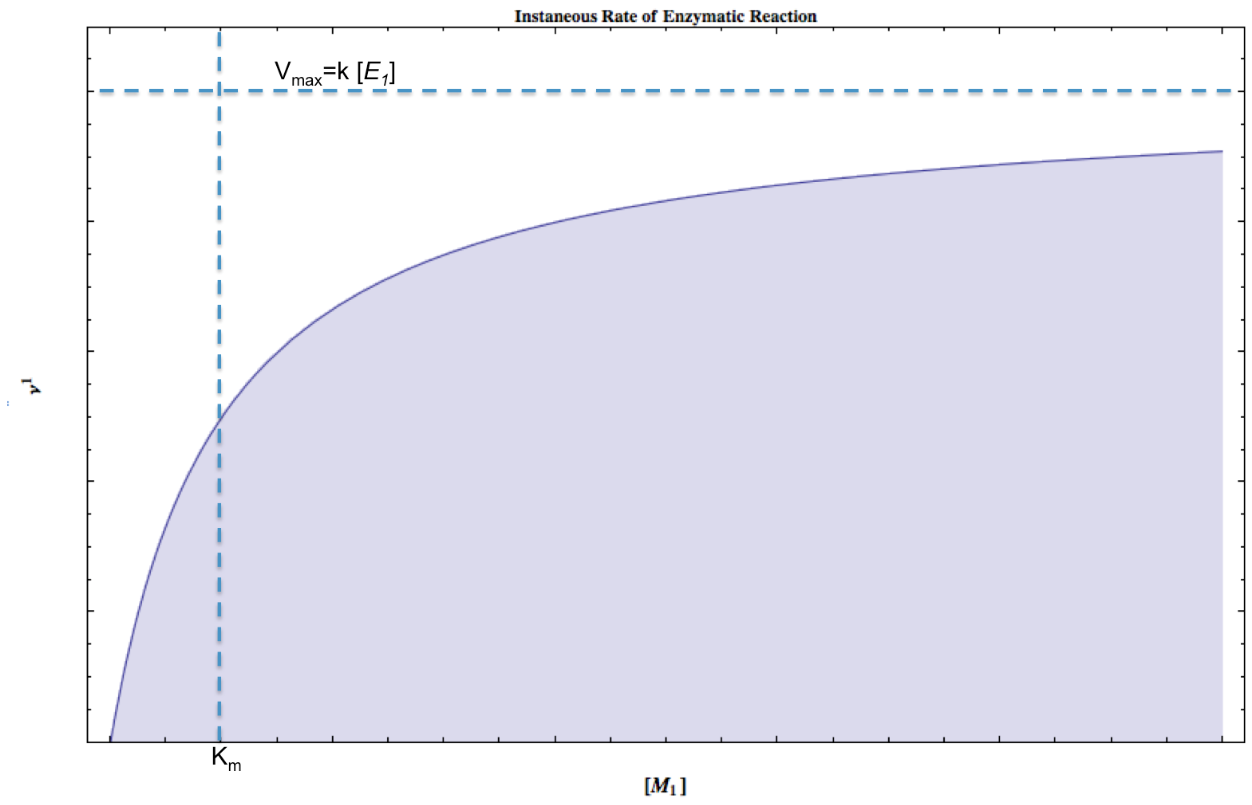


Figure 5. A cartoon of the instantaneous rate of an enzymatic reaction. It is generally observed that as you increase the amount of a substrate, M_1, the rate of the reaction increases to an asymptotic value, V_{max}, determined by the amount of enzyme in the system and its intrinsic turnover rate. In this figure we refer to the value of [M_1] at which the reaction has half its maximal rate as K_m.

To formally specify the part "*M1 + E* ⟶ *M2 + E*" we have to state a *mechanism* for the part. As written it looks like a mass action reaction, in which case the rate of M2 production is

$$\nu^1 = k * [M_1] * [E].$$

However, enzymatic rates are known to "saturate" at some amount of substrate (Figure 5). For example, there is a point above which increasing the concentration of M1 no longer increases the M2 production rate. Such saturation behavior is not appropriately captured in the above rate expression, suggesting that a more complex mechanism is required.

A different representation stems from a simple treatment of the individual reaction steps of the conversion:

$$M_1 + E \leftrightarrow E \cdot M \rightarrow E + M_2$$

In this case, the rate must be derived. One such derivation leads to the so-called “Michaelis-Menten” (MM) form of the kinetics (see Appendix XX). M2 production in this picture is given by:

$$v^1 = k[E]\frac{[M_1]}{K_m + [M_1]}$$

As $[M_1]$ goes to infinity the rate goes to a maximum of $V_{\max}=k[E]$ and the half-maximum rate is achieved when $[M_1]$ is exactly equal to K_m as in Figure 5.

This example reminds us, that, as designers, we must make choices about how we mathematically represent our systems and their substituent components, and these choices can have important implications when we are investigating our design validity. The importance of representational differences depends on the conditions and the problem being solved.

For example, let’s assume we need to estimate the production of M1 in our circuit, and we don’t care about perturbing the levels of M1, but we only have sensors for M2. We use E to convert some of M1 to M2 so that M2 can be a proxy for M1 concentration. If [M1] is relatively low and varying within the pre-saturation regime below K_m, then both models predict a similarly linear response. In this case, the design effectiveness would be essentially independent of the model choice. On the other hand, if M1 is varying within the saturation region of the second model far above K_m, then the first model predicts a nice linear response and the second model predicts zero response (or nearly). In this case, the apparent effectiveness of this design is crucially affected by the mechanism choice.

Regardless of the mechanistic model we choose for this simple enzymatic reaction, we must consider a crucial question: What is the part? Again, it may be tempting to say the enzyme is the part, because it is what you would clone and express in the cell to make this conversion, but remember, we have defined the part as the *process*, made up of a series of chemical reactions, *not* the individual molecule. Therefore, the part is really a process that links three molecular species which comprise *state variables* of the system. That is, the part is transformative of state and assumes the existence of those variables. In this case, the state of each species is simply its concentration, and the state is changed through the action of the reaction part. From a molecular biology point of view, simply cloning the enzyme into the host doesn’t guarantee that this part is present. If there is never any A to convert in this host, then the part is not functioning.

Example: A Simple Enzymatic Reaction

To begin illustrating how we mathematically represent enzymatic reactions in biological simulations, we will begin by modeling the simple enzymatic reaction discussed above. For this exercise, we will consider the simple MM model of the process:

$M_1 \rightarrow M_2$ mediated by E_1 with Rate Law: $k_{cat}[E_1]\frac{[M_1]}{(K_m+M_1)}$

We can model this part by mathematically representing the changes of state of all molecules. First we must initialize the system by specifying the initial states (concentrations) for each state variable. For example:

$$[\mathrm{E}_1] \text{ (at time 0)} = 10\mu\mathrm{M}$$
$$[\mathrm{M}_1] \text{ (at time 0)} = 10\mu\mathrm{M}$$
$$[\mathrm{M}_2] \text{ (at time 0)} = 0\mu\mathrm{M}$$

In this approximation, we will use ordinary differential equations to model the change in state of our system, encoded by the compositors:

Compositor for E_1: $\frac{d[E_1]}{dt} = 0$

Compositor for M_1: $\frac{d[M_1]}{dt} = -k_{cat}[E_1]\frac{[M_1]}{(K_m+M_1)}$

Compositor for M_2: $\frac{d[M_2]}{dt} = k_{cat}[E_1]\frac{[M_1]}{(K_m+M_1)}$

We then computationally evaluate these equations using an appropriate tool, such as our MATLAB PoSB Companion, with the output shown graphically in Figure 6. Note that even such simple equations are difficult to solve analytically to get closed forms. In addition, when the system is represented in a computational form, it becomes a sharable format that can be easily linked to other data and other models of the other parts.

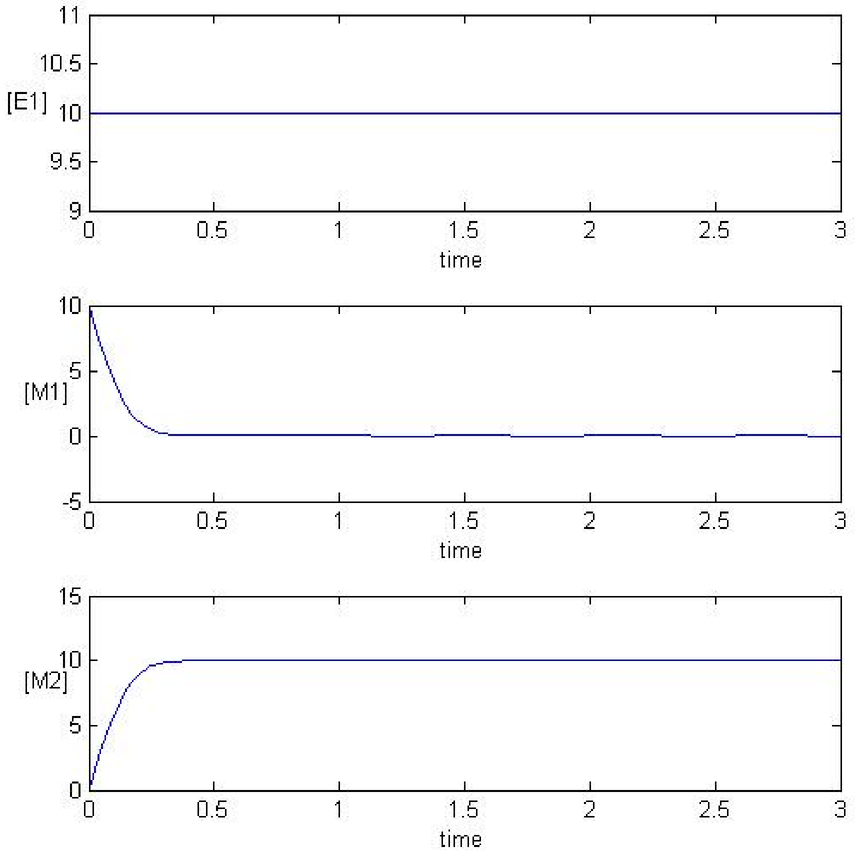


Figure 6. Output from simulation of the simple enzymatic reaction.

As expected from the differential equations, the concentration of enzyme E_1remains constant over time. The substrate, M_1, is stoichiometrically converted to the product M_2; that is, exactly as much M_2 is produced as M_1 is depleted. From direct inspection of the equations, we can tell that all M_1 will ultimately be converted to M_2 in this case, as is illustrated in the figure.

This simulation also tells us something about the time it takes for the reaction to proceed, which is an important property of engineered systems. As an example, if the enzyme were converting a prodrug to a drug at a site of action, the time it takes to do such a conversion may determine how long a treatment takes or if it can be effective when other processes may be converting the prodrug to an inaccessible form. In this case, we can determine the speed at which the reaction reaches 90% completion simply by inspecting the simulation results to determine when the value of M_1 goes below 1 and the value of M_2 rises above 9, which turns out to be approximately 0.25 time units.

Designing a Desired Output for the Simple Enzymatic Reaction

We may also want to control the time it takes to reach 90% completion, for example, to make sure that the reaction is fast enough to compete with other endogenous prodrug clearance processes. In principle, there are a few ways we could speed the reaction. We could increase the concentrations of either the substrate or the enzyme, or both, or we could attempt to change the fundamental reaction kinetics (k_{cat} and K_m). We assume, however, that the amount of initially available substrate is a known quantity, and that we cannot change it. We are also constrained by the amount of enzyme that the system can produce. Hence, a reasonable design strategy could be to change the enzyme kinetics of E_1, which means changing the properties of the part.

Experimentally, we might be able to use directed evolution to generate a library of E_1 mutants with potentially more desirable kinetic properties. We would then measure the $M_1 \rightarrow M_2$ conversion rate for each such E_1 mutant, specifically their k_{cat} and K_m, and select the mutant enzyme that best fits our goal.

We can determine our goal values computationally by simulating the system using parts with different values of k_{cat} and K_m to determine combinations that yield the desired temporal response. The number of theoretically possible combinations of k_{cat} and K_m is infinite, but we would consider only those that fall within realistic physical ranges. Assume the desired completion time is t_{dc}, which we will set as 4 time units in this case. Using the simulation tool, we can scan through ranges of k_{cat} and K_m, graph their completion time difference from t_{dc}, and display that using a surface or heat map. The algorithm would be structured in the following way:

```
iterate over low to high $k_{cat}$ values
    iterate over low to high $K_M$ values
        construct the corresponding system model
        set starting time to zero
        run simulation for a sufficient amount of time
        determine 90% completion rate
        compute difference from desired completion rate
        store this value in a matrix
```

The MATLAB PoSB Companion provides the corresponding programming code to generate this data matrix and represent it visually using a heat map (Figure 7).

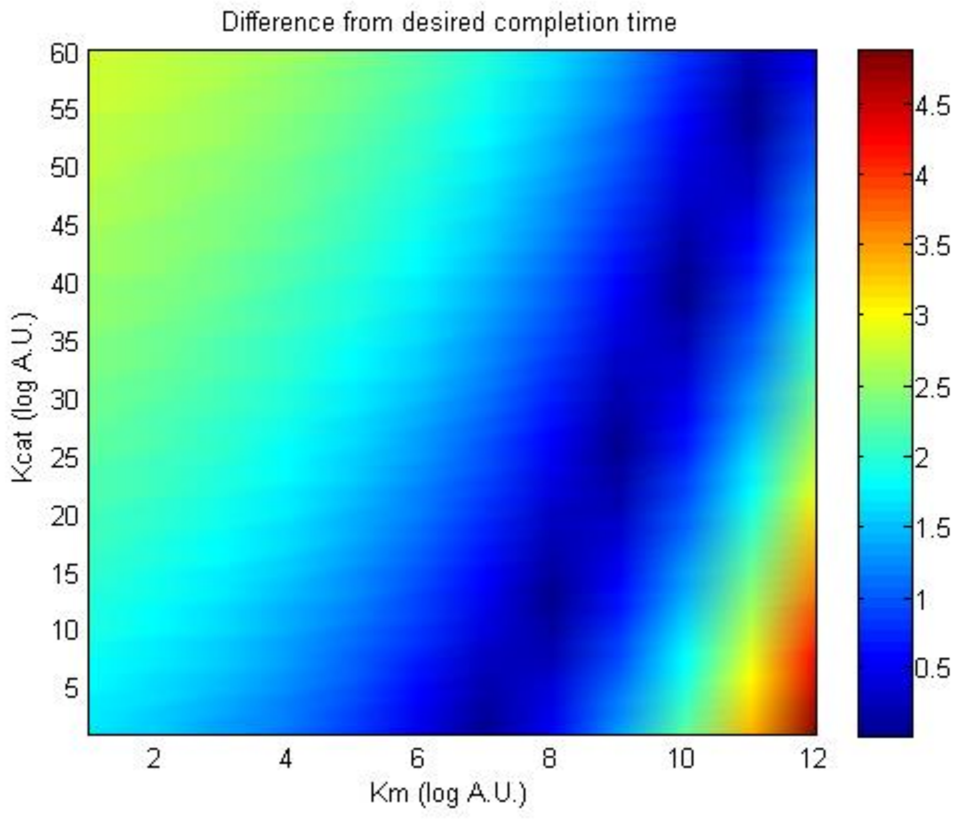


Figure 7: Heat map of matrix for enzyme engineering, depicting distance from desired 90% completion rate of 4 time units. In dark blue are the corresponding kcat/Km values that meet our design goals.

With this heat map, we can evaluate the expected performance of any E_1 mutant with respect to our desired timing completion objective. Those combinations of kcat and Km that meet our design goals fall along the dark blue diagonal. Note that the plot shows strong diagonal parallel lines of equal efficiency, where one can trade k_{cat} against K_m to achieve the same completion times. Logically, this makes sense: the lower the affinity of the enzyme of its substrate (Km), the faster the catalytic rate (kcat) must be to achieve the same overall efficiency.

In fact, at low substrate concentration one can linearize the Michaelis-Menten terms:

For low M_1 ($M_1 \ll K_M$), we get $\frac{M_1}{(K_M+M_1)} \Rightarrow \frac{M_1}{K_m}$

and hence $\frac{d[M_1]}{dt} \approx -(\frac{k_{cat}}{K_M}) \cdot [E_1][M_1]$

which can be solved as

$[M_1](t) \ = M_1(t=0) \cdot e^{\frac{-k_{cat}[E_1] \cdot t}{K_M}}$ (assuming E_1constant)

The time constant for this equation is the ratio of k_{cat} and K_m, which provides the mathematical foundation for the tradeoff graphically depicted in the figure.

From the equation above we can get the "desired time" in terms of desired $\frac{\mathrm{k_{cat}[E_1]}}{\mathrm{K_M}}$

$$log[\frac{M_1(t)}{M_1(0)}] = -\frac{k_{cat}[E_1]}{K_M} \cdot t$$

For the case we have been describing, we want $\frac{M_1(t)}{M_1(0)} = 0.9$, the $\log_{10}$of which is -0.046.

It follows that:

$$0.046 = \frac{k_{cat}[E_1]}{K_M} \cdot t_{90\%\_desired}$$

Rearranging, we see that:

$$t_{90\%\_desired} = 0.046 \cdot \frac{K_M}{k_{cat}[E_1]}$$

We wanted$\mathrm{t_{90\%_desired}} = 4$, so:

$$4 = 0.046 \cdot \frac{K_M}{k_{cat}[E_1]}$$

Rearranging again, for our desired objective we see that:

$$\frac{k_{cat}[E_1]}{K_M} = \frac{0.046}{4} \approx 0.011$$

The requirement for 90% completion by 4 time units only constrains the ratio of$k_{cat}$to K_m, not the individual values, so one may trade one against the other to achieve the same desired effect with different combinations.

This simple example demonstrates a few key concepts. First, there are key parameters that critically determine the operating characteristics of your designed system. Here there are quite a few, three which we examined independently: $\mathrm{k_{cat}}$, $\mathrm{K_m}$, and the level of $\mathrm{E_1}$ expression, all of which are important to time to completion. We saw that there isn't just one unique solution, but a locus of parameters that meet specification $\frac{k_{cat}[E_1]}{K_M}$= constant.

Practical Implementation of Design Parameters: A Simple Enzymatic Reaction

Now consider yourself engineering this. Which parameters can you most easily, and perhaps rationally, manipulate in the laboratory?

a) k_{cat}?

There is progress in rationally engineering enzymes to modify both k_{cat} and K_m (e.g. PROSAR methods) and there are library approaches to mutate the enzyme and screen for the right activities, but there are also still significant barriers to overcome. Specifically, rational design is still an unreliable art. Libraries can affect MANY properties of the enzyme other that the target, including thermostability, specificity, etc. If you know the location and structure of the enzyme active site, libraries can be better directed to affect only the desired parameters, but not all residues that affect k_{cat} and Km are located in the active site, and changing residues only in the active site may still affect other properties besides k_{cat} and K_m.

b) K_m?

The issues that impede rational design of k_{cat} also apply to K_m.

c) *[E1]*?

This is basically an issue of how much enzyme we can make, which is relatively easier to change rationally without directly affecting the enzyme function or other physical parameters. We can manipulate the enzyme's transcription and translation to make as much polypeptide as we need. We will discuss this approach further in the next section.

However, even if we make the right amount of polypeptide, there can be other challenges, like ensuring that translation is error-free, the polypeptide folds efficiently and is soluble, and in some cases has the proper post-translational modifications to function.

When moving toward practical implementation, a designer must also be aware of the many assumptions involved in developing a model. In the example above we made the following assumptions:

- We assumed a particular form of the kinetic mechanism, Michaelis-Menten, which is usually a good approximation, but there are many other such mechanisms that might apply and be more realistic.

- We chose a particular mathematical framework in which to analyze the system (homogeneous ordinary differential equations). This picture can sometimes break down with startling consequences (See section XXXX).

- We resorted to approximations about the relationships between physical parameters of the system ($M_1 << K_m$). Such approximations might be wrong, or only be correct in certain operation regimes. If we initialize the system this way, our approximation would hold for the entire time. but if we started with saturated enzyme, the approximation would only hold near the end of the process.

- We put forward a particular, process-oriented way of defining a synthetic biological part, rather than thinking about a particular physical piece of DNA encoding the molecules that physically carry out the process. Our part, in this case, was about the interactions about E_1, M_1 and M_2 rather than just "E_1" and the DNA encoding it. We show how this allows us to "compose" simulations of a system designed from such parts as a set of equations implied by the interaction among the processes.

In addition, we didn't consider whether we arrived at realistic solutions. Our specification could have been unrealistic. If we had asked for completion to occur faster than molecules can move, for example, we could have solved the equations and determined *Km'*s and k_{cat}'s that would meet the goal but could never be realized physically. As discussed in Appendix XXXX, k_{cat}/K_m can not be more than about 10^9 mol/L*sec in a free solution because of the maximal diffusion rate of molecules in water. If enzymes and substrates are somehow physically linked together so the paths to putting the substrate into the active site aren't diffusive, these values can be higher (see Substrate channelling and scaffolding Section XXX).

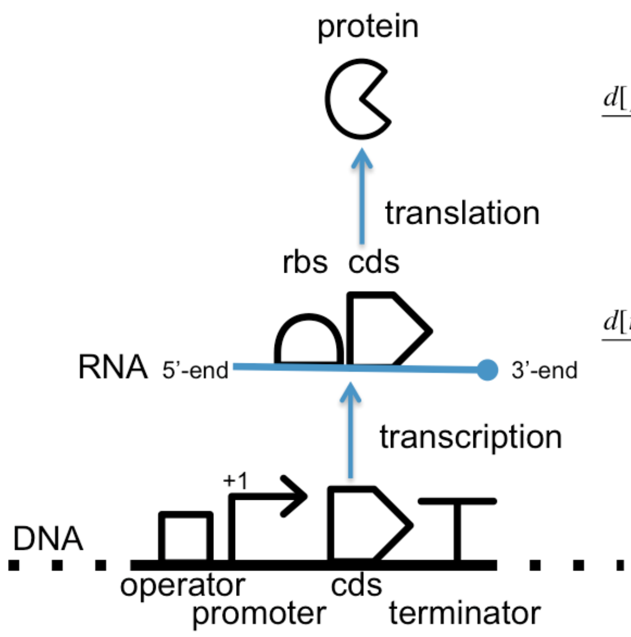
Even with realistic specifications, parameter choices to meet spec may still be unrealistic. For example, because design constrains only the ratio of kcat/Km and not the actual values of either, one can choose physically unrealizable values as targets; such enzymes would be impossible to make. There can also be thermodynamic limits on these values for specific chemical reactions, as discussed in Section XXXX. When the physical constraints on the parameters are available, these can be used to direct the search and engineering for parts meeting specification.

4. Gene expression

Engineering gene expression can be a powerful design tool to control the concentration of certain molecules, with potentially important implications for system kinetics and overall function. To add this biological process to our design framework, we need to determine how to represent it computationally, which, as always, will require certain simplifications and assumptions.

A Simple Gene Expression Model

In reality, gene expression is a complex process involving multiple steps and a variety of macromolecular machines. A technically accurate model would include promoter search modulated by transcription factors, open-complex formation, transcription initiation and elongation of the transcript, degradation of the transcript, translation of the transcript (ribosome binding, initiation, elongation), and polypeptide production, folding (possibly aggregation and inclusion body formation), and degradation. A partial simplification of this is shown in Figure XX below.



$$\frac{d[protein]}{dt} = k_t[mRNA][Ribosomes] - k_{dp}[protein]$$

$$\frac{d[mRNA]}{dt} = k_{oc}N_P\frac{[RNAP]}{K_P + [RNAP]} - k_{dm}[mRNA]$$

Figure 8. One abstraction of the processes underlying gene expression. A region of a DNA molecule may, in its sequence of nucleotides, encode sites (promoters) where RNA polymerase (RNAP) can bind and begin transcription, perhaps modulated by the presence of nearby transcription factors bound to operator sites that prevent or augment these processes. This region of DNA may be present in many copies (N_p) if on a high copy plasmid or if the genome is replicating more quickly than division, as is sometimes the case in *E. coli*. When RNAP transcribes the DNA "downstream" from the promoter into RNAP, starting at the "+1" site, it ultimately stops at a region that encodes a terminator. The resultant RNA is translated into perhaps multiple copies of a protein when ribosomes bind to and initiate translation from the ribosome binding site (RBS) positioned upstream of a coding sequence (CDS). The equations to the right of the graphic show one possible representation of the transcription and translation dynamics. The constants k_{dp} and k_{dm} refer to the degradation rates for the protein and the mRNA, respectively. These equations abstract away the dynamics of the processive transcription and translation elongation processes, the effects of DNA replication dynamics, and dependence on other host resources such as nucleotides and amino acids.

For the purposes of this simple introduction, however, we will model gene expression as the one-step production of folded, soluble protein from a "genetic unit". Despite the many complexities of the biological system, this simplified model often captures well the overall process. We can "derive" this one-step model by simplifying the model in Figure XX. If we assume "fast" production and decay of mRNA species compared to transcription rates and decay of protein, and that RNAP and ribosomes are at constant concentration in the cell, we can collapse the terms $k_{oc} * N_P \frac{[RNAP]}{K_P+[RNAP]}$ and $k_t[ribosomes]$ into the constants k_{tx} and k_{tr}, respectively. Further, we can assume the dynamics for $[mRNA]$ reach steady state quickly compared to everything else, so $\frac{d[mRNA]}{dt}$=0. Thus, solving for the steady-state amount of mRNA from its equation we find $[mRNA]_{\mathrm{steady-state}} = \frac{\mathrm{k_{tx}}}{\mathrm{k_{dm}}}$. Plugging this into the equation for protein production we find:

$$\frac{d[protein]}{dt} = \frac{k_{tr}k_{tx}}{k_{dm}} - k_{dp}[protein]$$

which we treat as the compositor for *P*. Based on this compositor, the cognate "abstracted" chemical "parts" for this system with protein P can be broken down into:

(1) Promoter → Promoter + P $\quad v_P^1 = \frac{k_{tr}k_{tx}}{k_{dm}}$

(2) P→ $\quad v_P^2 = -k_{dp}[P]$

where equation (1) refers to protein production and equation (2) describes the protein's degradation.

Given an initial condition, say P_0, we can rearrange the compositor to solve for the concentration of protein at a given time, *[P](t):*

$$[P](t) = \frac{k_{tr}k_{tx}}{k_{dm}k_{dp}} + (P_0 - \frac{k_{tr}k_{tx}}{k_{dm}k_{dp}})exp(-k_{dp}t)$$

Each term in this equation reveals key control parameters for the deterministic dynamics of this system. The first term is the "steady-state" of the system: the value of [P] as t goes to infinity. The amount of P is a competition between the production terms in the numerator and the degradation term in the denominator. The second term describes the relaxation from the initial condition to the steady state. The first factor ensures that the decay is from the initial condition to the steady state. The second factor, *$exp[-k_{dp}t]$*, shows that the second term disappears as t goes to infinity. The time scale is given by $1/k_{dp}$: the degradation time of the protein.

This equation shows how two key properties of the system, the steady-state concentration of P and the time it takes to establish this state, are fundamentally linked through the protein's degradation dynamics. In other words, we may increase the steady-state protein level by decreasing its degradation rate, but at the cost of achieving steady-state levels more slowly. On the other hand, we can also increase the steady-state protein level by changing the transcription rate or translational efficiency without affecting the time scale for achieving the steady-state.

Complications, Assumptions, and Implementation

Modeling gene expression can of course become more complicated. The above equation assumes a deterministic model of gene expression, which is not necessarily a good model, as we will discuss in a later section. The discrete and stochastic (random) nature of molecular collisions and reactions can lead to variability. Thus, new measures of performance arise that are differentially affected by the physical parameters used in the equation above.

In addition, transcription, translation and degradation all have an energy cost for the cell. Thus, it is energetically costly for the cell to try to maintain a target steady-state protein level while increasing the response speed, which requires increasing transcription, translation, and degradation of the protein simultaneously and in proportion. If our objective is to minimize time to steady-state, there must be a tradeoff: the faster the relaxation to steady state, the more energetically costly.

Roughly, to maintain a steady-state one needs to hold $\frac{k_{tr}k_{tx}}{k_{dm}k_{dp}}$constant. To decrease relaxation time one needs to increase k_{dp}. To increase k_{dp} while holding $\frac{k_{tr}k_{tx}}{k_{dm}k_{dp}}$ constant,$\frac{k_{tr}k_{tx}}{k_{dm}}$ must be increased. One could stabilize the mRNA by decreasing k_{dm} but then we would have to examine if our assumption of fast mRNA dynamics remains intact. We could alternatively increase k_{tr} or k_{tx} both of which incur an energy cost in protein synthesis. All solutions may tie up polymerases and/or ribosomes in the expression of this particular gene, sequestering them away from other genes, including those that permit the cell to grow and divide. This may then incur a fitness cost as well--expression of the target gene may cause the host cell to be less fit than its neighbors--which might ultimately lead to the loss of one's circuit from the environment.

Finally, the engineer needs to think about how easy it is to genetically implement changes in the key parameters. For example, there are genetic elements that are known to predictably mediate the degradation of transcripts and proteins. Similarly, there are known means to alter transcriptional and translational efficiencies. We will revisit the standardization of parts in later chapters. The choice of how to achieve a particular functional design goal then, by choosing parts and parameters for even simple systems can be fairly complex.

Introduction to gene regulation

Instead of tweaking individual parameters, we can also introduce gene regulation to achieve rapid response, as briefly introduced in our tissue homeostasis example from Chapter 1. Specifically, negative feedback, wherein a protein represses its own production rate when present above a certain level, can reduce the time-scale need to achieve a given concentration. It may seem counterintuitive that a negative feedback, a mechanism that seems to slow protein production, could speed response time, but it works because one can use very high transcription and translation rates to get a fast initial rise that is then quickly clamped down when the protein level rises enough to trigger the repression. (For a derivation of this result see PMID: 12417193)

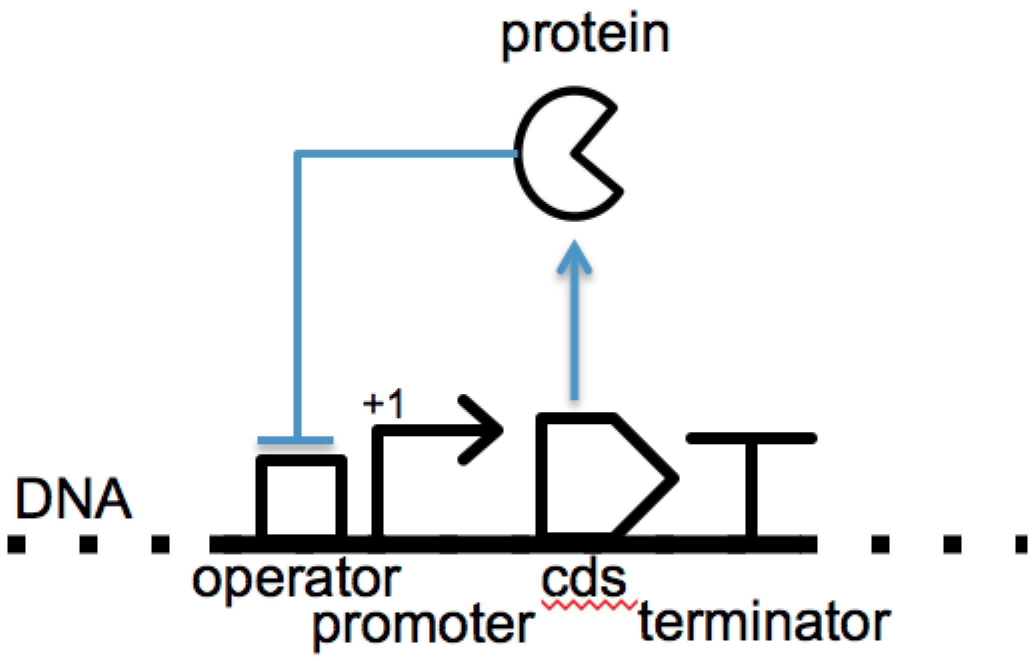


Figure 9. A depiction of the "reduced" negative feedback circuit. In this case binding of the protein to the operator site prevents RNAP from transcribing the gene.

Here we will use negative feedback to introduce the modeling and analysis of gene regulation. We model negative feedback by adding reactions in which the protein binds and sequesters the promoter away from polymerase.

(1) Promoter → Promoter + P $\quad v_P^1 = \frac{k_{tr}k_{tx}}{k_{dm}}[Promoter],\ v_{Promoter}^1 = 0$

(2) P→ $\quad v_P^2 = -k_{dp}[P]$

(3) Promoter + P → Promoter::P $\quad v_P^3 = v_{Promoter}^3 = -v_{Promoter::P}^3 = -k_f[Promoter][P]$

(4) Promoter::P → Promoter + P $\quad v_P^4 = v_{Promoter}^4 = -v_{Promoter::P}^4 = k_r[Promoter::P]$

Now, our compositor for P is:

$$\frac{d[P]}{dt} = v_P^1 + v_P^2 + v_P^3 + v_P^4$$

However, we can make an assumption about promoter dynamics that will allow us to cancel out the last two terms. Specifically, we assume that the binding and unbinding of the protein to the promoter is much faster than the transcription and translation reactions. (This approach is similar to how we arrived at our simplified protein production equations, when we assumed "fast" production and decay of mRNA species compared to transcription rates and decay of protein.) With this assumption, we can treat the dynamics for [*Promoter*] and [*Promoter::P*] as steady state. For simplicity we will say the binding is in quasi-equilibrium, such that:

$$\frac{d[Promoter]}{dt}=v^3_{Promoter}+v^4_{Promoter}=-k_f[Promoter][P]+k_r[Promoter::P]=0$$

reducing our compositor for P to:

$$\frac{d[P]}{dt}=v^1_P+v^2_P=\frac{k_{tr}k_{tx}}{k_{dm}}[Promoter]-k_{dp}[P]$$

Our goal is to eliminate the promoter dynamics from our representation and return to the simpler representation of constant amount of promoter as we had above. Since the promoter is neither created nor destroyed in these reactions, we know its total amount is conserved:

$$[Promoter]_{Total}=[Promoter]+[Promoter::P]$$

Using the definition$K_D=\frac{k_r}{kf}=\frac{[Promoter][P]}{[Promoter::P]}$, this conservation equation can be rearranged as follows:

$$[Promoter]_{Total}=[Promoter]+\frac{1}{K_D}[Promoter][P]$$

$$[Promoter]=[Promoter]_{Total}\frac{K_D}{K_D+[P]}$$

Plugging this back into the compositor for P we find:

$$\frac{d[P]}{dt}=\frac{k_{tr}k_{tx}}{k_{dm}}[Promoter]_{Total}\frac{K_D}{K_D+[P]}-k_{dp}[P]\ =k_p\frac{K_D}{K_D+[P]}-k_{dp}[P]$$

This has introduced only one new constant to the open-loop equations for protein production, K_D, but has substantially changed the "topology" of the reaction graph. As illustrated in Figure 10, the negative feedback loop allows the system to reach the same steady state more quickly than does the unregulated system. Therefore, this new design parameter can change the landscape such that we can have an initially much stronger promoter that is then shut down at high P.

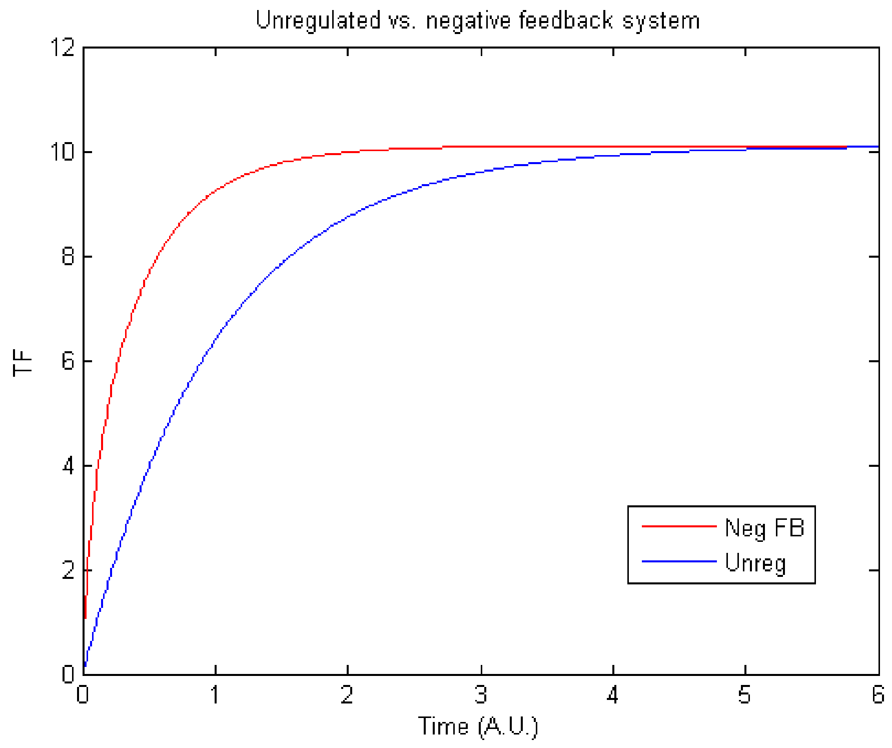


Figure 10: Unregulated promoter versus promoter with negative feedback.

However, the increased speed to steady-state does have a cost: to achieve the same initial speed in the open-loop system while maintaining the steady-state target, we have to balance the strong promoter by stronger protein degradation. We can derive how much more productive unrepressed transcription and translation must be in the closed-loop system to achieve the same steady state as obtained in the open-loop system for a given strength of repression. For the closed-loop system at steady state (compositor = 0), we have:

$$k_p \frac{K_D}{K_D + [P]} - k_{dp}[P] \ = 0$$

Rearranging, we obtain:

$$[P]^2 + K_D[P] - \frac{kp \quad K_D}{k_{dp}} = 0$$

and via the quadratic equation:

$$[P]_{steady-state} = \frac{-K_D \pm \sqrt{K_D^2 + 4\frac{kpK_D}{k_{dp}}}}{2} = \frac{-K_D \pm \sqrt{K_D^2 + 4bK_D}}{2}$$, where $b = k_p/k_{dp}$

Remember that, for the open-loop system, we derived:

$$[P]_{steady-state} = \frac{k_{tr}k_{tx}}{k_{dm}k_{dp}} = a$$

If the two steady states are equal, we can set *a* equal to the right hand side of the equation for the closed-loop system and solve for *b* as a function of *a and* K_D. That is, we can find out what ratio of synthesis and degradation in the closed-loop system is required to match the steady state in the open-loop system given the repression constant K_D:

$$b = \frac{a^2 + aK_D}{K_D}$$

As the dissociation constant goes towards infinity (that is, the protein does not repress the promoter), b converges towards a, the unregulated steady-state ratio of synthesis and degradation as expected. But as the dissociation constant goes to zero, meaning higher repression, b becomes very large in order to maintain the steady-state when repressed. So, there is a higher cost during the initial synthesis than in the open-loop system, but at steady state it converges to the same cost.

Figure 11 shows the energetic costs of achieving this speed-up. Since the cost is incurred only at start-up, it is likely less costly to achieve fast time response in this fashion than by simply using a strong promoter and a rapidly degrading protein, which would incur the energy cost over the entire lifetime of the system.

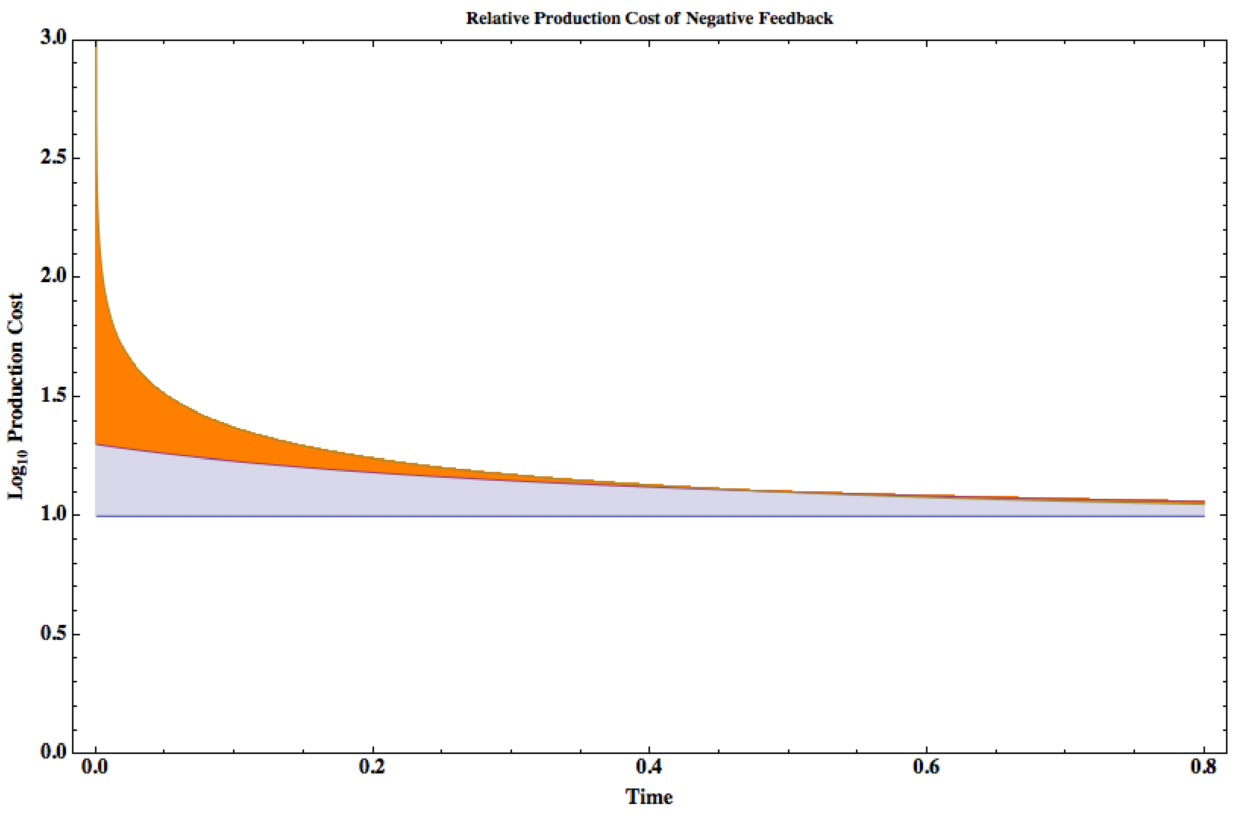


Figure 11. Relative cost of production (log-scale) over time in the negative feedback system with the same steady-state. Two different scenarios are shown. The curve bounding the blue shaded area is the cost incurred using a less repressive system. The curve bounding the orange space is the cost for repressing 10,000 fold more strongly.