# Modeling a *Hox* Gene Network

## Stochastic Simulation with Experimental Perturbation

Thesis by
Jason Kastner

In Partial Fulfillment of the Requirements
for the Degree of Doctor of Philosophy


California Institute of Technology
Pasadena, California

2003
(Defended September 25, 2002)

## Acknowledgments

Even though my name is on the title page, I am deeply indebted to a number of people, and without their help this work would not have been possible. My advisors Jerry Solomon and Scott Fraser were both instrumental in every aspect of my research, and this thesis would not have been nearly as interesting or complete without their continual guidance and help. Thanks as well to the rest of my committee, Joel Franklin, Niles Pierce, and Dan Meiron, and the funding from the Computation Molecular Biology program at Caltech, made possible by the Burroughs Wellcome fund.

All of the members of the Fraser lab helped my research, but Rusty Lansford, Paul Kulesa, Helen McBride and Reinhard Koester deserve special thanks for their advice and support. At the Stowers Institute for Medical Research, thanks to Heather Marshall, Kristen Correia, and especially Robb Krumlauf, who was incredibly generous with his time and resources.

My parents Victoria and George Kastner never failed to profess their belief in both my abilities and me, and for that I am forever indebted. But my deepest gratitude goes to my two closest friends, Jennifer Dooley and Tri Lindhom. They both finished their dissertations several years ago but were forced to relive it all again through me. Their constant support and encouragement through it all was invaluable. Thank you both.

## Abstract

The *Hox* genes show a striking segment specific pattern of expression in a variety of vertebrate embryos, and have been the topic of many experimental analyses. There are now sufficient data to construct a higher-level model for the interaction and regulation of the *Hox* genes. This thesis presents the results of an investigation into a regulatory network for the early *Hox* genes. Instead of using conventional differential equation approaches for analyzing the system, a stochastic simulation algorithm has been employed to model the network. The model can track the behavior of each component of a biochemical pathway and produce computerized movies of the time evolution of the system that is a result of the dynamic interplay of these various components. The simulation is able to reproduce key features of the wild-type pattern of gene expression, and *in silico* experiments yield results similar to their corresponding *in vivo* experiments. This work shows the utility of using stochastic methods to model biochemical networks and expands the stochastic simulation algorithm methodology to work in multi-cellular systems. In addition, the model has suggested several predictions that can be tested i*n vivo*.

A tight connection was also created between the modeling and laboratory experiments. To investigate a connection between two components of the network, retinoic acid (RA) and *Hoxa1*, a novel laboratory experiment was performed to perturb the system. An RA soaked bead was implanted into the neural tube of a developing chick embryo and the effect of the exogenous RA was assayed with an *in situ* hybridization for the gene *Hoxa1*. The resulting expression patterns suggested that one aspect of the model

design was not accurate, and based on these results the model was modified to encompass the new data, without losing the fit to the original data sets. The thesis work was therefore brought full circle, thus showing the utility of an interconnected effort: the act of constructing and using the model identified interesting biology questions, and the answer to one of those questions was used to enhance the model.

## Table of Contents

# List of Figures and Tables

# Chapter 1: Overview

> Every attempt to employ mathematical methods in the study of biological questions must be considered profoundly irrational and contrary to the spirit of biology.
>
> If mathematical analysis should ever hold a prominent place in biology—an aberration which is happily almost impossible—it would occasion a rapid and widespread degeneration of that science.
>
> - Auguste Comte, 1871

## *Introduction*

Every applied and computational mathematics thesis should start with a physical problem, and in that respect this thesis is true to form. Instead of culling a problem from physics however–the traditional inspiration for much of applied mathematics–the problem under investigation in this work was drawn from developmental biology. The goal of this thesis was to investigate a relevant and interesting biological problem from both the modeling and experimental arenas, and show the efficacy of an interconnected effort. This thesis presents the results of an investigation into a regulatory network for a set of genes expressed in the developing brain, the *Hox* genes. The network was created through integrating the results of numerous biology papers and constructing a higher-level model for the interaction and regulation of the *Hox* genes in a multicellular context.

Instead of using conventional differential equation approaches for modeling the resulting system, a stochastic simulation algorithm (SSA) has been employed to model

the network. This work improves on previous SSA investigations that had been limited to intracellular systems by expanding the SSA to work in an intercellular arena. One of the troublesome problems with modeling a multi-cellular system involved cell synchronization, and this was solved with the use of a priority queue to time-order the cells. The model tracks the behavior of each component of a biochemical pathway and captures the dynamic interplay of the various components in the multi-cellular system. The data can be rendered as computerized movies of the time evolution of the system. The simulation is able to reproduce key features of the wild-type pattern of gene expression, and *in silico* experiments yield results similar to their corresponding *in vivo* experiments. In addition, the model has suggested several predictions that can be tested *in vivo*.

An important goal of this thesis was a tight connection between the modeling and experimental work, and two novel perturbation experiments aimed at testing components of the model network were designed. The first investigation addressed the connection between two genes in the network, *Hoxb1* and *Krox20,* and the published hypothesis that *Krox20* is repressed by *Hoxb1* expression (Barrow et al., 2000). A specially constructed piece of DNA designed to repress *Hoxb1* was introduced into young chick embryos, and the effect on *Krox20* expression was assayed. The DNA did not, however, appear to work as intended. The second experiment explored the connection between retinoic acid and *Hoxa1* by altering the normal retinoic acid distribution in the embryo. This was accomplished by implanting a retinoic acid soaked bead into the midbrain of a developing chick and assaying the expression of *Hoxa1*. This experiment yielded intriguing results, and the resulting data suggested that one aspect of the model design

was not accurate. Based on these results the model was modified to encompass the new data, without losing the fit to the original data set. The thesis work was therefore brought full circle, thus showing the utility of an interconnected effort: the act of constructing the model identified interesting biology questions, and the answer to one of those questions was used to enhance the model.

## *Interdisciplinary Work*

With such a strong focus on interdisciplinary research, this work presented a number of challenges that are not typically found in a conventional thesis. They started with the need to learn the vocabulary of a new field. This was accomplished by sitting in on biology courses, reading the biology literature, and interacting with people working in a biology laboratory. At the same time, a search to identify a tractable yet interesting problem was undertaken. The prospect of modeling a gene network appeared fairly early in the research process, yet it took a great deal of time to identify a particular network.

The molecular studies of the hindbrain have offered sufficient details to assemble a model for the interactions important in regional control of gene expression. These factors helped identify a system in which to work; the interconnection of the early *Hox* genes and their connection to retinoic acid. The direct coupling of the stochastic simulation algorithm implementation of a network and individual molecular events would seem to lend itself to both the analysis and logical organization of the ever growing data on the control of *Hox* genes in the developing hindbrain.

One of the important features of the *Hox* system is that the amount of molecular information that has been gathered about the regulatory mechanisms allows for a

synthesis and construction of a higher-level system of interaction. At the same time, the data is far from complete, thus leading to questions that can be investigated through simulation. These include investigations of hypothesized interactions, mechanisms of interaction, and perturbations of the system.

Another key feature of the *Hox* network was an animal model, the chick hindbrain, which allowed for experimental perturbation of the system *in vivo*. A carefully designed experiment could be connected back to the model, and the data gathered from the experiments would offer support for, or evidence against, model hypotheses.

Finally, research into the *Hox* genes is relevant because of their strong connection to diseases. There is evidence linking *Hox* family members to leukemia (Thorsteinsdottir et al., 2001) and breast cancer (Lewis, 2000), and connections to genetic diseases include obsessive-compulsive disorder (Greer, 2002) and autism (Ingram et al., 2000; Rodier, 2000).

The laboratory work was designed from the outset to be a crucial part of this research. The experiments are intimately related to the *Hox* network, and early on in the work it was necessary to move beyond the literature and start work in a laboratory. The literature and consultations with experimentalists provided the initial guidance in perturbation techniques—the bead implantation (Chapter 4) and electroporation (Appendix A)—but the refinement of the methods came through trial and error. To do these experiments, it was necessary to learn an array of supporting techniques. These included early chick embryology and development, tissue culture, microscopy, and a

number of molecular biology techniques including antibody staining, cloning, and *in situ* hybridization. Many of these techniques are described in the thesis. During the course of working in the laboratory, numerous problems that are never mentioned in the literature or classes appeared on an almost daily basis. The Vital Stain experiment in Chapter 4 is an illustrative example.

To present this interdisciplinary work in the proper context, the thesis is broken into the following 5 chapters: Chapter 1 provides an overview of modeling biological problems, an introduction to modeling gene networks, as well as some comments about the goals of modeling in general. Chapter 2 focuses on the modeling of enzyme kinetics by presenting stochastic and deterministic implementations of the basic enzyme reaction and a comparison of the two. Chapter 3 includes an introduction to both developmental biology and the specific biology of the system under investigation. It goes on to present the model itself, and a sensitivity analysis of the model. Chapter 4 is devoted to experimental results, and how the experiments described tie back into the model. Chapter 5 contains the summary and a discussion of the work. The Appendices contain more experimental results, the source code for the simulations, and the laboratory protocols used to perform the experiments.

## *Biological Modeling*

Over 170 years after Comte made his thoughts concerning the role of mathematics in biology known, his sentiments are perhaps too widely shared in the biology community. D'arcy Wentworth Thompson echoed Comte's sentiment when he remarked

that “The introduction of mathematical concepts into natural science has seemed to many men no mere stumbling-block, but a very parting of ways” (Thompson, 1942).

Practically speaking, the reasons for the schism between math and biology are many. They start with the language barrier, a common obstacle between many fields. Unlike math and physics, which are inextricably linked by their vocabulary, math and biology each have a vocabulary that is very difficult for the outsider to understand. This has created a climate that does not encourage true interdisciplinary work and there are numerous instances of mathematics used to solve problems that are supposedly biological in nature, but in truth have little connection. The language barrier also presents problems when communicating the results of the work, but it has been shown that publishing the research in a journal relevant to the new field is an effective form of interdisciplinary information transfer (Pierce, 1999). Therefore, the fact that a portion of this work has been published in the journal *Developmental Biology* (Kastner et al., 2002) is a notable achievement.

Another problem is that modeling biological processes is inherently difficult; there are relatively few “toy problems” that can be easily identified, extracted, and solved. This often leaves an investigator in the difficult position of trying to model a system before it is well characterized. It is sometimes suggested that all the parts of the system must be known before a model can be created, or that any potential modeling approach must be proved on the simplest system before trying to apply it to something more complex. These objections are sometimes put forth as reasons not to start work on a problem, but they are shortsighted and in truth much can be accomplished by trying to model even poorly characterized biological problems. Indeed, a central reason for

modeling biology using mathematics and computers is precisely because the biological systems are so incredibly complex. The facts of the matter are simply these: all the parts of any real biological system are likely to never be known, and even the simplest biological systems are more complex than can be handled by any supercomputer. To quote an oft-repeated sentiment during many biology lectures: "but it's more complicated than that." Not only is it more complicated than that, it is more complicated than we can begin to imagine. Therefore, a major part of the problem with biological modeling is finding tractable yet interesting problems.

Finally, the scientific community is still trying to develop a mathematical framework for biological problems. There is no $F = ma$ for biology, and a variety of techniques can often be employed for each problem that appears. The closest biology has come to a universal law is the Central Dogma which states that genetic information is carried on DNA, then transcribed to RNA and subsequently translated to proteins. Adding to this problem is that data arising from biology experiments, especially in developmental biology, are often qualitative and don't always lend themselves to a rigorous mathematical analysis.

Despite these objections, it is important to try to bring communities together as there is much they can offer each other. For the mathematicians, biology affords a relatively untapped spring of interesting problems, and the opportunity to shape the future direction of investigations. For the biologists, mathematics can provide a framework for the biology problems, especially considering the sheer amount of biology data being generated. It can also be used to quantify results and suggest experiments to test hypotheses, ultimately adding to the understanding of how the biology may work.

## *Gene Networks*

One focus of traditional biology examines single genes or proteins in isolation. While this provides vital information, it is the interaction of these pieces that provides biological results. The logical next step is therefore combining the data from various sources to build a hierarchal picture of the true interactions of the pieces of the pathways. Because of the deluge of information, computer models are the key to the future of the information integration and to the understanding of how the systems work. Not only that, but by a thoughtful investigation into a system, it is even possible to determine the part of the model which may be missing or is not well understood. An excellent example of this has recently appeared with the use of a model to discover a missing control module for a sea urchin gene (Yuh et al., 2001).

Biological networks are the collection of biochemical entities (including messenger RNA, proteins, DNA, ions, or other molecules, like hormones), which interact to produce biological results. An analysis of these systems seeks to elucidate information about the interactions between the genes and their derivatives, and also hopes to provide predictive results about the overall behavior of the system. This type of work is commonly called systems biology because it seeks to simultaneously study the complex interaction of many levels of biological information.

Genetic networks currently lie in the forefront of biological research, and are in the border area where computer simulations and molecular biology meet. The most successful efforts have tightly coupled the modeling and experimental efforts (*cf.* Yuh et al., 1998; Yuh et al., 2001). They are also an area of increasing interest, evidenced by the

growth in the literature. Five years ago a literature search on the term "gene network" returned only 3 references, and none of the works involved modeling. In the first nine months of 2002 however, the same search produced nine times as many results, and a dozen of them clearly involve modeling of some sort.

Various methods have been employed to model biological networks including Bayesian networks (Friedman et al., 2000), rule based formalisms (Meyers and Friedland, 1984), true Boolean systems (Kauffman, 1993) and Boolean/continuous hybrids (Yuh et al., 1998; Yuh et al., 2001) but ordinary differential equations have been the preferred method to construct and analyze biochemical network models. Using the Law of Mass Action, which states that the rate of the reaction is proportional to the concentration of the reactants, it is possible to write down a set of coupled differential equations that hope to describe the time evolution of the system. The reasons for the prevalence of mass action based kinetic analysis are many, but by far the most important one is that the approaches based on differential equations produce results that are in general in good agreement with the data (*cf.* Hynne et al., 2001; Poolman et al., 2001). In addition, differential equations come with a wide range of analysis tools that allow for a detailed investigation of the model properties. But as will be addressed in Chapter 2, differential equations may not be appropriate for modeling biological processes in the small volumes inherent in single living cells.

Compared to differential equations, and despite their prevalence in modeling pure chemical processes, stochastic approaches in biology are still in a relative infancy. This is currently changing, and generalized tools for constructing and analyzing stochastic simulations are now starting to appear (Bray et al., 2001; Kierzek, 2002). A stochastic

process is one governed by a random process, and in a biological context this means that the system is subject to fluctuations. These fluctuations could be in the number of molecules present, the time it takes for a molecular creation or decay process, or the length of time molecules are bound together. More attention has been focused lately on stochastic effects in biology, especially as evidence shows that stochastic effects play major roles in gene expression (Greenwald, 1998; Ko, 1992; Zlokarnik et al., 1998). Instead of treating these factors explicitly, some differential equation approaches attempt to capture stochastic effects by adding a "noise" term to their otherwise deterministic treatment (*cf.* Meinhardt and de Boer, 2001). The resulting "ordinary" differential equation is called the Langevin equation and is of the form

$$\frac{dX(t)}{dt} = -aX(t) + f(t) \tag{1.1}$$

where the noise function $f(t)$ is assumed to be Gaussian and delta-correlated. But in effect this makes the noise term just another parameter instead of capturing it in a physical meaningful way. This may be a somewhat misguided approach: if there are fluctuations in the system that need to be accounted for, it might be preferable to incorporate those effects at the beginning in a way that is physically intuitive and physically based.

## *Stochastic Simulation*

As opposed to the deterministic view in which the reaction constants are the rates, reaction constants in the stochastic approach are considered to describe the probability (per unit time) that a reaction occurs. With this formulation, the chemical system can be

thought of a Markovian random walk in the space of the reacting molecular species. The time evolution of the system is described by the solution of a single differential difference equation, often called the master equation. The independent variables of the master equation are time and the populations of the reacting species. The master equation can be transformed into a partial differential equation by the use of a generating function.

From a mathematical point of view, the set of equations resulting from the Law of Mass Action is usually easier to solve than the corresponding master equation or the associated partial differential equation. In reality, it turns out that if the system involves more than a few reactants and chemical reactions, an analytic solution is out of reach for either method, and it is necessary to use a numerical scheme (McQuarrie, 1967). Of course numerical methods for solving even a single partial differential equation can be a research topic in and of itself; instead what was really needed was a general method for attacking the master equation. This came in 1976 when Dan Gillespie introduced the stochastic simulation algorithm, described in the next chapter (Gillespie, 1976).

Adam Arkin appears to be the first to use Gillespie's method in a biological context with a study of the growth of phage λ, a virus that infects the bacteria *E. coli* (Arkin et al., 1998; McAdams and Arkin, 1998). This thesis shows that stochastic simulation has a much wider range of applications by applying the methodology to a larger system, namely a collection of cells, each with a much more complicated network containing more molecular species than phage λ.

## *A Caveat Concerning Modeling*

With all these attempts to model a biological system, it is important to keep track of the goals and the pitfalls of modeling in general. This is most succinctly put in an article concerning the nature of numerical modeling in the earth sciences, but the nature of the arguments apply to any field in which models are created.

> Verification and validation of numerical models of natural systems is impossible. This is because natural systems are never closed and because model results are always nonunique. Models can be confirmed by the demonstration of agreement between observation and prediction, but confirmation is inherently partial. Complete confirmation is logically precluded by the fallacy of affirming the consequent and by incomplete access to natural phenomena. Models can only be evaluated in relative terms, and their predictive value is always open to question. The primary value of models is heuristic. (Oreskes et al., 1994)

This situation is clearly illustrated in this thesis. The *Hox* network model was constructed using the relevant biochemistry and biology, and the model results were in good agreement with the published laboratory experiments. When a new experiment was performed to test an implementation decision of the model, it turned out that the model was not in agreement with the new experimental results. This resulted in a change to the model to fit the new experimental data, but the new simulation results were essentially indistinguishable from the original results. So while the new model must now be seen as better, in so far as it is consistent with more of the real data, there is unfortunately no guarantee that future predictions will match laboratory observations more closely. This is especially true given the incredibly dynamic nature of the system and the model.

Of course, these criticisms are valid for any model that seeks to describe a natural system, and so it is important to remember what models actually can do: they are useful in identifying parts of a problem that are in need of further study, and in identifying the

data that is relevant to the problem at hand. Furthermore, the very act of constructing a model can stimulate questions about how the natural system behaves. In this instance, the questions lead to the retinoic acid soaked bead experiment described in Chapter 4. The resulting data adds to the understanding of the connection between retinoic acid and the gene *Hoxa1*, in particular, and the network of genes patterning the brain in general.

## *References for Chapter 1*

Arkin, A., Ross, J., and McAdams, H. H. (1998). Stochastic kinetic analysis of developmental pathway bifurcation in phage lambda-infected Escherichia coli cells. *Genetics* **149,** 1633-48.

Barrow, J. R., Stadler, H. S., and Capecchi, M. R. (2000). Roles of Hoxa1 and Hoxa2 in patterning the early hindbrain of the mouse. *Development* **127,** 933-44.

Bray, D., Firth, C., Le Novere, N., and Shimizu, T. (2001). StochSim.

Friedman, N., Linial, M., Nachman, I., and Pe'er, D. (2000). Using Bayseian networks to analyze expression data. *J. Comput. Biol.* **7,** 601-620.

Gillespie, D. T. (1976). A General Method for Numerically Simulating the Stochastic Time Evolution of Coupled Chemical Reactions. *Journal of Computational Physics* **22,** 403.

Greenwald, I. (1998). LIN-12/Notch signaling: lessons from worms and flies. *Genes Dev* **12,** 1751-62.

Greer, J. M. a. C., M.R. (2002). *Hoxb8* Is Required for Normal Grooming Behavior in Mice. *Neoron* **33,** 23-34.

Hynne, F., Danø, S., and Sørensen, P. G. (2001). Full-scale model of glycolysis in *Saccharomyces cerevisiae*. *Biophysical Chemistry* **94,** 121-163.

Ingram, J. L., Stodgell, C. J., Hyman, S. L., Figlewicz, D. A., Weitkamp, L. R., and Rodier, P. M. (2000). Discovery of allelic variants of HOXA1 and HOXB1: genetic susceptibility to autism spectrum disorders. *Teratology* **62,** 393-405.

Kastner, J. C., Solomon, J. E., and Fraser, S. E. (2002). Modeling a Hox Gene network *in silico* using a Stochastic Simlulation Algorithm. *Developmental Biology* **246,** 122-131.

Kauffman, S. A. (1993). "The Origins of Order." Oxford University Press, Oxford.

Kierzek, A. M. (2002). STOCKS: STOChastic Kinetic Simulations of biochemical systems with gillespie algorithm. *Bioinformatics* **18,** 470-481.

Ko, M. S. (1992). Induction mechanism of a single gene molecule: stochastic or deterministic? *Bioessays* **14,** 341-6.

Lewis, M. T. (2000). Homeobox genes in mammary gland development and neoplasia. *Breast Cancer Res*. **2,** 158-169.

McAdams, H. H., and Arkin, A. (1998). Simulation of prokaryotic genetic circuits. *Annu. Rev. Biophys. Biomol. Struct*. **27,** 199-224.

McQuarrie, D. A. (1967). Stochastic Approach to Chemical Kinetics. *Journal of Applied Probability* **4,** 413-478.

Meinhardt, H., and de Boer, P. A. J. (2001). Pattern formation in *Escherichia coli*: A model for the pole-to-pole oscillations of Min proteins and the localization of the division site. *PNAS* **98,** 14202-14207.

Meyers, S., and Friedland, P. (1984). Knowledge-based simulation of genetic regulation in bacteriophage lambda. *Nucleic Acids Res*. **12,** 1-9.

Oreskes, N., Shrader-Frechette, K., and Belitz, K. (1994). Verification, Validation, and Confirmation of Numerical Models in the Earth Sciences. *Science* **263,** 641-646.

Pierce, S. J. (1999). Boundary crossing in research literatures as a means of interdisciplinary information transfer. *Journal of the American Society for Information Science* **50,** 271-279.

Poolman, M. G., Ölçer, H., Lloyd, J. C., Raines, C. A., and Fell, D. A. (2001). Computer modelling and experimental evidence for two steady states in the photosynthetic Calvin cycle. *Eur. J. Biochem*. **268,** 2810-2816.

Rodier, P. M. (2000). The early origins of autism. *Sci. Am*. **282,** 56-63.

Thompson, D. A. W. (1942). "On Growth and Form." Dover, Mineola, New York.

Thorsteinsdottir, U., Kroon, E., Jerome, L., Blasi, F., and Sauvageau, G. (2001). Defining roles for HOX and MEIS1 genes in induction of acute myeloid leukemia. *Mol. Cell. Biol*. **21,** 224-234.

Yuh, C. H., Bolouri, H., and Davidson, E. H. (1998). Genomic cis-regulatory logic: experimental and computational analysis of a sea urchin gene. *Science* **279,** 1896-902.

Yuh, C. H., Bolouri, H., and Davidson, E. H. (2001). Cis-regulatory logic in the endo16 gene: switching from a specification to a differentiation mode of control. *Development* **128,** 617-29.

Zlokarnik, G., Negulescu, P. A., Knapp, T. E., Mere, L., Burres, N., Feng, L., Whitney, M., Roemer, K., and Tsien, R. Y. (1998). Quantitation of transcription and clonal selection of single living cells with beta-lactamase as reporter. *Science* **279,** 84-8.

# Chapter 2: Modeling Enzyme Kinetics

> From a mathematical point of view, the art of good modeling relies on: (i) a sound understanding and appreciation of the biological problem; (ii) a realistic mathematical representation of the important biological phenomena; (iii) finding useful solutions, preferably quantitative; and most crucially important, (iv) a biological interpretation of the mathematical results in terms of insights and predictions. The mathematics is dictated by the biology and not vice versa. Sometimes the mathematics can be very simple. Useful mathematical biology research is not judged by mathematical standards but by different and no less demanding ones.
>
> - Jim Murray, 1993

## *Introduction*

When investigating a novel method, it is often very useful to use a small example, or "toy problem," to examine its workings before jumping into a larger problem. As mentioned previously, these simplified problems are hard to come by in biology but there is a toy problem at the heart of the simulation, namely, enzymatic reactions. This chapter contains a description of the basic enzyme reaction first described by Michaelis and Menten in 1913, as well as a comparison between the results of the deterministic solution and the stochastic solution. This problem was chosen for a variety of reasons. Firstly, this problem contains much of the basics of enzymatic biology in its midst. Secondly,

this is one of the few problems for which a detailed solution can be constructed. Finally, it affords a readily understandable introduction to the stochastic simulation.

In the case of the deterministic solution, perturbation theory will be used to provide an approximate solution, but because any modern computer algebra system will be able to easily provide a numerical solution to the resulting differential equations, that will be included as well. To solve this problem using a stochastic solution, a short program using the *Mathematica* programming language has been developed. This will allow a detailed example of the stochastic simulation algorithm. By comparing the solutions from the deterministic methods to the stochastic simulation solutions, it will be shown that they are good agreement with each other in a global sense. However, it will also be shown that there are situations in which the deterministic solution may not capture the true state of the system.

## *Deterministic Solution*

The theory for chemical kinetics in a large volume is well grounded in experiments. Early forms of the Law of Mass Action, which states that the rate of the reaction is proportional to the concentration of the reactants, appeared at least as early as 1802 with Berthollet's nearly correct formulations. The final correct formulation came from extensive experiments that Waage and Gulberg published in 1864 (Waage, 1986). But an important piece of the puzzle was still missing, and it was another 25 years before the discovery of the process that allowed some molecules to react while others remained inactive.

In1889, while investigating an offshoot of his work on ionic solutions (work that would eventually win him a Nobel Prize), Svante Arrhenius studied the effects temperature has on the rate of a reaction. His data led him to conclude that in a reaction system only a certain number of molecules are able to react at any given time. He proposed that some sort of chemical catalyst must have activated the molecules that are able to react. His theory said that the catalyst ($C$) would first form an intermediate compound ($CS$) with the substrate ($S$), and the resulting compounds are then able to enter a transition state that lowers the amount of energy that is needed to perform a chemical reaction. The compound then decomposes into a product ($P$) and the catalyst ($C$), and the catalyst is then free to participate in another reaction:

$$C + S \Rightarrow CS, CS \Rightarrow P + C \tag{2.1}$$

Thus, the notion of activation energy for a chemical reaction was born (Teich, 1992).

Two decades later, Michaelis and Menten published a seminal piece of work on how this type of system behaves. In their paper, they focused on a biological system that has come to be known as the basic enzyme reaction (Michaelis and Menten, 1913). It was very similar to Arrhenius system but with the addition of a backwards reaction (disassociation) of the complex ($ES$). There was also a terminology change from catalyst ($C$) to enzyme ($E$). This was just a minor change as an enzyme is defined to be an organic catalyst. Schematically, this can be represented by

$$E + S \underset{k_{-1}}{\overset{k_1}{\Leftrightarrow}} ES, ES \overset{k_2}{\Rightarrow} P + E\,. \tag{2.2}$$

In words, one molecule of the enzyme combines with one molecule of the substrate to form one molecule of the complex. The complex can disassociate into one molecule of

each of the enzyme and substrate, or it can produce a product and a recycled enzyme. In this formulation $k_1$ is the rate parameter for the forward substrate/enzyme (catalyst), $k_{-1}$ is the rate parameter for the backwards reactions, and $k_2$ is the rate parameter for the creation of the product. There is no backwards reaction forming the complex from the product and the enzyme, as it is assumed that this reaction is energetically unfavorable and the enzyme is much more likely to participate with the substrate in the formation of the complex. Given an initial amount (or concentrations) of the reactants and the rate parameters, the question is to determine the amount of product at some later time.

Using the Law of Mass Action, it is possible to write down the change in the amount of each of the reactants, leading to one differential equation for each of the reactants. The fact that it may not adequately capture the true state of small systems is a problem that will be addressed shortly. The presentation of the basic enzyme reaction that follows draws from the conventional approaches (Edelstein-Keshet, 1998; Murray, 1993).

Denoting the concentrations in (2.2) by

$$e = [E], s = [S], c = [ES], p = [P] \tag{2.3}$$

the Law of Mass Action applied to this system leads to the following four differential equations that describe the kinetics of the basic enzyme reaction:

$$\begin{aligned} \frac{ds}{dt} &= -k_1 es + k_{-1} c, \quad \frac{de}{dt} = -k_1 es + (k_{-1} + k_2) c \\ \frac{dc}{dt} &= k_1 es - (k_{-1} + k_2) c, \quad \frac{dp}{dt} = k_2 c \end{aligned} . \tag{2.4}$$

As the system starts with only the substrate and enzymes, the initial conditions are then

$$e(0) = e_0,\ s(0) = s_0,\ c(0) = 0,\ p(0) = 0. \tag{2.5}$$

Before solving this system, it is important to note that the equations are not all independent. First of all, given a fixed amount of enzyme, it is possible to write down a conservation law by noting that the amount of free enzyme and bound enzyme must be constant:

$$e(t) + c(t) = e_0\ . \tag{2.6}$$

Combining this back into the first three differential equations, it is possible to eliminate one to end up with

$$\frac{ds}{dt} = -k_1 e_0 s + (k_1 s + k_{-1} c), \quad \frac{dc}{dt} = k_1 e_0 s - (k_1 s + k_{-1} + k_2)c\ , \tag{2.7}$$

with the initial conditions

$$s(0) = s_0, c(0) = 0\ . \tag{2.8}$$

Finally, the equation for the product can be uncoupled from the others, and integration leads to

$$p(t) = k_2 \int_0^t c(u)\,du, \tag{2.9}$$

which provides the solution for the product once the solution for the complex is known. The end result is a reduction of the set of four differential equations into two coupled ones.

As the situation under consideration is one where there are a small number of enzymes compared to the number of substrate molecules available, let

$$\varepsilon = \frac{e_0}{s_0}, \tag{2.10}$$

which leads to using the following variables to nondimensionalize the equations

$$\tau = k_1 e_0 t, \quad u(\tau) = \frac{s(t)}{s_0}, \quad v(\tau) = \frac{c(t)}{e_0}, \quad \lambda = \frac{k_2}{k_1 s_0}, \quad K = \frac{k_{-1} + k_2}{k_1 s_0}. \tag{2.11}$$

Then the system in (2.7) becomes

$$\frac{du}{d\tau} = -u + (u + K - \lambda)v, \quad \varepsilon \frac{dv}{d\tau} = u - (u + K)v, \tag{2.12}$$

with the initial conditions

$$u(0) = 1, \quad v(0) = 0. \tag{2.13}$$

In looking for a solution for this problem, the appearance of the small parameter $\varepsilon$ in front of a derivative in (2.12) suggests that this is a singular perturbation problem, and looking for a single regular Taylor series expansion solution in terms of the variables $u,v$ and $\varepsilon$ will not be fruitful. Because of this, it is necessary to create a multiscale solution from matching inner and outer solutions. This can be accomplished by first looking for the regular Taylor expansion solution in the form

$$u(\tau;\varepsilon) = \sum_{n=0} \varepsilon^n u_n(\tau), v(\tau;\varepsilon) = \sum_{n=0} \varepsilon^n v_n(\tau). \tag{2.14}$$

Substituting this into (2.12) and equating like powers of $\varepsilon$ yields for the $O(1)$ system

$$\frac{du_0}{d\tau} = -u_0 + (u_0 + K - \lambda)v_0, \quad 0 = u_0 - (u_0 + K)v_0, \tag{2.15}$$

with the initial conditions

$$u_0(0) = 1, \quad v_0(0) = 0. \tag{2.16}$$

At this point the problem with this type of solution is clear; the second equation does not satisfy the initial condition. This will be taken care of later when the outer and inner solutions to the system are matched. Plunging ahead and solving this system leads to

$$v = \frac{u_0}{u_0 + K}, \frac{du_0}{d\tau} = -\lambda \frac{u_0}{u_0 + K}, \tag{2.17}$$

and therefore

$$u_0(\tau) + K \ln\left(u_0(\tau)\right) = A - \lambda\tau, v_0(\tau) = \frac{u_0(\tau)}{u_0(\tau) + K}. \tag{2.18}$$

In searching for an inner solution, define

$$\sigma = \frac{\tau}{\varepsilon}, U(\sigma;\varepsilon) = u(\tau;\varepsilon), V(\sigma;\varepsilon) = v(\tau;\varepsilon), \tag{2.19}$$

then with these transformations the system in (2.12) becomes

$$\frac{dU}{d\sigma} = -\varepsilon U + \varepsilon(U + K - \lambda)V, \frac{dV}{d\sigma} = U - (U + K)V, \tag{2.20}$$

with the initial conditions

$$U(0) = 1, \quad V(0) = 0. \tag{2.21}$$

The system no longer has the small parameter $\varepsilon$ multiplying a derivative term, and therefore it is possible to look for a solution in terms of a regular perturbation expansion

$$U(\sigma;\varepsilon) = \sum_{n=0} \varepsilon^n U_n(\sigma), V(\sigma;\varepsilon) = \sum_{n=0} \varepsilon^n V_n(\sigma). \tag{2.22}$$

Substituting this expansion into (2.20) and setting $\varepsilon = 0$ yields the $O(1)$ system

$$\frac{dU_0}{d\sigma} = 0, \frac{dV}{d\sigma} = U_0 - (U_0 + K)V_0, \tag{2.23}$$

with the initial conditions

$$U_0(0) = 1, \quad V_0(0) = 0. \tag{2.24}$$

The solutions of this inner system are then found to be

$$U_0(\sigma) = B, V_0(\tau) = \frac{B}{B+K} + C\exp\left[-\tau(K+B)\right]. \tag{2.25}$$

At this point, all that is left is to match the solutions. Using the initial conditions and requiring that

$$\lim_{\sigma\to\infty} U_0(\sigma) = \lim_{\tau\to 0} u_0(\tau) \text{ and } \lim_{\sigma\to\infty} V_0(\sigma) = \lim_{\tau\to 0} v_0(\tau) \tag{2.26}$$

results in $A = 1, B = 1, C = \frac{-1}{1+K}$. The resulting multiscale solution then correctly matches as the respective limits are

$$\lim_{\sigma\to\infty} U_0(\sigma) = \lim_{\tau\to 0} u_0(\tau) = 1 \text{ and } \lim_{\sigma\to\infty} V_0(\sigma) = \lim_{\tau\to 0} v_0(\tau) = \frac{1}{1+K}. \tag{2.27}$$

The $O(1)$ solution to the inner system is

$$U_0(\sigma) = 1, V_0(\tau) = \frac{1-\exp\left[-(1+K)\sigma\right]}{1+K}, \tag{2.28}$$

while the $O(1)$ solution to the outer system is

$$u_0(\tau) + K\ln\left(u_0(\tau)\right) = 1 - \lambda\tau, v_0(\tau) = \frac{u_0(\tau)}{u_0(\tau)+K}. \tag{2.29}$$

In practice it is extremely difficult, if not impossible, to construct even approximate solutions to a system that contains any more reactions than the Michaelis-Menten problem and numerical methods must be used (McQuarrie, 1967).

As shown above, the Law of Mass Action applied to the basic enzyme reaction leads to a set of coupled differential equations that can be approximated using perturbation theory, and the differential equations are easily solved numerically as well. Because the Law of Mass Action is not only well grounded in experiments but also leads to equations that can be readily solved. But while differential equations are a natural way

to model chemical reactions in a vat, they might not adequately represent the true state of the system in a cell.

Implicit in using the Law of Mass Action are two key assumptions that should be mentioned: continuity and determinism. With regards to the continuity assumption, it is important to note that the individual genes are often only present in one or two copies per cell. Therefore, there are only one or two regulatory regions to which the regulatory molecules can bind. In addition, the regulatory molecules that bind to these regions are typically produced in low quantities: there may be only a few tens of molecules of a transcription factor in the cell nucleus. This has been shown explicitly in bacterial cells, but there is ample evidence supporting this fact in eukaryotic cells as well (Davidson, 1986; Guptasarma, 1995). The low number of molecules may compromise the notion of continuity.

As for determinism, the rates of some of these reactions are so slow that many minutes may pass before, for instance, the start of mRNA transcription after the necessary molecules are present, or between the start and finish of mRNA creation (Davidson, 1986). This may call into question the notion of the deterministic change presupposed by the use of the differential operator due to the fluctuations in the timing of cellular events. As a consequence, two regulatory systems having the same initial conditions might ultimately settle into different states, a phenomenon strengthened by the small numbers of molecules involved.

There have been some recent experimental results that strongly suggest that cells do in fact behave stochastically. A review can be found in a recent article by the pioneers

of modeling stochastic processes in biology, and they drive home the point that regulatory molecules are present in very low concentrations in cells, with a few hundred being an upper limit, and dozens being a normal phenomenon (McAdams and Arkin, 1999). A study of these systems has shown that the stochastic fluctuations in such a system can produce erratic distributions in protein levels between the same type of cell in a population (McAdams and Arkin, 1997). This is especially true when the molecule under investigation is part of the regulatory mechanism of the cell (Arkin et al., 1998). Most recently, a study in yeast has produced intriguing data concerning the noise in a biological system due to the intrinsic fluctuations (Elowitz et al., 2002).

When the fluctuations in the system are small, it is possible to use a reaction rate equation approach. But when fluctuations are not negligibly small, the reaction rate equations will give results that are at best misleading (showing only the mean behavior), and possibly very wrong if the fluctuations can give rise to important effects. The real problem arises in that it is not always known beforehand whether fluctuations are important. The only way to find out is to use a stochastic simulation: If several stochastic trajectories give results that appear to be identical, then reaction rate equations could indeed have been used. But if the differences in the trajectories were noticeable, then reaction rate equations probably would not have been appropriate. It is possible to forge ahead, and the result is usually a mathematical model that describes the phenomena, but fails to capture the fluctuations present in the system.

Some of the concerns about fluctuations in a system have been around for a long time, if only in theory. With regards to the number of molecules in a cell, this was first mentioned in the English literature by the biochemist J. B. S. Haldane when he

mentioned that critical processes might be carried out by one of a few enzymes per cell (Haldane, 1930). Fifteen years later, this was repeated as a known fact in Nature (McIlwain, 1946). More recently there appeared a paper on the question of whether the laws of chemistry apply to living cells (Halling, 1989). It isn't quite as elegant as Purcell's paper on life at low Reynolds numbers (Purcell, 1977), but like this famous talk, the paper points out that it is a very different world inside a cell.

Consequently, the fluctuations in the system may actually be an important part of the system. With these concerns in mind, it seems only natural to investigate an approach that incorporates the small volumes and small number of molecular species (and the inherent fluctuations that are present in a system) and may actually play an important part. These investigations are still relatively new, but in recent years the stochastic simulation algorithm has been used to model phage λ infected *E. coli* cells (Arkin et al., 1998), and calcium wave propagation in rat hepatocytes (Gracheva et al., 2001).

## *Stochastic Solution*

The first mention of using stochastic methods to model chemical reactions appeared in 1940 (Delbruck, 1940; Kramers, 1940). But it wasn't until the early 1950s that it became clear that in small systems the Law of Mass Action breaks down (Renyi, 1954) and even small fluctuations in the number of molecules may be a significant factor in the behavior of the system (Singer, 1953). Soon after, it became evident that some processes in biological cells fell into this category and that a proper mathematical formulation of the chemical reactions in the cells will most likely be based on stochastics (Bartholomay, 1958).

The stochastic approach considers the sets of possible reactions and examines the possible transitions of the system. As an example, consider the following irreversible unimolecular reaction

$$\mathbf{A} \xrightarrow{k} \mathbf{B}, \tag{2.30}$$

which is common in radioactive decay processes. In words, the molecule A is converted to B with rate parameter *k*. The stochastic description of the system is characterized in the following manner. Let $X(t)$ be a random variable that denotes the number of **A** molecules at time *t*. Then

*1)* The probability of a transition from $(x+1)$ molecules to $(x)$ molecules in the interval $(t,t+\Delta t)$ is $k(x+1)\Delta t + o(\Delta t)$. $k$ is the rate constant and $o(\Delta t)$ takes the usual meaning that $o(\Delta t)/\Delta t \to 0$ as $\Delta t \to 0$.

*2)* The probability of a transition from $(x)$ to $(x-j), j>1$ in the interval $(t,t+\Delta t)$ is $o(\Delta t)$.

*3)* The probability of a transition from $(x)$ to $(x+j), j \geq 1$ in the interval $(t,t+\Delta t)$ is zero.

Denoting the probability of $X(t)=x$ by $P_x(t)$, a balance of the terms yields

$$P_x(t+\Delta t) = k(x+1)\Delta t P_{x+1}(t) + (1-kx\Delta t)P_x(t) + o(\Delta t). \tag{2.31}$$

Simplifying and taking the limit $\Delta t \to 0$ yields the differential-difference equation

$$dP_x(t)/dt = k(x+1)P_{x+1}(t) - kP_x(t), \tag{2.32}$$

which is also called the chemical master equation for the system.

The solution of the chemical master equation can be thought of as a Markovian random walk in the space of the reacting variables. It measures the probability of finding

the system in a particular state at any given time, and it can be rigorously derived from a microphysical standpoint (Gillespie, 1992). Analytic solutions of master equations are difficult to come by, but in this example it is possible to transform the differential-difference equation into a partial differential equation through the use of the probability generating function

$$F(s,t) = \sum_{x=0}^{\infty} P_x(t) s^x . \tag{2.33}$$

Substituting (2.33) into (2.32) and simplifying leads to

$$\frac{\partial F}{\partial t} = k(1-s)\frac{\partial F}{\partial s}. \tag{2.34}$$

Given the initial condition $F(s,0) = s^{x_0}$, the solution is then

$$F(s,t) = \left[1 + (s-1)e^{-kt}\right]^{x_0}. \tag{2.35}$$

Recall that if $X(t)$ is a random variable, then $E[X(t)]$, the expected value, is defined as $\sum x P_t(x)$ which is, conveniently enough, $\left.\frac{\partial F}{\partial s}\right|_{s=1}$. Computing this value leads to

$$E\{X(t)\} = x_0 e^{-kt}, \tag{2.36}$$

which is the solution of the Mass Action formulation for the system:

$$\frac{dA}{dt} = -kA . \tag{2.37}$$

Thus, the two representations are consistent. However, this is only true in general for unimolecular reactions (McQuarrie, 1967).

Historically, numerical methods were used to construct solutions to the master equations, but the solutions constructed in this manner have some pitfalls. These include

the need to approximate higher-order moments as a product of lower moments, and convergence issues (McQuarrie, 1967). What was needed was a general method that would solve these sorts of problems and this came with the stochastic simulation algorithm.

## *Stochastic Simulation Algorithm*

Given a set of molecular species $\{S_\mu\}_{\mu=1}^{N}$ and a set of reactions in which they can participate $\{R_\mu\}_{\mu=1}^{N}$, the Gillespie algorithm, as it has come to be known, is an exact method for numerically computing the time evolution of a chemical system. By exact it is meant that the results are provably equivalent to the chemical master equation, but at no time is it necessary for the master equation to be written down, much less solved.

The fundamental hypothesis of the method is that the reaction parameter $c_\mu$ associated with the reaction $R_\mu$ can be defined in the following manner:

$c_\mu \delta t \equiv$ the average probability, to the first order in $\delta t$, that a particular combination $R_\mu$ of reactant molecules will react in the next time interval $\delta t$.

In his original work, Gillespie shows that this definition does in fact have a valid physical basis and in fact the reaction parameter $c_\mu$ can be easily connected to the traditional reaction rate constant $k_\mu$ (Gillespie, 1976).

The method is based on the joint probability density function $P(\tau,\mu)$, defined by

$P(\tau,\mu)d\tau \equiv$ the probability at time *t* that the next reaction will occur in the differential time interval $(t+\tau, t+\tau+d\tau)$ and will be of type $R_\mu$.

This is a departure from the usual stochastic approach that starts from the probability function $P(X_1,X_2,\ldots,X_N;t)$, defined as the probability that at time t there will be $X_1$ molecules of $S_1$, $X_2$ molecules of $S_2$, …, and $X_N$ molecules of $S_N$. By using $P(\tau,\mu)$ as the basis of the approach, it is possible to create a tractable method to compute the time evolution of the system. To construct a formula for this quantity, Gillespie starts by defining the quantity $h_\mu$ as the number of distinct molecular reactant combinations for the reaction $R_\mu$. This is nothing more than a combinatorial factor and Table 2.1 lists some example values.

| Reaction | $h_\mu$ | Reaction order |
|---|---|---|
| $* \rightarrow \mathbf{S}_j$ | 1 | Zeroth |
| $\mathbf{S}_j \rightarrow \mathbf{S}_k$ | $\mathbf{X}_j$ | First |
| $\mathbf{S}_j + \mathbf{S}_k \rightarrow \mathbf{S}_l$ | $\mathbf{X}_j \cdot \mathbf{X}_k$ | Second |
| $\mathbf{S}_j + \mathbf{S}_j \rightarrow \mathbf{S}_k$ | $\mathbf{X}_j\left(\mathbf{X}_j - 1\right)/2$ | Second |
| $\mathbf{S}_i + \mathbf{S}_j + \mathbf{S}_j \rightarrow \mathbf{S}_k$ | $\mathbf{X}_i\mathbf{X}_j\left(\mathbf{X}_j - 1\right)/2$ | Third |

**Table 2.1 Appropriate combinatorial factors for various reactions**. In actuality, everything can be thought of as a zeroth-, first-, or second-order reaction, or a sequential combination of these, and there is no need for the higher-order reactions.

Combining this definition of $h_\mu$ with the previous definition for the reaction parameter $c_\mu$, leads to the conclusion that the probability, to the first order in $\delta t$, that a $R_\mu$ reaction will occur in the next time interval time $\delta t$ is therefore

$$h_\mu c_\mu \delta t. \tag{2.38}$$

Now $P(\tau,\mu)d\tau$ can be computed as the product of $P_0(\tau)$, the probability that no reaction occurs in the time interval $(t,t+\tau)$, and $h_\mu c_\mu \delta t$, the probability that the specific reaction $R_\mu$ occurs in the next time interval $(t+\tau,t+\tau+d\tau)$:

$$P(\tau,\mu)d\tau = P_0(\tau)h_\mu c_\mu d\tau. \tag{2.39}$$

All that is now required is to calculate the term $P_0(\tau)$. To construct an expression for this term, divide the interval $(t,t+\tau)$ into *K* subintervals, each of length $\varepsilon = \tau/K$. The probability that none of the reactions $\{R_\mu\}_{\mu=1}^{N}$ occurs in the time interval $(t+j\varepsilon,t+j\varepsilon+1)$ (for any arbitrary j) is

$$\prod_{i=1}^{M}\left[1-h_i c_i \varepsilon + o(\varepsilon)\right] = 1-\sum_{i=1}^{M} h_i c_i \varepsilon + o(\varepsilon). \tag{2.40}$$

Since there are K subintervals and the probabilities are mutually exclusive,

$$P_0(\tau) = \left[1-\sum_{i=1}^{M} h_i c_i \frac{\tau}{K} + o\left(\frac{\tau}{K}\right)\right]^K. \tag{2.41}$$

But as this expression is valid for any K, even infinitely large ones, the expression can also be written as

$$P_0(\tau) = \lim_{K\to\infty}\left[1-\left(\sum_{i=1}^{M} h_i c_i \tau + o\left(K^{-1}\right)/K^{-1}\right)\Big/K\right]^K. \tag{2.42}$$

However, this is nothing more than one of the limit formulas for the exponential function, and thus

$$P_0(\tau) = \exp\left(-\sum_{i=1}^{M} h_i c_i \tau\right). \tag{2.43}$$

Therefore, after defining

$$a_\mu \equiv h_\mu \cdot c_\mu, a_o \equiv \sum_{i=1}^{M} h_i \cdot c_i \,, \tag{2.44}$$

the result is an expression for $P(\tau,\mu)$:

$$P(\tau,\mu) = a_\mu \exp[-a_0 \tau]. \tag{2.45}$$

## Implementation

This algorithm can easily be implemented in an efficient modularized form to accommodate quite large reaction sets of considerable complexity.

For an easy implementation, the joint distribution can be broken into two disjoint probabilities using Bayes' rule:

$$P(\tau,\mu) = P(\tau) \cdot P(\mu|\tau) \,. \tag{2.46}$$

But note that the addition property for probabilities can be used to calculate an alternate form for $P(\tau)$:

$$P(\tau) = \sum_{\mu=1}^{M} P(\tau,\mu), \tag{2.47}$$

and substituting this into (2.45) leads to values for its component parts:

$$P(\tau) = a_0 \exp(-a_0 \tau) \,, \tag{2.48}$$

$$P(\mu|\tau) = \frac{a_\mu}{a_0} \,. \tag{2.49}$$

Given these fundamental probability density functions, the following algorithm can be used to carry out the reaction set simulation:

1) Initialization

a. Set values for the $c_\mu$.

b. Set the initial number of the $S_\mu$ reactants.

c. Set $t = 0$, and select a value for $t_{\max}$, the maximum simulation time.

2) Loop

a. Compute $a_\mu \equiv h_\mu \cdot c_\mu, a_o \equiv \sum_{i=1}^{M} h_i \cdot c_i$.

b. Generate two random numbers $r_1$ and $r_2$ from a uniform distribution on $[0,1]$.

c. Compute the next time interval $\tau = \frac{1}{a_0} \ln\left(\frac{1}{r_1}\right)$ (Draw from the probability density function of (2.48)).

d. Select the reaction to be run by computing $\mu$ such that $\sum_{\nu=1}^{\mu-1} a_\nu < r_2 a_0 \leq \sum_{\nu=1}^{\mu} a_\nu$ (Draw from the probability density function of (2.49)).

e. Adjust $t = t + \tau$ and update the $S_\mu$ values according to the $R_\mu$ reaction that just occurred.

f. If $t > t_{\max}$, then terminate. Otherwise, goto a.

Because the speed of the SSA is linear with respect to the number of reactions, adding new reaction channels will not greatly increase the runtime of the simulation *i.e.*, doubling either the number of reactions or the number of reactant species doubles (approximately) the total runtime of the algorithm. The speed of the SSA depends more on the number of molecules. This is seen by noting that the computation of the next time

interval in (2c) above depends on the reciprocal of $a_0$, a term comprised of, among other things, the number of molecules in the simulation. If the reaction set contains at least one second-order reaction, then $a_0$ will contain at least one product of species population. In this case the speed of the simulation will fall off like the reciprocal of the square of the population. However, the runtime can be reduced by noting that not all of the $a_\mu$ values will need to be recalculated after each pass, but only the ones for which $S_\mu$ appears as a reactant in the $R_\mu$ reaction. An efficient implementation will take advantage of this fact.

Recent improvements to the algorithm, including a method that does not require the probabilities to be updated after every reaction, are helping to keep the runtime in check (Gibson and Bruck, 2000; Gillespie, 2001). As currently implemented, a typical run of the *Hox* simulation presented in Chapter 3 (without the aforementioned speedups) consists of over 23 million events, and takes less than 6 minutes on a computer with a 2GHz Pentium 4 processor.

Two important points should be noted about the SSA: the solution of a system of coupled chemical reactions by this method is entirely equivalent to the solution of the corresponding stochastic master equations (Gillespie, 1976; Gillespie, 1977c; McQuarrie, 1967), and in the limit of large numbers of reactant molecules, the results of this method are entirely equivalent to the solution of the traditional kinetic differential equations derived from the Law of Mass Action (Gillespie, 1977a).

One added benefit of the SSA is the formalism that is forces on the user. Each reaction in the set must be dealt with explicitly, and the connection between the reacting species (and the roles that they play in other reactions) must be clearly specified. The

fact that this algorithm generates its own (nonuniform) time sample should also be noted. Thus, as the simulation proceeds it generates time samples based on the probability density function of (2.43), *i.e.*, simulation time steps are based on draws from an exponential distribution. This of course is one of the reasons why this algorithm is so robust.

## Extensions

In order to apply the concepts involved in Gillespie's algorithm to a collection of cells, the original algorithm must be extended to accommodate the introduction of spatial dependencies of the concentration variables. Work has been done which extends the stochastic simulation algorithm to reaction-diffusion processes, and the modification to the method is straightforward. Diffusion is considered to be just another possible chemical event with an associated probability (Stundzia and Lumsden, 1996). As with all the other chemical events, the diffusion is assumed to be intracellular and the basic idea behind this approach is incorporated into the simulation. But one of the important molecules in the simulation is retinoic acid, an intercellular molecule that acts through cell surface receptors, and so the diffusion must be treated in a larger context.

Introducing a spatial context into the SSA is done by creating an interacting cell population represented as a rectangular array of square cells with nearest neighbor only cell-cell interactions. In this model of interacting cells, it is assumed that each cell is running its own internal program of biochemical reactions.

The fact that simulation of any given reaction generates its own “local” simulation time steps poses something of a problem for a model consisting of more than one cell,

each of which is running a reaction simulation independent of all the other cells. This problem arises when an intercellular event must be accounted for, since the internal simulation times of the two partner cells involved will not in general be the same. When implementing such simulations in serial code on a single-processor machine, converting the algorithm from what is essentially a spatial-scanning method to a temporal-scanning method can solve this problem. This is accomplished by first making an initial spatial scan through all of the cells in the array, and inserting the cells into a priority queue that is ordered from shortest to longest local cell time. All succeeding iterations are then based on the temporal order of the cells in the priority queue. In other words, a cell is drawn from the queue, calculations are performed on the reaction set for that cell, and then the cell is placed back on the queue in its new temporal-ordered position. By doing this there is no need to worry about synchronizing reaction simulations between any pair of neighboring cells.

Each reaction that occurs changes the quantity of at least one reactant. When this happens, the combinatorial factors $h_\mu$ change and it is necessary to recalculate the $a_\mu$ values. This is one of the drawbacks of the approach: if it weren't for having to recalculate the probabilities at every time step, the system is a Markov process with a fixed transition matrix and all standard analysis tools can be brought to bear. In general, only a small number of the $a_\mu$ will actually have to be updated and an efficient implementation needs to take advantage of this fact. After the $a_\mu$ values are updated, all cells that changed are reordered into their appropriate new position in the priority queue.

Because cells are stored as C-language structures, all of the information required to define the state of any given cell is readily available. The use of a priority queue to order the cells was a unique innovation, and solves the synchronizing problem inherent in a multicellular situation. Not only does this allow an easy mechanism for intercellular signaling, but this methodology can also readily accommodate local inhomogeneities in the molecular populations.

## *Comparison of the Approaches*

The programming language *Mathematica* was used to construct a numerical solution to the original set of differential equations in (2.4) and (2.5). *Mathematica* uses an Adams Predictor-Corrector method for non-stiff differential equations and backward difference formulas (Gear method) for stiff differential equations. It switches between the two methods using heuristics based on the adaptively selected step size. It starts with the non-stiff method, and checks for the advisability of switching methods every 10 or 20 steps. The result is an interpolating function that can be used to construct graphs of the solution for any time interval of interest.

*Mathematica* was also used to implement the stochastic simulation algorithm for the Michaelis-Menten basic enzyme reaction. This boiled down to a very short piece (less than 25 lines) of code and is included in Appendix D.

Plots of the trajectories of these two methods can be seen in Figure 2.1 below and the reader can easily see the differences between the stochastic and differential equation solutions to the basic enzyme reaction.

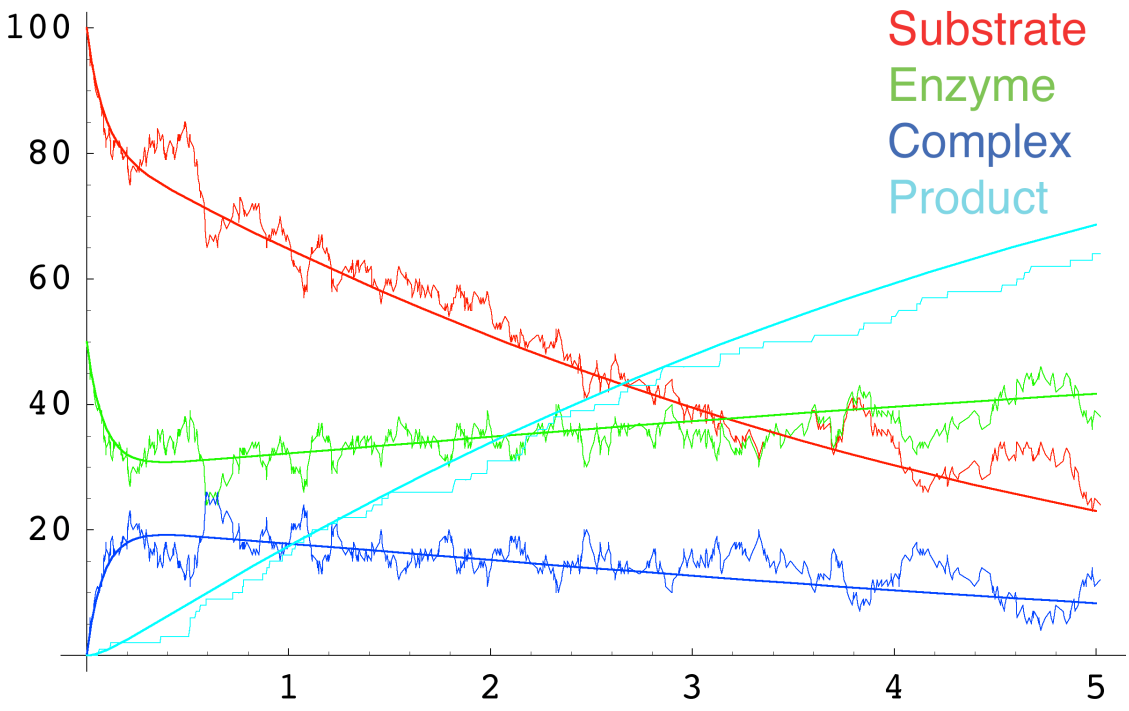


**Figure 2.1 Typical solutions of the Michaelis-Menten basic enzyme reaction, low numbers.** Both stochastic solution and differential equation solutions to the basic enzyme reaction are shown. The vertical axis is number of molecules and the horizontal axis is time in seconds. Notice that the fluctuations around the differential equations can range up to 50% of the solution when there are low quantities of molecules. The parameters used were $s_0$=100, $e_0$=50, $c_0$=$p_0$=0, $k_1$ = .005, $k_{-1}$ = 5.0, $k_2$ = 1.0.

"On average," the solutions are the same, but the stochastic approach captures the fluctuations in the system. Notice that there are some marked differences in these solutions. For instance, in the differential equation solution there are always fewer molecules of the complex than there are of the enzyme, but this is not true for the stochastic solution: at about .6 seconds the lines numbers coalesce. Another difference

can be seen in a comparison of the numbers of the substrate and the enzyme. Both the stochastic solution and the differential equation solution meet at about 3.2 seconds, but in the stochastic solution these quantities are very closely matched for the next .5 seconds while the differential equations solution quickly diverge.

But compare Figure 2.1 with Figure 2.2. The rate parameters have not been changed for this figure, only the starting numbers of substrate and enzyme. In this instance the reaction rate method and the stochastic method are in close agreement, both qualitatively and quantitatively.

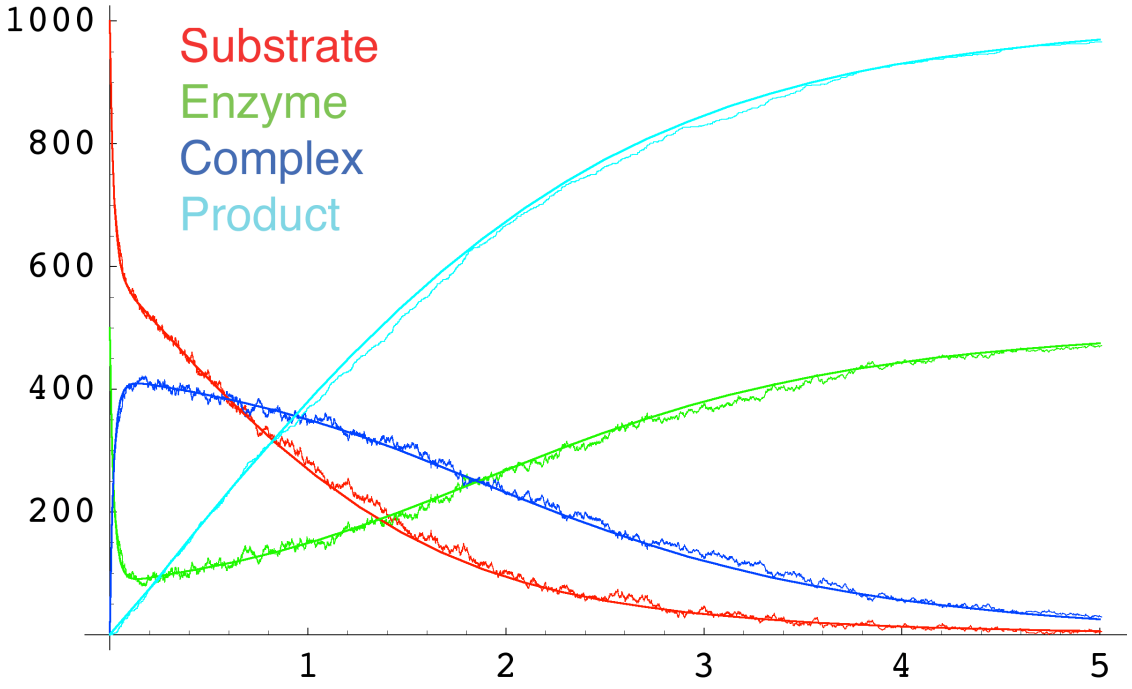


**Figure 2.2 Typical solutions of the Michaelis-Menten basic enzyme reaction, high numbers.** When there are a large number of molecules, the fluctuations are much less noticeable. The parameters used were $s_0$=1000, $e_0$=500, $c_0$=$p_0$=0, $k_1$ = .005, $k_{-1}$ = 5.0, $k_2$ = 1.0.

In comparing Figures 2.1 and 2.2, it is clear that when the number of molecules is large, the fluctuations might take the appearance of noise. But when there are small numbers of molecules, the fluctuations are may in fact no longer be just noise and may in fact be a significant part of the signal. Whether these fluctuations make a difference in the basic behavior of the system depends on the characteristics of that particular system. In the basic enzyme reaction the fluctuations do not matter, while in the *Notch-Delta* system described below they do. It may also be the case that the system moves between situations in which the fluctuations do and do not matter. Automatically detecting the need for a transition between these situations is part of an ongoing investigation (D. Gillespie, personal communication). However, when it is known that the system contains small numbers of molecules and the network is nonlinear—both of which are true for the *Hox* network—the stochastic approach appears to be a more appropriate method, because both of these situations will magnify any fluctuations that already exist in the system.

## *Notch-Delta Lateral Inhibition*

As previously mentioned, the SSA is an exact method (*i.e.,* the results are provably equivalent to the chemical master equation) for numerically computing the time evolution of a chemical system. It was also proved that in the limit of large numbers of reactant molecules, the results of the SSA method are consistent with the solution of the traditional kinetic differential equations derived from the Law of Mass Action (Gillespie, 1976; Gillespie, 1977c; McQuarrie, 1967). This is not surprising, because the first moment solution to the master equation describes the mean behavior of the system, just as the ODE solution does. But an interesting question concerns the practical connection

of these two methods: in practice, do the two different approaches yield similar results in a system that is sensitive to fluctuations? A related question is what exactly constitutes a large number of molecules.

These questions were explored by modeling lateral inhibition, the process by which a cell adopting a particular fate is able to prevent its neighbors from adopting the same fate. The *Notch-Delta* receptor-ligand pair is found to be involved in lateral inhibition in the cell fate specification in the developing nervous systems (Artavanis-Tsakonas et al., 1995; Chitnis, 1995). A simplified view of this process is shown in Figure 2.3 below.

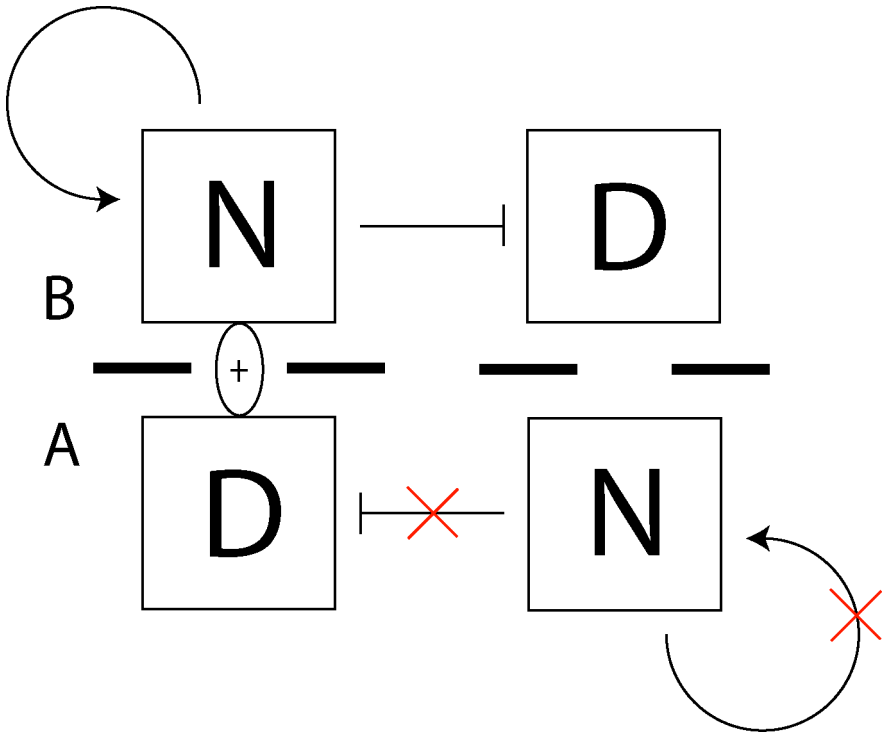


**Figure 2.3 *Notch-Delta* lateral inhibition.** When a *Delta* ligand in cell A binds (denoted by the plus inside the oval) to a *Notch* receptor in cell B, the *Notch*

undergoes a modification into an activated form. The activated *Notch* up-regulates *Notch* and down-regulates *Delta* in that cell. On the other hand, the down-regulation of *Delta* in cell B results in fewer *Notch* bindings in cell A. Because of this, activated *Notch* is not formed, and so *Notch* is not up-regulated and *Delta* is not down-regulated in cell A. The collection of events results in cell A becoming *Notch* dominant, and cell B becoming *Delta* dominant.

A study of the *Notch-Delta* lateral inhibition network using ODEs to model the network was undertaken a few years (Collier et al., 1996). The authors of the work examined three situations: a two-cell system, an infinite line, and a two-dimensional grid of cells. The former case was examined using phase plane analysis, while the latter cases were examined numerically using a Runge-Kutta-Merson method. In the two-cell system, they authors proved that if the feedback is sufficiently strong, one cell becomes *Notch* dominant and the other *Delta* dominant. The infinite line case was modeled using periodic boundary conditions, and the results were as expected; alternating *Notch* and *Delta* dominant cells

The two dimensional set of cells was much more interesting. Again there was a regular spatial periodicity to the cells, but they found that the results were very dependant on the boundary conditions. In particular, the default "checkerboard" solutions appeared only when the boundary conditions were compatible with the pattern, but not if the boundary conditions were not compatible with the pattern. This is one of the concerns with the ODE approach: The boundary conditions exert a very strong effect on the

system. Another concern is that the results are not nearly so regular in biological systems and it is known any cell can adopt the default fate (Greenwald, 1998). Finally, the model was heavily non-dimensionalized and caricatured, and the outputs of the model cannot be readily connected to number of molecules or concentrations (N. Monk, personal communication). Therefore, it seemed that a stochastic simulation of the system evolution might be enlightening.

A SSA model was built using the C programming language, and the complete source code can be found in Appendix D and the accompanying CD-ROM. The simulation consisted of 5 types of reactions (creation of *Notch* and *Delta*, decay of *Notch* and *Delta* and binding) and 5 species of molecules (*Notch* and *Delta* Protein, *Notch* and *Delta* mRNA and Activated *Notch*). The investigation was carried out in a 16-by-16 collection of rectangular cells with nearest neighbor communication. The binding between *Notch* and *Delta* required the use of a priority queue to efficiently synchronize the intercellular events. An example of a typical result is seen in Figure 2.4.

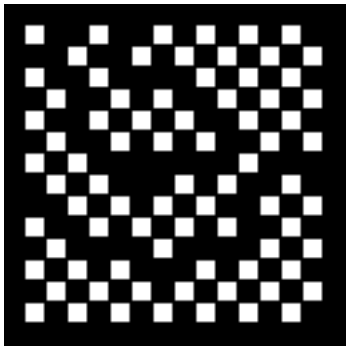

**Figure 2.4 *Notch-Delta* lateral inhibition typical results.** The white cells are *Notch* dominant, the black cells are *Delta* dominant. Each cell started with 500

molecules each of *Notch* and *Delta* protein, and the simulation was run until equilibrium was reached. Notice that while the cells show general pattern of alternating dominance, there is not strict compliance. This is reflective of the actual pattern of cells as seen in the *Drosophila* (Greenwald, 1998) and so the SSA seems to more accurately predict the observed behavior of cell fate determination then the deterministic approach.

While the stochastic model of *Notch-Delta* lateral inhibition seemed to show results that were consistent with the real cell fate, it was unclear if in the limit of large molecules the stochastic simulation would produce a more regular checkerboard, similar to the deterministic approach. It was also not clear what constitutes a large number of molecules in this case. Therefore, the number of proteins in each cell was increased from the default values of 500 molecules per cell, and results of these simulations are shown in the figures below. Figure 2.5 shows the results when binding does not go beyond the edge of the grid, while Figure 2.6 allows binding to wrap around the edge or the array.

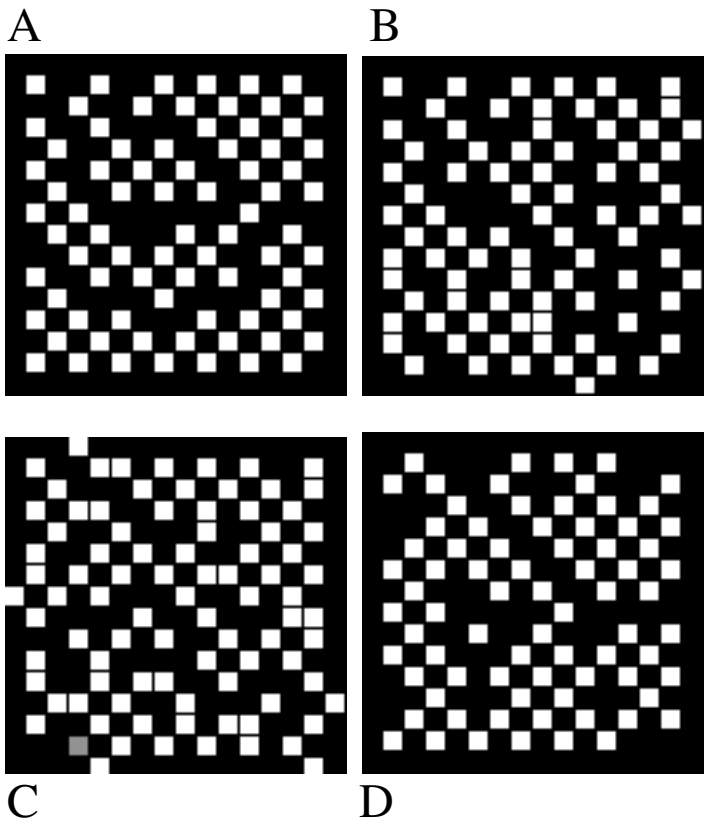


**Figure 2.5** ***Notch-Delta*** **larger number results, hard boundary.** The white cells are *Notch* dominant, the black cells are *Delta* dominant, and gray cells are ones in which neither is dominant. All input parameters except the starting number of molecules were as in **(A)** the default case of 500 molecules of *Notch* and *Delta* per cell **(B)** 1000 molecules per cell **(C)** 2500 molecules per cell **(D)** 5000 molecules per cell.

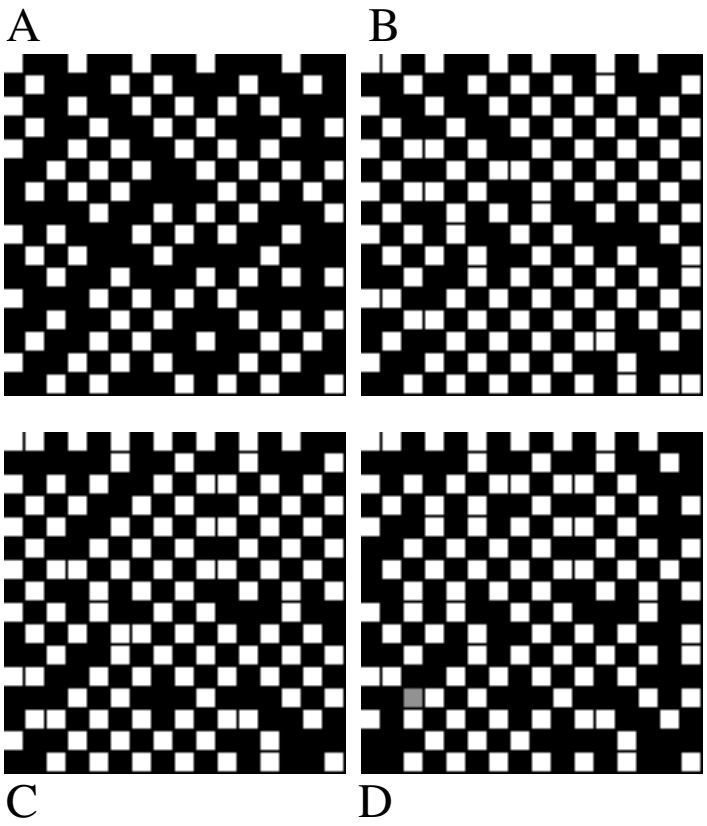


**Figure 2.6 *Notch-Delta* larger number results, wrap around binding.** Because the cells only communicate using nearest neighbor connections, the wrap around binding results in a torus of cells. **(A)** 500 molecules of *Notch* and *Delta* per cell **(B)** 1000 molecules per cell **(C)** 2500 molecules per cell **(D)** 5000 molecules per cell.

Just like in Figure 2.4, all of these results in Figures 2.5 and 2.6 show a general pattern of alternating cell dominance. In Figure 2.5, the number of molecules does not appear to play a role in the regularity of the pattern, but this may be related to the strong role the boundary plays: most of the cells on the edge are *Delta* dominant. Figure 2.6 is

much more interesting. In the larger number cases, the figures appear more regular. To quantify this, the following metric was calculated

$$m = \sum_{Notch \text{ cells}} \frac{\#\text{ of adjacent } Delta \text{ cells}}{4},$$

and the results are listed in Table 2.2.

| | Figure 2.5 | Figure 2.6 |
|---|---|---|
| A | 79 | 91 |
| B | 75.25 | 106.25 |
| C | 73 | 107 |
| D | 76 | 99.5 |

**Table 2.2 Regularity metric values.** The regularity metric quantifies the similarity to a perfect checkerboard. The maximum value possible is 128.

The larger numbers do not lead to a more regular pattern for the hard boundary case, while the metric for the torus (Figure 2.6) suggests that the larger numbers of molecules leads to a more regular pattern. For both of these cases however, it should be noted that 5000 molecules per cell is only 1250 of each type per face, and it is not clear that this is yet a "large" number of molecules. Unfortunately, with regards to the *Notch-Delta* simulation 5000 molecules per cell is approaching the practical upper limit of the capabilities of the stochastic simulation. Because one of the reactions is a binding between two different species of molecules, the $a_0$ value for this reaction contains a

product of terms, and so the speed of the SSA scales quadratically. In addition, the larger number of molecules means that the simulation takes longer to reach equilibrium. So while the results of Figure 2.5A took a little over an hour to generate, Figure 2.5D and Figure 2.6D each took over two days to generate. The deterministic approach is not subject to these sorts of runtime issues, and though the stochastic implementation is exact – even for large number of molecules – this example shows that it is not practical to use in all situations, and deterministic methods will often be a better choice. The stochastic framework appears to be much more at home with small numbers of molecules. Not only does it appear to be on a firmer physical basis than the deterministic approach in this realm (Gillespie, 1976; Gillespie, 1977b; Gillespie, 1992), but the runtime is more likely to be reasonable.

## *References for Chapter 2*

Arkin, A., Ross, J., and McAdams, H. H. (1998). Stochastic kinetic analysis of developmental pathway bifurcation in phage lambda-infected Escherichia coli cells. *Genetics* **149,** 1633-48.

Artavanis-Tsakonas, S., Matsuno, K., and Fortini, M. E. (1995). Notch Signalling. *Science* **268,** 225-232.

Bartholomay, A. (1958). Stochastic models for chemical reactions. I. Theory of the unimolecular reaction process. *Bull. Math Biophys.* **20,** 175-190.

Chitnis, A. B. (1995). The Role of Notch in Lateral Inhibition and Cell Fate Specification. *Mol. Cell Neurosci.* **6,** 311-321.

Collier, J. R., Monk, N. A., Maini, P. K., and Lewis, J. H. (1996). Pattern formation by lateral inhibition with feedback: a mathematical model of delta-notch intercellular signalling. *J. Theor. Biol.* **183,** 429-46.

Davidson, E. H. (1986). "Gene activity in early development." Academic Press, Orlando.

Delbruck, M. (1940). Statistical fluctuations in autocatalytic reactions. *Journal of Chemical Physics* **8,** 120-124.

Edelstein-Keshet, L. (1998). "Mathematical Models in Biology." McGraw-Hill, Boston.

Elowitz, M. B., Levine, A., Siggia, E. D., and Swain, P. S. (2002). Stochastic Gene Expression in a Single Cell. *Science* **297,** 1183-1190.

Gibson, M. A., and Bruck, J. (2000). Efficient Exact Stochastic Simulation of Chemical Systems with Many Species and Many Channels. *J. Phys. Chem. A* **104,** 1876-1889.

Gillespie, D. T. (1976). A General Method for Numerically Simulating the Stochastic Time Evolution of Coupled Chemical Reactions. *Journal of Computational Physics* **22,** 403.

Gillespie, D. T. (1977a). Concerning the Validity of the Stochastic Approach o Chemical Kinetics. *Journal of Statistical Physics* **16,** 311-319.

Gillespie, D. T. (1977b). Concerning the Validity of the Stochastic Approach of Chemical Kinetics. *Journal of Statistical Physics* **16,** 311-319.

Gillespie, D. T. (1977c). Exact Stochastic Simulation of Coupled Chemical Reactions. *Journal of Physical Chemistry* **81,** 2340-2361.

Gillespie, D. T. (1992). A rigorous derivation of the chemical mater equation. *Physica A* **188,** 404-425.

Gillespie, D. T. (2001). Approximate accelerated stochastic simulation of chemically reacting systems. *Journal of Chemical Physics* **115,** 1716-1733.

Gracheva, M. E., Toral, R., and Gunton, J. D. (2001). Stochastic effects in intercellular calcium spiking in hepatocytes. *J. Theor. Biol.* **212,** 111-25.

Greenwald, I. (1998). LIN-12/Notch signaling: lessons from worms and flies. *Genes Dev* **12,** 1751-62.

Guptasarma, P. (1995). Does replication-induced transcription regulate synthesis of the myriad low copy number proteins of *E. coli*? *BioEssays* **17,** 987-997.

Haldane, J. B. S. (1930). "Enzymes." Longmans, Green amd Co., London.

Halling, P. J. (1989). Do the laws of chemistry apply to living cells? *TIBS* **14,** 317-318.

Kramers, H. A. (1940). Brownian motion in a field of force and the diffusion model of chemical reactions. *Physica* **7,** 284-304.

McAdams, H. H., and Arkin, A. (1997). Stochastic mechanisms in gene expression. *PNAS* **94,** 814-9.

McAdams, H. H., and Arkin, A. (1999). It's a noisy business! Genetic regulation at the nanomolar scale. *Trends Genet.* **15,** 65-9.

McIlwain, H. (1946). The Magnitude of Microbial Reactions Involving Vitamin-like Compounds. *Nature* **158,** 898-902.

McQuarrie, D. A. (1967). Stochastic Approach to Chemical Kinetics. *Journal of Applied Probability* **4,** 413-478.

Michaelis, L., and Menten, M. I. (1913). Die Kinetik der Invertinwirkung. *Biochem. Z.***,** 333-369.

Murray, J. D. (1993). "Mathematical Biology." Springer-Verlag, Berlin.

Purcell, E. M. (1977). Life at Low Reynolds Number. *American Journal of Physics* **45,** 3-11.

Renyi, A. (1954). Treatment of chemical reactions by means of the theory of stochastic processes (In Hungarian). *Magyar Tud. Akad. Alkalm. Mat. Int. Közl.* **2,** 93-101.

Singer, K. (1953). Application of the theory of stochastic processes to the study of irreproducible chemical reactions and nucleation processes. *Journal of the Royal Statistical Society B*. **15,** 92-106.

Teich, M. (1992). "A documentary history of biochemistry 1770-1940." Associated University Presses, Cranbury, NJ.

Waage, P., Gulberg, C.M. and Abrash, H.I. (trans). (1986). Studies Concerning Affinity. *Journal of Chemical Education* **63,** 1044-1047.

# Chapter 3: *Hox* Network

> It turns out to be remarkably difficult for mathematicians and computer scientists who are enthusiastic about biology to learn enough biology not to be dangerous, and vice versa. After all, many of us became biologists because we didn't like math. For biologists to learn the mathematics turns out to be challenging in quite a different way. And there is a huge amount of non-understanding—I would not go so far as to say misunderstanding—that results. But getting these disciplines together has turned out to be a much easier thing to say than to do…We have to do a much better job of teaching at the interfaces of the disciplines.
>
> - David Botstein, 2002

## *Introduction*

The problem under investigation is a study of the *Hox* regulatory mechanism in the developing hindbrain using a mathematical model based on a stochastic simulation algorithm (SSA) presented in Chapter 2. Much of this chapter is based on my paper published in the journal *Developmental Biology* (Kastner et al., 2002).

## *Developmental Biology Introduction*

In developmental biology, the establishment of asymmetry early in embryogenesis sets the stage for the formation of the body proper. The first axis formed is along the anterior-posterior (or rostral-caudal) axis of the embryo. Cells are endowed

with positional information that allows the proper formation of structures that correspond to their position along the axis. In other words, head structures form from the anterior part of the newly formed axis, and tail structures form from the posterior part of the axis.

The beginnings of the central nervous system in vertebrates occur early in development with the formation of the neural plate. The neural plate then folds into the neural tube. There are variations in how this occurs in different species, but in general the process is fairly similar: the tube begins as a groove down the midline of an embryo, and eventually closes from the joining of the flaps on either side (Gallera, 1971). This is a crucial process in development, and if the neural tube fails to close properly it can lead to defects like Spina bifida or Anencephaly (Van Allen et al., 1993).

Although initially straight, the upper section of the neural tube nearest the head forms a variety of bulges and constrictions that compartmentalize brain and spinal cord into distinct sections. The anterior most bulges will give rise to cells that make the prosencephalon (forebrain) and structures such as the olfactory lobes, the cerebrum, and the retina. Just posterior to that, the mesencephalon (midbrain) will give rise to structures like the optic lobes and the tectum. The most posterior bulges are the developing rhombencephalon (hindbrain) which gives rise to the cerebellum and the brain stem (Gilbert, 1997). Shortly after the closure of the neural tube, the vertebrate hindbrain further develops a series of axial bulges called rhombomeres that effectively compartmentalize the rhombencephalon into 8 smaller segments. The rhombomeres have been shown to be cell lineage restricted in that cells from one rhombomere do not cross over into another (Fraser et al., 1990). The segmentation of the hindbrain into rhombomeres is a crucial process in the proper specification of the developing structures

of the hindbrain (Guthrie and Lumsden, 1991). In a series of closely aged chick embryos, Figure 3.1 shows the closing of the neural tube and the rhombomeres.

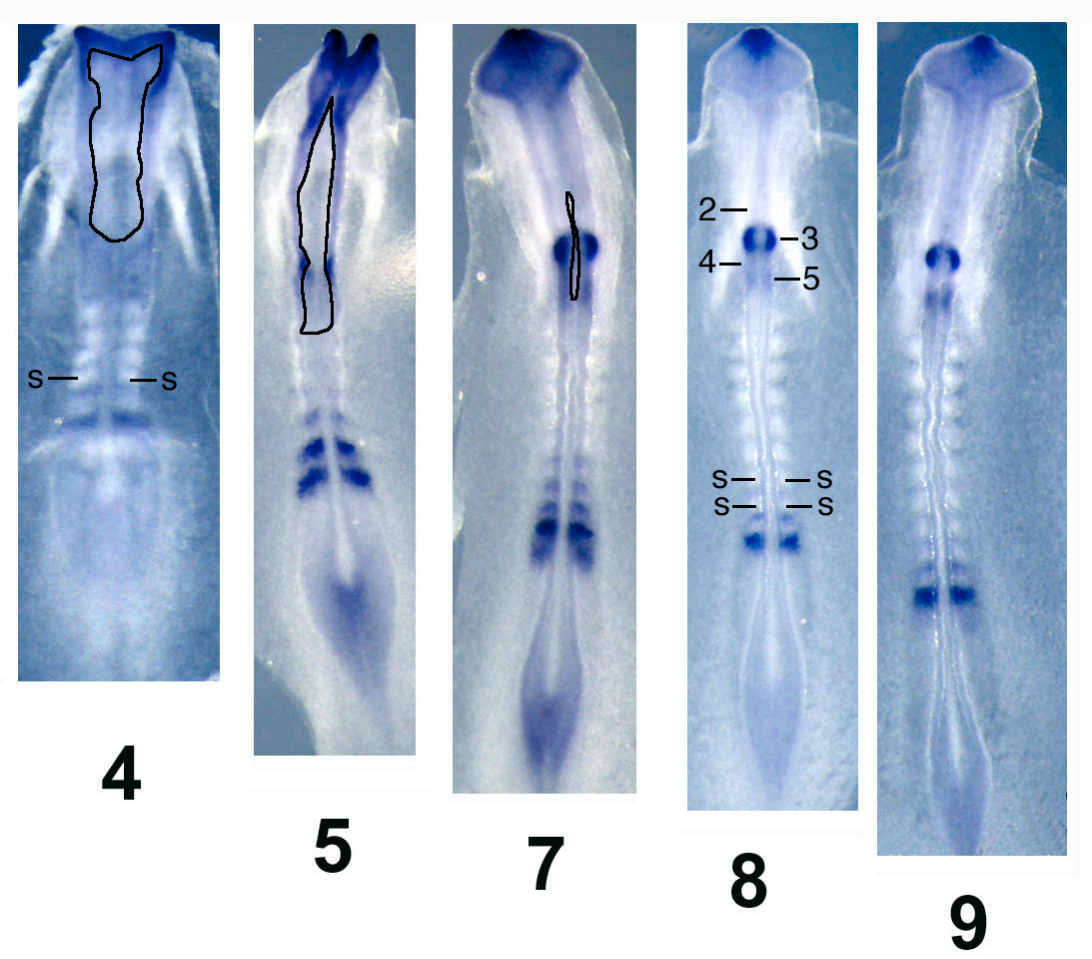


**Figure 3.1 Neural tube closure and rhombomere emergence**. These five embryos are stained for the segmentally expressed gene *EphA4* (previously called *Sek2*, the probe is courtesy of C. Tabin). The embryos are oriented with the head at the top of the page and the tail at the bottom. The somites (examples marked by **S** in **4** and **8** above) are block-like collections of cells that form in pairs along the rostral-caudal axis of the embryo. They appear in a regular fashion, a new pair appearing every 90 minutes or so. Because of this, the somites are commonly used for a staging

mechanism and the numbers below the embryos are the pairs of somites in each embryo. The outlined areas in **4**, **5** and **7** show the gap between the neural folds before the neural tube is fully closed in the mid and hindbrain. Notice that in **4** the tube is wide open, in **7** the tube is almost completely closed, and in **8** and **9** the tube is closed. In **8** rhombomeres 2 through 5 are marked, with rhombomere 3 being the most prominent due to its strong expression of *EphA4*. Rhombomere 3 is also clearly visible in **7**. A slightly different version of this figure will be appearing in the 7th edition of the book Developmental Biology by S. Gilbert.

The rhombomeres are transitory structures that appear for about 15% of the development time of the embryo. In the chick, they appear after about 25 hours of development, and disappear by the100 hour mark. In a cartoon adapted from Lumsden (1990), Figure 3.2 shows the order and approximate timing of the formation of rhombomere boundaries. The *Hox* gene network under investigation is expressed in rhombomeres 4 and 5.

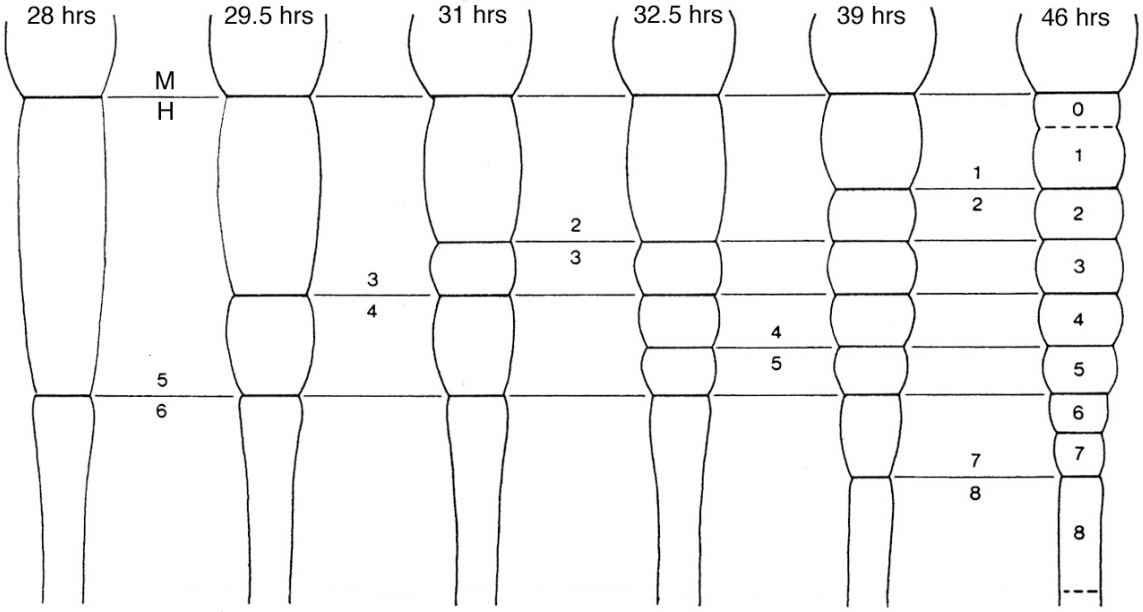


**Figure 3.2 Rhombomere emergence**. The first boundaries noticeable are the boundary between the midbrain and hindbrain (M/H), and the boundary between rhombomeres 5 and 6 (r5 and r6), both visible by 28 hours of development. The first fully formed rhombomere is r3 at 31 hours of development, followed by r4 and r5 at 32.5 hours, r2 at 39 hours, then r6, r7, r8 and r1 by 46 hours. The existence of rhombomere 0 is under debate, and there is no discernable boundary between rhombomere 8 and the developing spinal cord. The initial formation of the 5/6 boundary is actually very dependent on incubation conditions, and the initial start time may vary significantly.

## Introduction to the Control and Expression of Genes

This section contains a short introduction to the molecular biology behind the control and expression of genes. It is not intended to be all encompassing, and for more details, the reader is directed to Alberts et al. (1994). However, it is intended to give the reader enough information to follow the construction of the model presented below.

The problem of tissue differentiation mentioned above also needs to be addressed at a different level: that of the cell. The different cells in a multicellular organism contain the same DNA yet they differentiate from each other by creating and accumulating different messenger RNA (mRNA) and different proteins. The process by which a cell creates protein can be broken down into two major pieces: transcription and translation.

Transcription is the process by which mRNA is created from the DNA, while translation is the process by which the mRNA is turned into protein. Collectively, this process is called the Central Dogma. Obviously this is a simplified view as many other steps can occur. These include RNA splicing in which parts of the RNA are excised from the original strand. But while these steps are important in understanding the biology of the problem, they are not crucial to include from a modeling standpoint. This is because each of these steps is part of a cascade that affects the timing of the end result, but not what the end result is.

Transcriptional activators are the major building blocks of the model and it is this process that garners the most attention. Transcriptional factors are proteins that recognize a defined DNA sequence in the regulatory control region of a particular gene. Factors can be activators, which means that they contribute to the making of mRNA, or

repressors that prevent the mRNA for that gene being made. When even one molecule of a transcription factor is available for binding to the regulatory region of a gene, the probability that transcriptional will occur is significantly increased. Transcriptional control is a very complicated process and it can take multiple transcription factors acting in tandem to switch the gene on and allow the transcription of mRNA. This work focuses on the *cis*-regulation of genes: regulation that is controlled by sequences close to the start site for transcription. *Cis*-regulatory factors are generally the most important elements in transcription initiation.

## *Hox* Genes

Discovering regulatory genes, genes that control the major aspects of a biological system, has been the focus of biological research ever since molecular tools have become available. While no single master regulator gene has appeared, there have been some remarkable discoveries in developmental biology in the past few decades. In particular the homeotic genes have been identified as a family of genes that control genetic aspects of development (Duboule, 1994). First identified in the fruit fly *Drosophila melanogaster,* an evolutionary study showed that the homeobox—a set of 60 amino acids found in several different genes in *Drosophila* and encoding a DNA binding domain—also appeared in beetles, earthworms, chicken, mouse, and human (McGinnis et al., 1984). Mutation studies have been carried out in *Drosophila*, and they show that if a homeobox gene is mutated, the axial organization of the body is altered, leading researches to conclude that the homeobox genes are critical in the proper formation of the body plan (McGinnis and Krumlauf, 1992). In addition, it now appears that the

homeobox genes might indeed be the master regulatory genes of the body axis. It has recently been shown that natural alterations in the homeobox protein Ubx are likely to be the critical event that led to the evolution of hexapod insects from multilegged crustacean ancestors (Ronshaugen, 2002).

The 39 *Hox* (homeobox containing) genes found in higher vertebrates—like human and mouse—are organized into four chromosomal clusters located on different chromosomes. A *Hox* related family is found in invertebrates as well, but in this instance the genes can be found in a single cluster on one chromosome. Using information about their amino acid makeup, the genes can be aligned to one another using the *Drosophila* genes as a reference. They are easily grouped into13 paralog groups, or subfamilies. The *Hox* genes are collinear: the order they appear on the chromosome is the same as the order in which they appear in the body axis. Not only that, they have a temporal expression that is related to the order on the chromosome as well; the lower numbered families appear earlier in development than the higher number families. Finally, they also have a response to retinoic acid (RA), both in sensitivity and in the efficiency of the binding, that can be correlated to their order on the chromosome; the lower number families are very sensitive to RA and bind it tightly (when there is a retinoic acid response element in the control region of the gene), and the higher numbered families are less sensitive to RA and bind it more weakly. This information is summarized graphically in Figure 3.3 below.

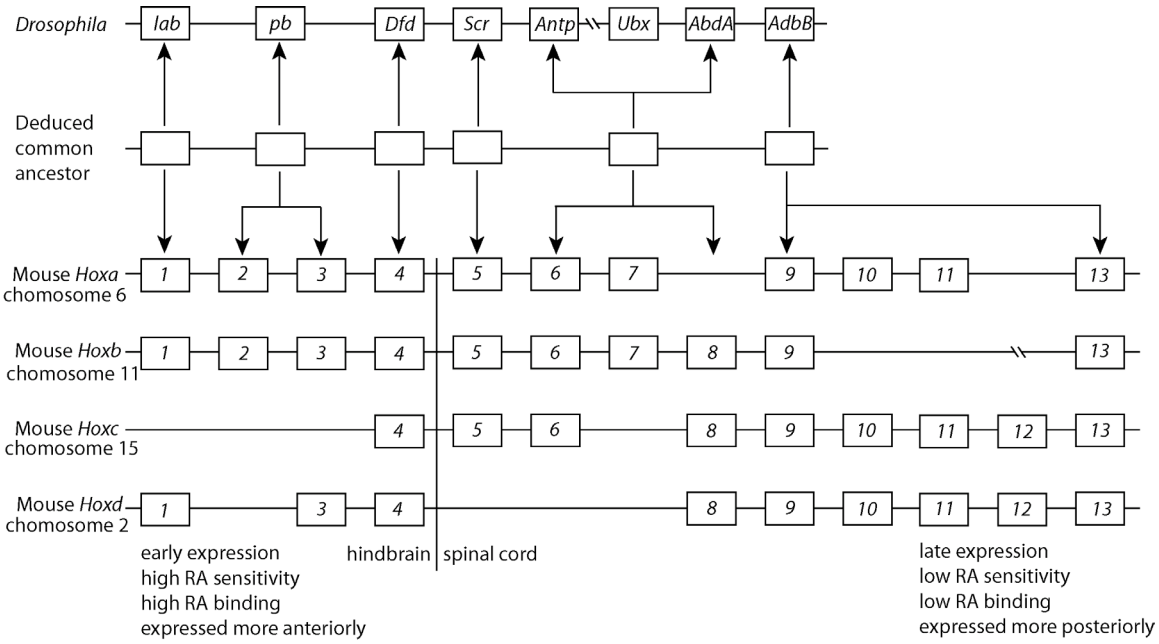


**Figure 3.3 Hox Paralog families** Alignment of the *Drosophila* HOM-C complex, the four mouse *Hox* chromosomal clusters, and their deduced common ancestor. After (Lufkin, 1997), with additional information from (Neuteboom and Murre, 1997; Pellerin et al., 1994).

The *Hox* gene family is a set of transcription factors that has been shown to be crucial in helping to confer rhombomere identity (Wilkinson, 1993). This can be shown dramatically by altering the expression of just a single gene: it was shown that misexpression of *Hoxb1* was able to transform rhombomere identity (Bell et al., 1999). The *Hox* genes exhibit rhombomere-restricted patterns of expression and the expression of several major rhombomere restricted genes (including the *Hox* genes) is shown below in Figure 3.4A.

But Figure 3.4A is very idealized. While the *Hox* genes certainly display rhombomere restricted patterns of expression, the expression does not stop cleanly at the boundaries. This is best shown in Figures 3.3B, a 10x magnification picture of rhombomeres 3 through 7 (r3-r7) of a chick embryo stained for *Hoxb1*.

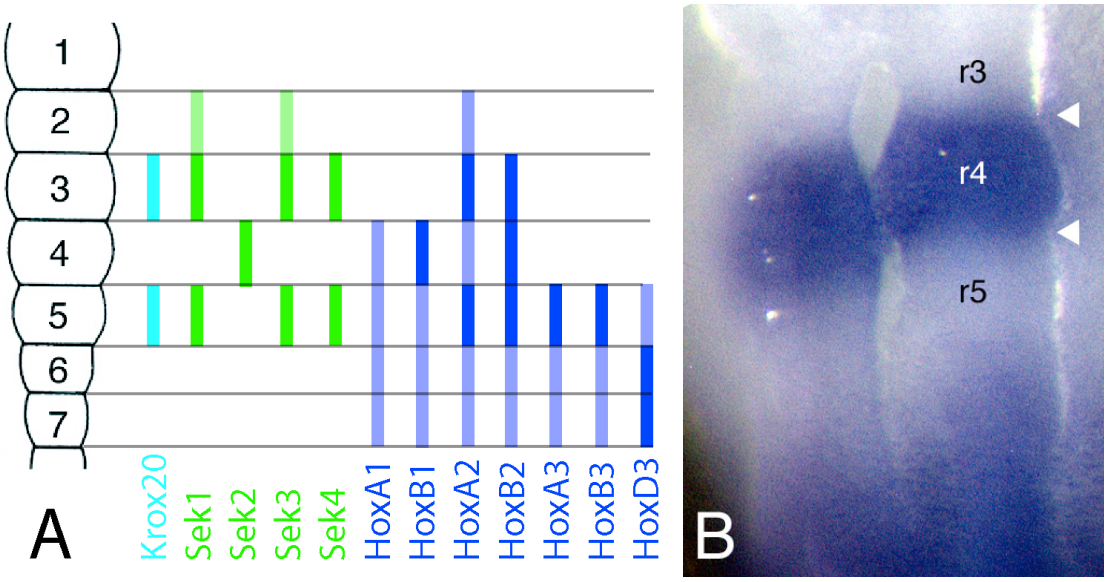


**Figure 3.4 Rhombomere restricted expression of several genes (A)** Expression patterns for several genes with rhombomere restricted boundaries. The lighter colors signify transient expression, and the darker colors correspond to continued levels of expression. After (Lumsden and Krumlauf, 1996). **(B)** A10x picture of r3 (top) through r7 (bottom) of a chick hindbrain that has been stained for the gene *Hoxb1* (probe courtesy of R. Krumlauf). The rostral and caudal boundaries of r4, as exemplified by the bulge in the tissue, have been marked with arrows. Notice that the gene expression is essentially restricted to r4, but the boundary is not a sharp one and there is some expression of the gene in the adjacent rhombomeres, most notably r3.

### Retinoic Acid

It has been long known that elevated levels of the retinoid vitamin A disturbs axial formation in vertebrates (Kalter and Warkany, 1959) and recently it has been shown that sufficient levels are necessary for proper development (Niederreither et al., 1999). Retinoic acid (RA) is the biological active derivative of vitamin A, and it acts through two classes of receptors, the RA receptors (RAR) $\alpha$, $\beta$, and $\gamma$ and the retinoid X receptors (RXR) $\alpha$, $\beta$, and $\gamma$. RA also plays an important part in the this process as it is able to directly regulate the expression of *Hox* family members, and alterations in the RA response elements in the *cis*-regulatory domain of reporter genes significantly change the expression patterns (Gavalas and Krumlauf, 2000).

## *Modeling*

### Network Creation

Stochastic investigations in biology models have so far been focused on intracellular systems. The goal of this thesis was to explore the utility of a SSA approach to modeling a gene network involving many cells. The direct coupling of the SSA implementation of a network and individual molecular events would seem to lend itself to both the analysis and logical organization of the ever growing data on the control of *Hox* genes in the developing hindbrain. The analysis presented here shows that the approach captures the timing, patterning, and variation in *Hox* gene expression without the need for artificially injected noise. The tests against some of the available experimental

perturbations suggest that the SSA will have predictive value and allow researchers in the laboratory to identify and focus attention on the most fruitful experiments.

Several of these predictions are noted, and two experiments were designed to clarify and test aspects of the model. One of the experiments (found in Chapter 4) suggested that a design decision made during the creation of the model was incorrect. The novel biological data resulted in a refinement of the model, thus closing the loop between modeling and experiments.

The SSA investigation into the *Hox* network focused on an investigation of the interaction of *Hoxa1*, *Hoxb1*, *Hoxb2*, *Krox20* and RA in rhombomeres 4 and 5 (r4 and r5). *Krox20* is not a homeobox gene, but it regulates *Hox* genes and is important for proper segmentation (Schneider-Maunoury et al., 1993). As mentioned previously, this system was chosen for a variety of reasons including the amount of information that is known: the molecular studies of the hindbrain have offered sufficient details to assemble a model for the interactions important in regional control of gene expression. In addition, the accessibility of the chick hindbrain early in development made this an attractive system in which hypothesis could be tested.

The following discussion will be enhanced by a brief comment on nomenclature. Names in italics (*Hoxa1)* refer to the genes or the mRNA for the gene, while names in normal font (Hoxa1) refer to the protein product of the mRNA. *Hoxa1* is the first of the *Hox* genes to be expressed in the hindbrain (Murphy and Hill, 1991) and its expression appears to be directly regulated by a retinoic acid response element (RARE) (Frasch et al., 1995; Langston and Gudas, 1992). *Hoxb1* expression also appears to depend on

RAREs, an element on the 3' end of the gene (the end of the DNA without a phosphate) the which helps establish early expression (Marshall et al., 1994), and a repressor element on the 5' end of the gene (the end of the DNA with a phosphate) which acts in r3 and r5 (Studer et al., 1994) and which appears to start altering gene expression around 8.0 days post coitus (dpc) in the mouse (R. Krumlauf, personal communication). The early expression of *Hoxb1* is also dependent on Hoxa1 (Studer et al., 1998) with the cofactor pbx (Green et al., 1998; Phelan et al., 1995), but continued expression in r4 is controlled by a strong auto regulatory loop with the cofactors exd/pbx (Popperl et al., 1995) and prep1 (Berthelsen et al., 1998a). *Hoxa1* is expressed to a rostral limit in the developing neural tube to the presumptive r3/r4 boundary at 7.75-8.0 dpc, but the expression then regresses, vanishing from the hindbrain by 8.5 dpc. The expression of *Hoxb1* is very similar, except for the continued autoregulatory maintenance in r4 (Maconochie et al., 1996). Hoxb1, pbx, and prep1 all have a hand in up-regulating *Hoxb2* in r4 (Ferretti et al., 2000; Maconochie et al., 1997), while the later r5 expression of *Hoxb2* is regulated by Krox20 (Nonchev et al., 1996a; Nonchev et al., 1996b; Sham et al., 1993). In r5 *Krox20* appears to be repressed by Hoxa1 and Hoxb1, and expression of *Krox20* occurs in r5 after they retreat from the hindbrain around 8 dpc. By 8.5 dpc expression of *Krox20* and *Hoxb2* can be detected in r5 (Barrow et al., 2000; Wilkinson et al., 1989). Thus, the mouse *cis*-regulatory network can be drawn as in Figure 3.5 below.

The synthesis of this data into Figure 3.5 is a new result and has been received favorably by one of the leaders in the field (R. Krumlauf, personal communication). The organization of the figure itself draws upon ideas presented in the literature, but several features of the diagram are novel and go beyond current representations. For instance,

the activation and repression binding sites are correctly drawn in their relative positions on the chromosome, with the exception of *Krox20* (as it is still unclear how the Hoxa1 and Hoxb1 repression mechanism works and where the components are). The horizontal orientation of *Hoxb1* and *Hoxb2* highlights the fact that they appear on the same chromosome, while the vertical orientation of *Hoxa1* and *Hoxb1* highlights the fact that they are paralogs. *Krox20* is offset both vertically and horizontally, from all the other genes, thus showing that it is not connected. This presentation brings a new depth to the standard representations (*cf.* Davidson, 2001).

The figure also shows the complexity of the situation. Even though this system was chosen because there was a readily identifiable network that had a minimum number of inputs, the network is still very complicated and includes a nonlinear feedback term for the autoregulation of *Hoxb1*.

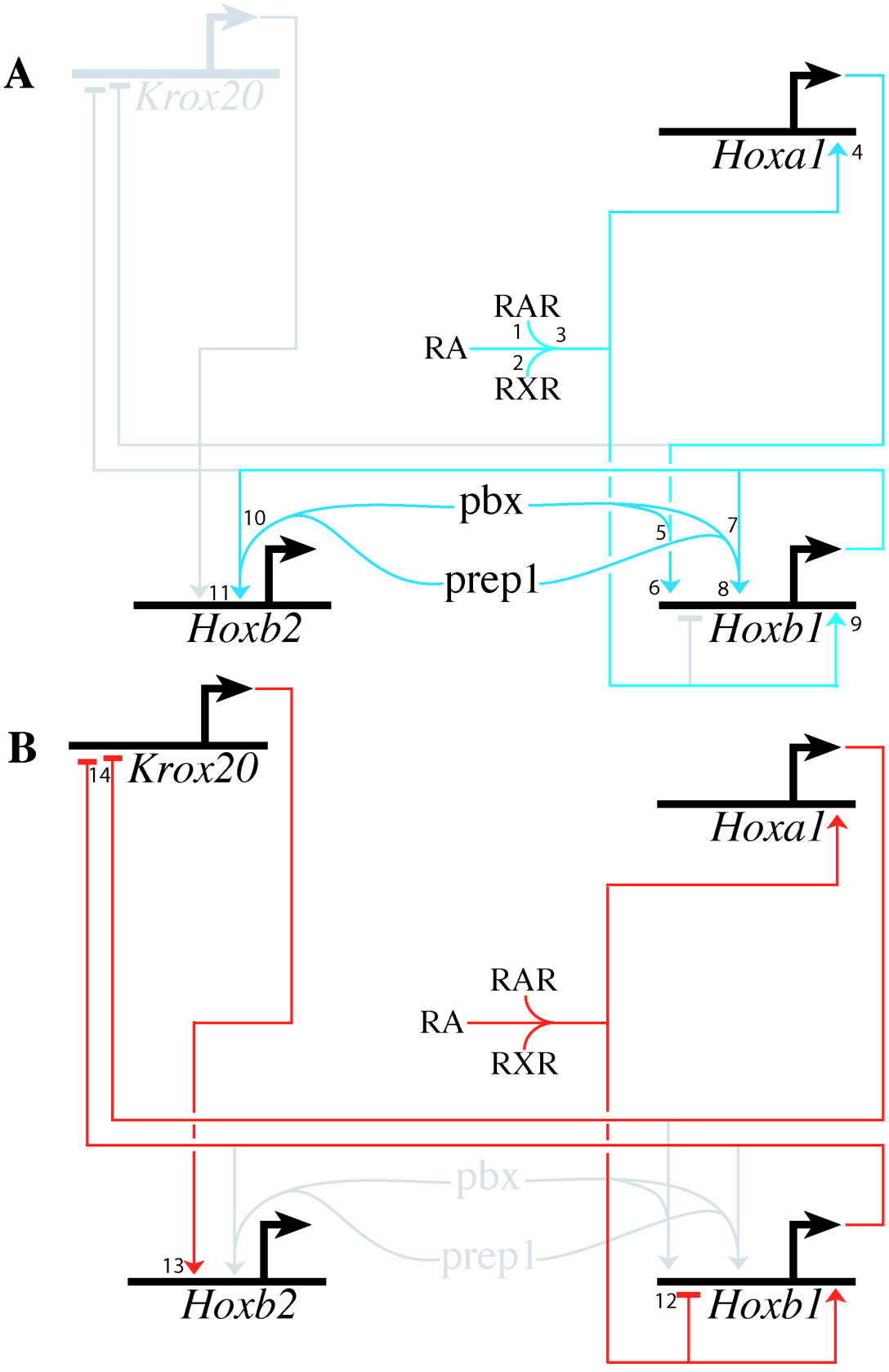
A
Krox20
Hoxa1
4
RAR
1
3
RA
2
RXR
pbx
10
5
7
prep1
11
6
8
Hoxb2
Hoxb1
9
B
14
Krox20
Hoxa1
RAR
RA
RXR
pbx
prep1
13
12
Hoxb2
Hoxb1

**Figure 3.5 *Hox cis*-regulatory network** in r4 **(A)** and r5 **(B)** The network is drawn in a way to emphasize that (1) each cell contains the entire biochemical network, and (2) certain interactions dominate in a particular rhombomere. Inactive elements are denoted in gray. The numbers near each intersection refer to the references for the interaction. **(A)** Starting with retinoic acid (RA) in the middle of the diagram, the RA binds with RAR (1: Petkovich et al., 1987) and RXR (2: Leid et al., 1992a), which can then form a dimer (3: Leid et al., 1992b). The dimer can bind as a transcriptional activator to *Hoxa1* (4: Frasch et al., 1995; Langston and Gudas, 1992) or *Hoxb1* in r4 (9: Marshall et al., 1994). The Hoxa1 protein, after binding with the pbx/prep1 complex (5: Berthelsen et al., 1998b), can then bind as a transcriptional activator to *Hoxb1* (6: Studer et al., 1998). The Hoxb1 protein, in conjunction with pbx/prep1 can bind to *Hoxb1,* which provides an auto-regulatory mechanism (7,8: Popperl et al., 1995). The Hoxb1/pbx/prep1 complex can also bind as a transcriptional activator to *Hoxb2* (10,11: Maconochie et al., 1997). **(B)** The RAR/RXR dimer can bind as a transcriptional activator to *Hoxa1* (4: Frasch et al., 1995; Langston and Gudas, 1992) or *Hoxb1* (9: Marshall et al., 1994) in r5, and it can also bind as a transcriptional repressor to *Hoxb1* (12: Studer et al., 1994). Hoxa1 and Hoxb1 are hypothesized to be transcriptional repressors of *Krox20* (14: Barrow et al., 2000), while Krox20 is a transcriptional activator of *Hoxb2* (13: Sham et al., 1993).

While most of the *cis*-regulatory studies have been carried out in mice, chick has proven to be a useful system for investigation of RA distribution. RA has long been thought to be a diffusible morphogen that is able to pattern the hindbrain (Gavalas and Krumlauf, 2000; Maden, 1999) and recent studies of RALDH-2 and CYP26, enzymes important in RA synthesis and degradation, reveal expression patterns that continue to support this view (Berggren et al., 1999; Swindell et al., 1999). In addition, a RALDH-2 knockout shows effects similar to vitamin A deficiency (Niederreither et al., 1999). More direct tests of sensing this gradient in mouse or chick have been challenging; there has been no conclusive evidence (Gavalas and Krumlauf, 2000). Despite this lack of direct evidence for a gradient, circumstantial evidence for it continues to accumulate. Most recently a study of RAR blocking by an antagonist has suggested that the establishment of hindbrain boundaries is dependent on RA concentration (Dupe and Lumsden, 2001). The work also suggested that the cells in the mid- and hindbrain are still responsive to RA through stage10. Therefore, RA cannot still be present in the midbrain and anterior part of the hindbrain, otherwise genes that respond to RA—including *Hoxa1* and *Hoxb1*—would be expressed in this region. Thus, even if there is not an actual RA gradient, there may be a graded response to retinoids, possibly involving other factors in the system that help modulate the ability of the cell to respond to RA. Taken together, the evidence is suggestive that a differential of some sort, perhaps through RA concentration, or through the temporally modulated ability to respond to RA, helps establish the *Hox* gene patterns.

Because the SSA model is built on, and driven by, the underlying biochemistry of the system, the reactions can be translated directly into the discrete events of the

simulations. In this investigation, some of the steps of the system were deliberately omitted. For example, instead of creating explicit reactions for the transcription of nuclear RNA, the splicing into mRNA, and the exporting of the mRNA to the cytoplasm, the simulation instead creates mRNA as a primary transcript. This is not unreasonable as long as the rate parameters $c_\mu$ are adjusted to reflect the subsequent delay, and as more data that describes these reactions is collected, these pieces can be easily incorporated at a later date.

Using Figure 3.5 as the network of interest, an SSA that described the *Hox* network system has been created using the C programming language. The source code for the model can be found in Appendix C and on the accompanying CD-ROM. The model contains 59 chemical events that can occur in each cell. They can be classified into 5 main categories: binding (including activation, repression, dimerization, and *Hox*/pbx/prep1 complex formation), unbinding, transcription, translation, and decay (of mRNA, dimers, complexes, proteins, and receptors). The two remaining events that do not fall into these categories are diffusion and division.

Of the 59 chemical events, most of them are first-order reactions. First-order reactions are ones with a single reactant, and so the rate of the reaction is proportional to the number of molecules. Therefore, the probabilistic rate for the stochastic simulation is of the form $a_\mu = c_\mu s_1$, where $s_1$ can be the number of mRNA available to be turned into proteins, or the number of molecules (including RA, mRNA, proteins, complexes, and receptors) available for decay. This is, of course, a simplified view of the true state of affairs in the cell. For instance, the mRNA cannot be translated into protein without the

presence of a ribosome and the necessary amino acids, but these are assumed to be available in excess.

Zeroth-order reactions are ones that reactions that occur "spontaneously" and are not linked to any of the expressed genes in the simulation. Instead, they are considered as a stochastic event that can occur with some constant (low) probability and are governed by equations of the form $a_\mu = c_\mu$. One example of a zeroth-order reaction is the cell division function. The typical simulation encompasses 18 hours of developmental time and so the model includes a rudimentary mechanism for cell division and this is why the presumptive boundary sometimes shifts in the movies. When the division occurs, the resources in the cell are divided subject to a normal distribution between the daughter cells. The other zeroth-order reactions describe the creation of the RAR and RXR receptors and the pbx protein complex.

Second-order reactions involve two species of the simulation that combine and are of the form $a_\mu = c_\mu fg$, where $f$ is the number of molecules of the first species, and $g$ is the number of molecules of the second species. The four second-order reactions in the simulation describe RA binding to RAR, the binding of RA to RXR, the dimerization of the bound RAR and RXR forms, and the formation of the Hox/pbx/prep complexes. Because the species in these second-order reactions are different, there is no need to introduce a combinatorial factor as in Table 2.1.

There are a variety of ways to implement activation functions. These include binary activation, sequential activation, proportional activation, and Hill functions. A binary activation would be when a single transcription factor binds to the gene, thus

creating an "activated" form of the gene. This activated form is then primed for the transcription of mRNA. Because of the large binding coefficients that accompany transcription factors and DNA, even a small number of molecules of a transcription factor are enough to enable transcription. However, they must be present in sufficient numbers to establish a steady state in the binding/dissociation reactions.

Yet another way of implementing a transcription function is to assume that the probability of transcription is proportional to the number of transcription factor molecules. In other words, $a_\mu = c_\mu fg$ but in this case $g$ is either 1 if a gene is available for transcription or 0 if the gene is not available for transcription, and $f$ is the number of transcription factor molecules present. This form doesn't assume an explicit notion of an activated gene.

In the first incarnation of the model, the activation and repression functions are implemented using a Hill function (Hill, 1910), a typical way to represent cooperative binding. This takes the general form $a_\mu = c_\mu \frac{f^h}{\kappa_\mu + f^h} f \cdot g$, where $f$ is the number of molecules of a particular transcription factor, $\kappa_\mu$ is a threshold factor, and $g$ is the number of molecules of a gene available. Similar to the proportional case, if a gene is currently unbound, the value of $g$ is 1, while if it is bound by a transcriptional factor the value of $g$ is 0. The exponent $h$ is called the Hill coefficient and it affects the steepness of the response. The Hill function is an empirically derived expression, used in differential equation models, that yields the observed kinetics in these situations. Thus, in the stochastic reaction approach the complete Hill function expression is treated as simply another rate coefficient for the purposes of converting it to the appropriate probability of

occurrence of the corresponding reaction. Others have used a similar method in their stochastic description of gene transcription (Arkin et al., 1998).

When it comes to the activation of *Hoxb1* in r4, there are actually two transcription factors that can bind to the gene. This is implemented using a variety of gene states controlled by a combination of Hill functions and sequential activations. *Hoxb1* is initially up-regulated by the RA dimers and the cross activation by *Hoxa1*. Therefore if one of those two factors is bound, the gene is marked as in an activated state, but if both are bound, the gene is marked as "superactivated." Each of those two activated states carries its own probability of transcription, with the superactivated form much higher. Maintenance is controlled by the *Hoxb1* auto-regulatory loop, and once the *Hoxb1* protein is present in sufficient numbers, auto activation can occur, again with an associated probability of transcription.

Diffusion is yet another first order reaction, and more molecules of RA means that there is greater chance of a diffusion event occurring. But the diffusion is secondary to the actual creation of the RA, and that needs to be treated with some care.

## Retinoic Acid Source

In the course of considering different ways that RA might pattern the hindbrain, a paper appeared that provided additional insight (Dupe and Lumsden, 2001). This work suggested that cells in the hindbrain are less able to respond to RA over time. This is not inconsistent with the previously mentioned investigations that suggest a physical variation in RA patterns the hindbrain (Gavalas and Krumlauf, 2000; Maden, 1999), but it does make modeling the system more challenging. Taken together, these studies

propose that a variation of some sort (either temporal or spatial or possibly both) is an important component in patterning the hindbrain, and provided support of some of the hypotheses used to construct the model.

There are two main ways that this variation can be implemented. The first is to create cells that are less responsive to RA over time, and the second is to create a variation in the RA. The model was built to allow for both of these possibilities. There is more evidence for a physical variation however, and the modeling efforts reflect this fact.

There are a variety of possible functions that can be used for modeling a physical variation of RA and many forms were considered. In Equations 3.1 are a set of differential equations derived from the Law of Mass Action that captures part of the network. While this formulation is problematic in general, especially for situations such as these with the low levels of the transcription factors, it was useful in quantifying the effects on the *Hoxa1*, *Hoxb1* and *Hoxb2* due to different RA source terms. Briefly, the rate of change of *Hoxa1* $(A_1)$ is dependent upon the creation effects of RA, and the depletion effects $(-\phi A_1)$ caused by normal decay or use as an up-regulator for *Hoxb1* $(B_1)$. Positive effects for *Hoxb1* include RA, the up-regulation by *Hoxa1* $(\alpha A_1)$ and the Hill auto-regulatory loop, while the depletion effects $(-\beta B_1)$ are caused by normal decay or its use as an up-regulation for *Hoxb2* $(B_2)$. The rate of change of *Hoxb2* is up-regulated by the amount of *Hoxb1* $(\delta B_1)$, and depleted by decay processes $(-\varepsilon B_2)$.

$$\frac{dA_1(t)}{dt} = RA(t) - \phi A_1(t)$$
$$\frac{dB_1(t)}{dt} = RA(t) + \alpha A_1(t) - \beta B_1(t) + \gamma \frac{B_1^{\,2}(t)}{1 + B_1^{\,2}(t)} \qquad (3.50)$$
$$\frac{dB_2(t)}{dt} = \delta B_1(t) - \varepsilon B_2(t)$$

**Equations 3.1 A simplified set of equations describing the behavior of the rhombomere 4 gene network.** Note that in this description there is only one cell, and this cell contains only 4 products and 6 reactions. This is a dramatic simplification from the full simulation of the 40 cells, each containing 30 products and 59 chemical reactions. But because the full simulation contains these basic reactions as well, this reduced set provided insight into the possible effects of different RA source terms.

A variety of different functions were considered for the RA source, and Figure 3.6 shows the trajectories of the solutions. The x-axis is time, and the y-axis is concentration. It is important to keep in mind that the experimental results in rhombomere 4 show that the *Hoxa1* mRNA increases then decreases, while the *Hoxb1* and *Hoxb2* mRNA reach a steady state. Therefore, the solutions that exhibit this behavior are the most interesting.

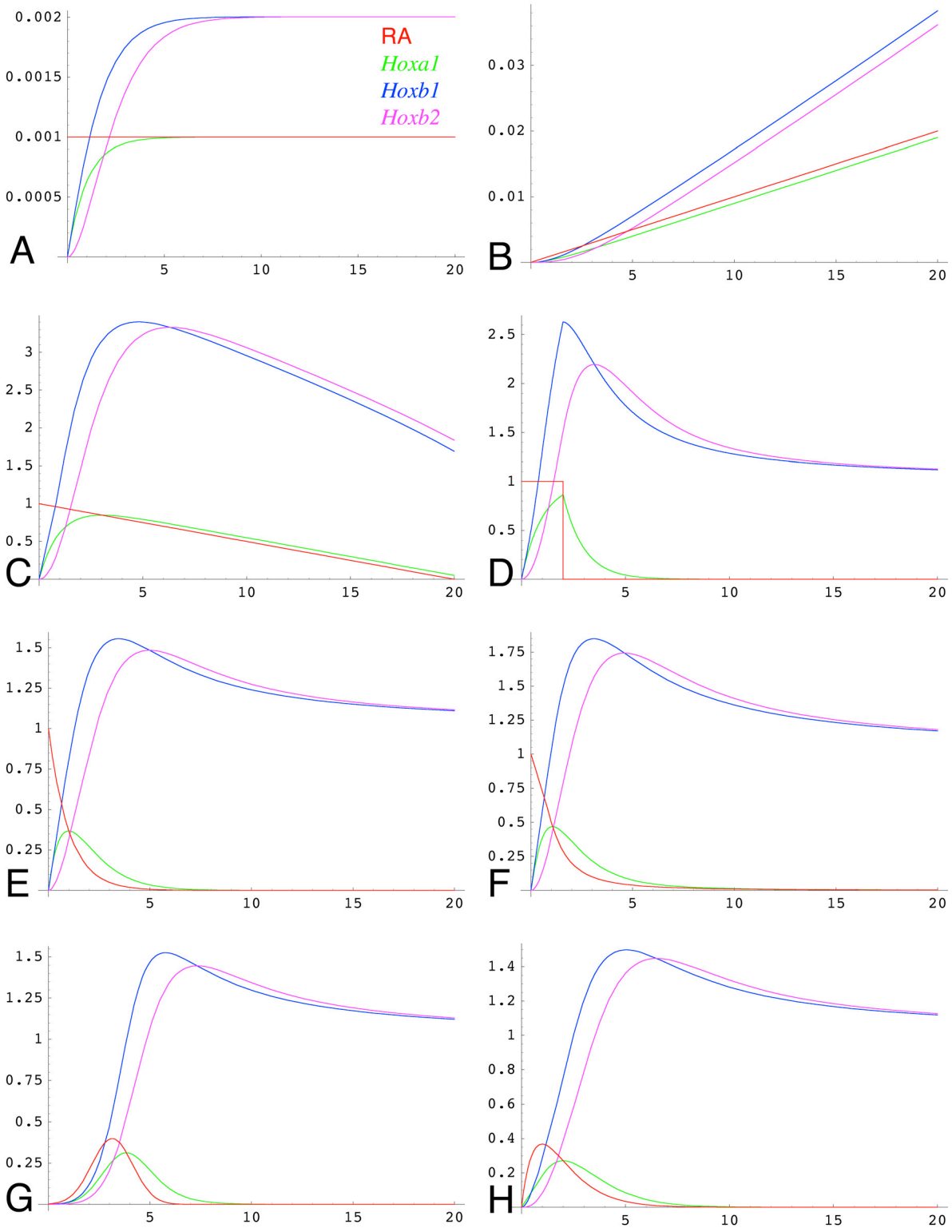
RA
Hoxa1
Hoxb1
Hoxb2
A
B
C
D
E
F
G
H

**Figure 3.6 A-H Response curves for various RA functions**. A variety of functions were investigated for the RA source term using the simplified network described in Equation 3.1. The legends for the plots **(B-H)** are the same as in **(A)**: RA in red, *Hoxa1* in green, *Hoxb1* in blue, and *Hoxb2* in magenta. The response curves were qualitatively the same for a wide range of the parameters. The parameters used to generate these particular plots were $\varphi = \alpha = \beta = \delta = 1, \gamma = 2, \varepsilon = 1/2$. **(A)** The source term $RA(t) = .001$ causes the cell to create a constant amount of RA over time. This causes the *Hoxa1* to increase to the same level as the RA source and is therefore not an appropriate model for the RA source. **(B)** A linearly increasing RA source term ( $RA(t) = .001t$ ) results in all the *Hox* genes to increase linearly over time, while **(C)** a linearly decreasing source term ( $RA(t) = 1 - .05t$ ) results in the *Hox* genes to decrease over time after an initial surge in *Hoxb1* and *Hoxb2* because of the auto-regulatory loop. Both of these are expected, and neither is appropriate. **(D)** The investigation took an interesting turn when the RA was modeled with the step function $RA(t) = \text{UnitStep}[2 - t]$. This resulted in the right type of qualitative behavior, namely, a surge or *Hoxa1* and steady state levels of *Hoxb1* and *Hoxb2*. Two of the problems with this include the square non-biological source term and the sharp response from the *Hoxa1*. But two other functions **(E)** $RA(t) = e^{-t}$, a decaying exponential, and **(F)** a quadratic decay $RA(t) = \frac{1}{1+t^2}$, produced very nice qualitative results. The *Hoxa1* increased then decreased, and the *Hoxb1* and *Hoxb2* reached a steady state due to the *Hoxb1* auto-regulatory loop. In addition,

both of these have a RA source that diminishes smoothly over time. The only problem with using a source term from one of these families is that they both start at $t = 0$ with a large amount of RA immediately. This is not possible biologically, but the following two functions do exhibit behavior that can occur biologically as they both exhibit a smooth ramp-up as well as a smoothly diminishing tail. **(G)** A Gaussian curve of the general form $RA(t) = e^{-(t-\pi)^2/2}$ or a Rayleigh function like **(H)** $RA(t) = te^{-t}$ meet all the desired criteria. Ultimately, the Rayleigh function was chosen because of the connection to other biological sources like insulin, which has a biphasic response with a strong initial response and a longer continuing source (Rorsman et al., 2000).

A Rayleigh function was ultimately chosen to model the diffusion source term for RA from the posterior of the embryo. This is implemented by having the first cell create the RA according to the probabilistic rate $a_0 = c_0 \cdot RA_0 \tau e^{-\alpha\tau^2}$ where $RA_o$ is the initial amount of RA in the system, and $\alpha$ controls the decay time of the source.

## Parameters

Using appropriate values for the model parameters is an important component in modeling the system behavior. Fortunately, several key parameters are known, but many of the important parameters for the model have not been assayed directly in experiments on the developing hindbrain. Estimates of many of their values can be made from data obtained in other systems, and were used in selecting parameters here (Table 3.1).

| Event | $K_d$ | Reference |
|---|---|---|
| RA binding to RAR | 0.5 nM | (Allegretto et al., 1993) |
| RA binding to RXR | 2 nM | (Allegretto et al., 1993) |
| RAR/RXR dimerization | 17 nM | (Depoix et al., 2001) |
| Dimer binding to *Hoxa1* | 3.8 nM | (Mader et al., 1993) |
| Dimer binding to *Hoxb1* | 5.3 nM | (Mader et al., 1993) |
| Hox/pbx/prep binding to DNA | 2 nM | (Pellerin et al., 1994) |

**Table 3.1** Various measured binding coefficients for the interactions of the components of the model. The measured values are not measured in the systems under investigation, namely mouse and chick, but in cell culture systems. For example, the $K_d$ value for RAR/RXR dimerization has been determined in HeLa cells. Because the $K_d$ value is the rate (in M) at which these complexes come apart, this is a first order reaction and so the stochastic "probabilistic rate" parameter $c_d$ is equal to $K_d$ (Gillespie, 1977). Note that these values are the ratio of the backwards to forward binding rate constants $c_b$ and $c_f$ . This is a typical state of affairs: the values $c_b$ and $c_f$ are very difficult to measure. This allows a bit of leeway in picking the forward and backwards binding, but the literature provides some typical forward values which adds credence to the values used and listed in Table 3.2 (Lauffenburger and Linderman, 1993).

It is not expected that the model results will be significantly different when newly measured parameters are incorporated in place of the estimated values. A sensitivity

analysis, in which the model is re-run with systematically varied parameters, shows that the model remain qualitatively unchanged for moderate changes in the parameters. This is encouraging, as biological systems are generally robust, and it would be unusual that the overall biological system would be overly sensitive to moderate changes in the concentrations or rates.

The half-lives for mRNA can range from minutes to hours and values for the *Hox* mRNA have not been measured. In this model the values of around 15-20 minutes were chosen as a typical half-life, numbers that are in line with other values in early embryogenesis (Davidson, 1986). The half-lives of the proteins in the network have not been measured and the values chosen were between 15 and 30 minutes. These numbers are again in an acceptable range for transcription factors (A. Varshavsky, personal communication). Similar values were used for the turnover of the receptors and complexes. With respect to the number of RARs and RXRs, values of around one thousand of each type were chosen (Lauffenburger and Linderman, 1993). No distinction is made between the α, β, and γ forms. The cofactors pbx and prep1 are treated as a single molecule, which the *Hox* proteins can bind with on the DNA.

| Parameter | Value used | Description | Equation Type |
|---|---|---|---|
| $c_0$ | 4.0 | Create RA | Rayleigh |
| $c_1$ | 10000000.0 | Bind RA to RAR | Second-order |
| $c_2$ | 0.00006 | Decay RA | First-order |
| $c_3$ | 0.0001 | Create RAR | Zeroth-order |
| $c_4$ | 0.00006 | Decay RAR | First-order |
| $c_5$ | 0.005 | Unbind RA from RAR | First-order |
| $c_6$ | 0.0004 | Decay BRAR | First-order |

| | | | |
|---|---|---|---|
| $c_7$ | 1000000000 | Bind dimer to *Hoxa1* DNA | Hill |
| $c_8$ | 3.0 | Unbind dimer from *Hoxa1* DNA | First-order |
| $c_9$ | 0.02 | Transcribe *Hoxa1* mRNA | First-order |
| $c_{10}$ | 0.0007 | Decay *Hoxa1* mRNA | First-order |
| $c_{11}$ | 0.005 | Translate *Hoxa1* protein | First-order |
| $c_{12}$ | 0.001 | Decay *Hoxa1* protein | First-order |
| $c_{13}$ | 100000000.0 | Bind dimer to *Hoxb1* DNA | Hill |
| $c_{14}$ | 0.5 | Unbind dimer from *Hoxb1* DNA | First-order |
| $c_{15}$ | 0.02 | Transcribe *Hoxb1* | First-order |
| $c_{16}$ | 0.001 | Decay *Hoxb1* mRNA | First-order |
| $c_{17}$ | 0.02 | Translate *Hoxb1* protein | First-order |
| $c_{18}$ | 100000000.0 | Bind *Hoxa1* complex to Hoxb1 DNA | Hill |
| $c_{19}$ | 0.3 | Unbind *Hoxa1* complex from *Hoxb1* DNA | First-order |
| $c_{20}$ | .02 | Transcribe *Hoxb1* protein | First-order |
| $c_{21}$ | 1000000.0 | Bind dimer to *Hoxb1* repression site | Hill |
| $c_{22}$ | 0.00003 | Unbind dimer from *Hoxb1* repression site | First-order |
| $c_{23}$ | 1000000000 | Bind *Hoxb1* complex to *Hoxb1* DNA | Hill |
| $c_{24}$ | 0.3 | Unbind *Hoxb1* complex from *Hoxb1* DNA | First-order |
| $c_{25}$ | 0.02 | Transcribe *Hoxb1* protein | First-order |
| $c_{26}$ | 0.004 | Decay *Hoxb1* protein | First-order |
| $c_{27}$ | 1000000.0 | Bind *Hoxb1* complex to *Hoxb2* DNA | Hill |
| $c_{28}$ | 0.03 | Unbind *Hoxb1* complex from *Hoxb2* DNA | First-order |
| $c_{29}$ | 0.02 | Transcribe *Hoxb2* mRNA | First-order |
| $c_{30}$ | 0.00001 | Decay *Hoxb2* mRNA | First-order |
| $c_{31}$ | 0.002 | Transcribe *Hoxb2* mRNA | First-order |
| $c_{32}$ | 0.004 | Decay *Hoxb2* protein | First-order |
| $c_{33}$ | 0.00000015 | Cell division | Zeroth-order |
| $c_{34}$ | 100000.0 | Activate *Krox20* | First-order |
| $c_{35}$ | 0.002 | Unactivate *Krox20* | First-order |
| $c_{36}$ | 0.2 | Transcribe *Krox20* mRNA | First-order |
| $c_{37}$ | 0.0003 | Decay *Hoxa1* mRNA | First-order |
| $c_{38}$ | 12000.0 | Bind *Hox* complex to *Krox20* repression site | Hill |
| $c_{39}$ | 0.003 | Unbind complex from Krox20 repression site | First-order |
| $c_{40}$ | 0.0001 | Translate *Krox20* protein | First-order |
| $c_{41}$ | 0.00001 | Decay *Krox20* protein | First-order |
| $c_{42}$ | 10000000.0 | Bind RA to RXR | First-order |
| $c_{43}$ | 0.0001 | Create RXR | Zeroth-order |
| $c_{44}$ | 0.00006 | Decay RXR | First-order |
| $c_{45}$ | 0.02 | Unbind RA from RXR | First-order |
| $c_{46}$ | 0.002 | Decay bound RXR | First-order |

| | | | |
|---|---|---|---|
| $c_{47}$ | 5000.0 | Bind BRXR to BRAR | Second-order |
| $c_{48}$ | 0.0001 | Unbind BRXR from BRAR | First-order |
| $c_{49}$ | 10.0 | Decay BRAR/BRXR dimer | First-order |
| $c_{50}$ | 10000000.0 | Bind *Hoxa1* protein to PBX complex | Second-order |
| $c_{51}$ | 0.02 | Unbind *Hoxa1*/PBX protein complex | First-order |
| $c_{52}$ | 0.009 | Decay *Hoxa1*/PBX protein complex | First-order |
| $c_{53}$ | 10000000.0 | Bind *Hoxb1* protein to PBX complex | Second-order |
| $c_{54}$ | 0.02 | Unbind *Hoxb1*/PBX protein complex | First-order |
| $c_{55}$ | 0.01 | Decay *Hoxb1*/PBX protein complex | First-order |
| $c_{56}$ | 0.01 | Create bare PBX complex | Zeroth-order |
| $c_{57}$ | 0.005 | Decay bare PBX complex | First-order |
| $K_1$ | 1000 | Threshold for ActivateA1 Hill function | N/A |
| $K_2$ | 1000 | Threshold for ActivateB1 Hill function | N/A |
| $K_3$ | 1000 | Threshold for SuperActivateB1 Hill function | N/A |
| $K_4$ | 10000 | Threshold for AutoActivateB1 Hill function | N/A |
| $K_5$ | 1000 | Threshold for ActivateB2 Hill function | N/A |
| $K_6$ | 100 | Threshold for repression functions | N/A |
| a1hill | 4.0 | Hill coefficient for ActivateA1 Hill function | N/A |
| b1hill | 4.0 | Hill coefficient for ActivateB1 Hill function | N/A |
| b1auto | 6.0 | Hill coefficient for AutoActivateB1 Hill function | N/A |
| b2hill | 2.0 | Hill coefficient for ActivateB2 Hill function | N/A |
| rephill | 4.0 | Hill coefficient for repression functions | N/A |

**Table 3.2 Parameters used in the simulation**. The type of reaction and the associated value used is listed. As examples, the function for binding RA to the retinoic acid Receptor RAR is $a_1 = 1\times10^7\{\text{RA}\}\{\text{RAR}\}$ where $\{\ \}$ denotes the number of molecules of each type. The first order reaction of the *Hoxa1* mRNA decaying is given by $a_{10} = 7\times10^{-4}\{\text{mHoxa1}\}$, and the Hill activation of *Hoxb2* is given by

$$a_{27} = 1\times10^{6}\frac{\{\text{Hoxb1 pbx complex}\}^{2}}{\left(1\times10^{6}+\{\text{Hoxb1 pbx complex}\}^{2}\right)} * \{\text{Hoxb1 pbx complex}\} * \{\text{Hoxb2 DNA}\}$$

In implementing the repression of *Hoxb1*, the simulation started this mechanism around 8.0 dpc because of the current understanding that the repression starts later than the activation (R. Krumlauf, personal communication). The *Hoxa1* and *Hoxb1* repression for *Krox20* is also started at around 8.0 dpc to ensure the establishment of *Hoxa1* and *Hoxb1* before the *Krox20* expression.

## Results

The early *Hox* genes first appear around 7.75 dpc (headfold) and the patterns of *Hoxa1*, *Hoxb1, Hoxb2* and *Krox20* stabilize by 8.5 dpc (~10 somites). Using the network shown in Figure 3.5, the goal was to capture this wild-type expression. Accordingly, the model was run for a simulated time of 18 hours. The model is one dimensional along the rostral-caudal axis of the embryo. Running the simulation with different random number seeds show that the model is not overly sensitive to the initial seed values. In the figures, a number of these independent runs are assembled side-by-side to construct a two-dimensional sheet of cells that resemble the tissue (with a medio-lateral dimension). This offers insights into the expected two-dimensional pattern of gene expression in the hindbrain and displays the variability in the results.

A custom built notebook in *Mathematica* (found in Appendix D) was used to display the results of the simulations. The raw data (the number of molecules of each

type in each cell) has been scaled to numbers between 0 and 1 by dividing by the maximum value in that data set. This allows the creation of a color shading so that differences in levels of molecules are clear. The results are displayed in an easy to understand format: a virtual dynamic *in situ*. Because the maximum value used to scale the data is on the order of tens to a couple hundred molecules, the color variations that are seen in the figures and the movie may in fact be too small to distinguish in a laboratory setting using conventional *in situ* staining.

## Wild Type

Figure 3.7 presents the dynamics of the model concerning the emergence of *Hoxa1*, *Hoxb1*, *Hoxb2* and *Krox20*, over time from approximately 7.75 dpc to 8.5 dpc. The figure presents single frames from the movie wt.mov. Along with all the other movies referenced in this thesis, wt.mov can be found on the included CD-ROM. The movie offers a dynamic view of the mRNA and RA in the developing hindbrain. Each rhombomere starts out with 20 cells, and the presumptive boundary is clearly marked. Even though the movies and figures show the mRNA levels, the model also tracks the amount of protein, bound and unbound complexes, and bound and unbound receptors, and any of these data can be displayed in a similar manner.

The low levels of *Hoxa1*, *Hoxb1* and *Hoxb2* mRNA in r4 and r5 are first seen soon after the simulation starts when the RA sweeps across the cells (Figure 3.7A). After the mRNA is translated into protein and subsequently forms a complex with pbx and prep1, it can then bind to the DNA. The effects of the *Hoxa1* binding site on *Hoxb1* and the *Hoxb1* auto-regulatory loop are seen next, namely the higher levels of *Hoxb1* in r4

(Figure 3.7B). By 8 dpc the RA has long since vanished from the hindbrain and consequently the RAR/RXR dimers are no longer being created. This is the main reason that *Hoxa1* starts to vanish from the hindbrain. The lack of available dimers also contributes to *Hoxb1* vanishing from r5, as does the late repression mechanism (Figure 3.7C). Now that *Hoxa1* and *Hoxb1* no longer repress *Krox20* in r5, its expression rises and subsequently brings up *Hoxb2* in r5. At about this time, *Hoxb2* has appeared in r4 due to the up-regulation by *Hoxb1* (Figure 3.7D). The ending expression pattern of the five genes at 8.5 dpc (Figure 3.7E) is very similar to reported patterns (Lumsden and Krumlauf, 1996).

It is clear from laboratory data that cells sometimes "misfire," and using this simulation it is possible to see the consequences of such misfirings. In Figure 3.7, (A, B, D, E) the cell marked with an arrow deviates from its normal fate and ends up not expressing any genes. At the same time, there are other cells that appear to misfire early, exemplified by low levels of expression, but later recover. This is exemplified by the lone white cell in the r4 *Hoxb1* data at 8.15 dpc. For whatever reason, it was not expressing *Hoxb1* at this timepoint, but it recovers by 8.5 dpc. Both of these events are known to happen in biological systems, and it is encouraging to see this behavior in the model, as these events are not captured with conventional modeling methods. This result suggests that fluctuations are a factor in the network under investigation.

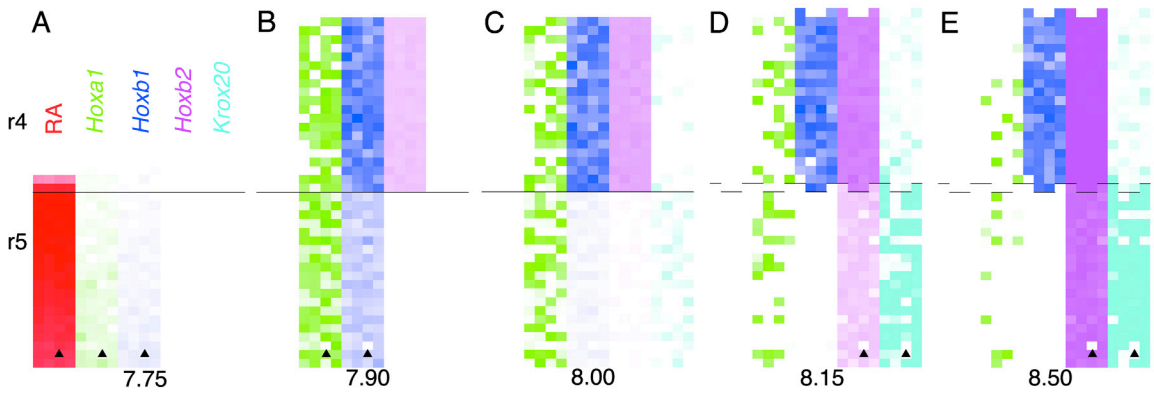


**Figure 3.7 Simulated wildtype mRNA and RA patterns from 7.75 dpc to 8.5 dpc (A-E)** Selected frames from the computer generated time-lapse movie wt.mov. Four runs of the simulation were required to create this picture, with each run contributing a row of RA, *Hoxa1*, *Hoxb1*, *Hoxb2* and *Krox20* data for each timepoint. Notice that sometime between 8 dpc and 8.15 dpc there is a cell division in r5 in the first and fourth data sets. This can be seen most clearly in the *Hoxb2* and *Krox20* data at 8.5 dpc. When a cell divides, its resources are normally distributed between the daughter cells. The data for the marked cell was generated during one of the simulations, and the consequences of this cell misfiring can clearly be seen **(A)** At 7.75 dpc there is an abundance of RA and low levels of both *Hoxa1* and *Hoxb1* expression are evident in the marked cell. **(B)** The expression of *Hoxa1* and *Hoxb1* fades in this cell by 7.90 dpc, a bit earlier than some of its neighbors. **(E)** By 8.5 dpc the cell has failed to initiate its proper expression of *Krox20* and *Hoxb2***.** This result suggests that fluctuations are important in the network under investigation.

## *In Silico* Experiments

The versatility of the computer simulation also allows for the possibility of performing *in silico* experiments. The results of two experiments are reported here and the simulation output shows that the results are similar to their corresponding *in vivo* experiments. In addition, the simulation suggests results that have not been reported in the laboratory, and these predictions warrant further investigation *in vivo*.

## *Hoxb1* Mutant

In the investigation of the cross-regulation of *Hoxb2* by *Hoxb1* in r4 (Maconochie et al., 1997), the authors showed that the up-regulation of *Hoxb2* in r4 is lost in *Hoxb1* mutants. Duplicating this experiment *in silico* requires a minimum number of changes to the model, and is accomplished by not allowing any transcription factors to bind to the *Hoxb1* DNA. The input parameters used were the same as in the wild type (Table 3.2). In stills taken from the movie Hoxb1mutant.mov, it starts as in the wild type: the RA comes through the hindbrain at 7.75 dpc and induces the expression of *Hoxa1*. However, because the *Hoxb1* gene is "turned off," there is no *Hoxb1* expression (Figure 3.8A). Later on, as reported in the literature, *Hoxb2* is absent from r4. It is also clear that *Krox20* fails to be well repressed in r4 (Figure 3.8B). By 8.5 dpc, *Hoxb1* expression is still absent and high levels of *Krox20* are firmly established in r4 (Figure 3.7C). This last result has yet to be thoroughly investigated, but there are two ways that this could be tested in the laboratory. The first is to acquire the mice used in the study and check the *Krox20* expression, while the second is to create a DNA construct that mimics this type of behavior in chick. Acquiring the mutant mice is not an easy, quick, or inexpensive

task, and so the second approach was taken. The attempt to perform this perturbation experiment is fully described in Appendix A.

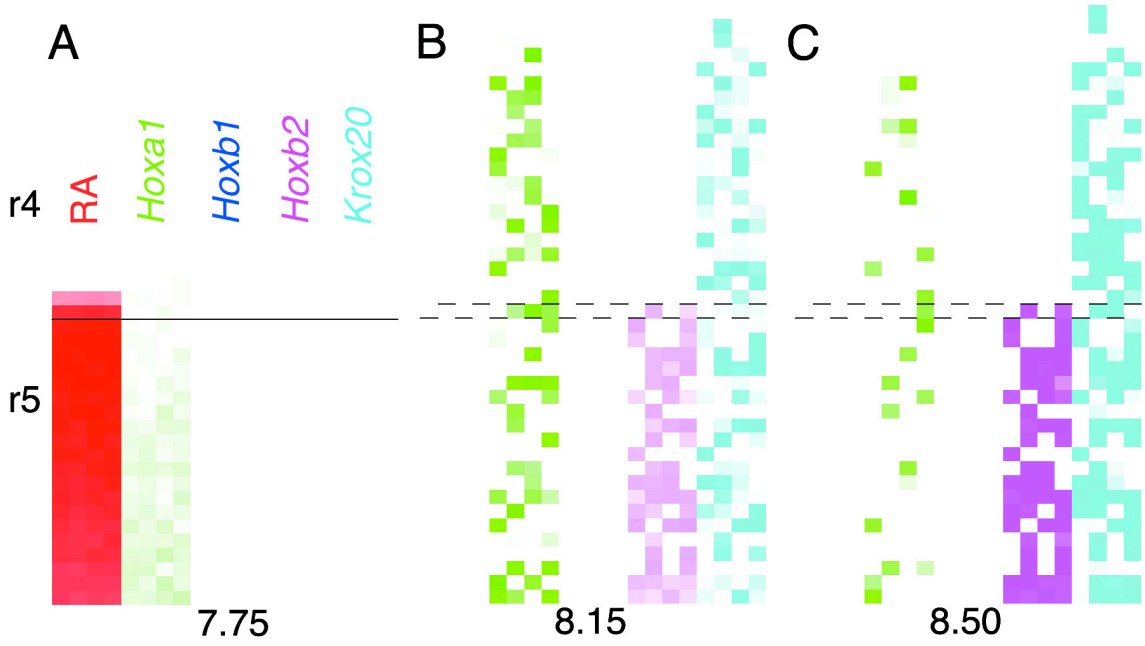


**Figure 3.8 Simulated *Hoxb1* mutant mRNA expression patterns. (A-C)** Selected frames from the computer generated time-lapse movie Hoxb1mutant.mov. This data set shows cell division having occurred in both r4 and r5. Besides affecting the *Hoxb2* expression in r4, the *Hoxb1* mutant also has an effect on *Hoxb2* and *Krox20* in r5. **(B)** The levels of *Krox20* are lower at 8.15 dpc than in the wild-type (Figure 3.6D). **(C)** By 8.5 dpc, the levels of *Krox20* and *Hoxb2* are noticeably lower than the wild type (Figure 3.6E). The observation on the level of *Krox20* expression is a prediction that can be tested in the laboratory.

## 5' RARE Mutant

The effects of a selected deletion in the *Hoxb1* 5' RARE showed that the RARE plays a role in the r4 restricted expression of *Hoxb1* (Studer et al., 1994). In this work the authors showed that if the construct lacked the 5' RARE, the reporter expression spread to r3 and r5. Further study suggests that the r3/r5 repressor region that contains the RARE is activated later than the 3' enhancer element (R. Krumlauf, personal communication). Duplicating this experiment using the model is again a simple matter, and is accomplished by not turning on the repressor. As in the *Hoxb1* mutant experiment described above, the parameters used were the same as in the wild type (Table 3.2). The stills from the movie RAREmutant.mov show that the expression pattern looks normal at 7.75 dpc (Figure 3.9A). However, at 8.0 dpc the repression mechanism is not turned off, and by 8.15 dpc the expression of *Hoxb1* in r5 is still strong (Figure 3.9B). By 8.5 dpc, the *Hoxb1* expression has faded in r4 somewhat due to the lack of available RAR/RXR dimers, but is still noticeable (Figure 3.9C). In addition, there is once again a change in the pattern of *Krox20*, but this time there are lower expression levels in r5 (Figure 3.9C). This is due to the continued repression effects of *Hoxa1* and *Hoxb1*. This result has yet to be fully investigated in the laboratory.

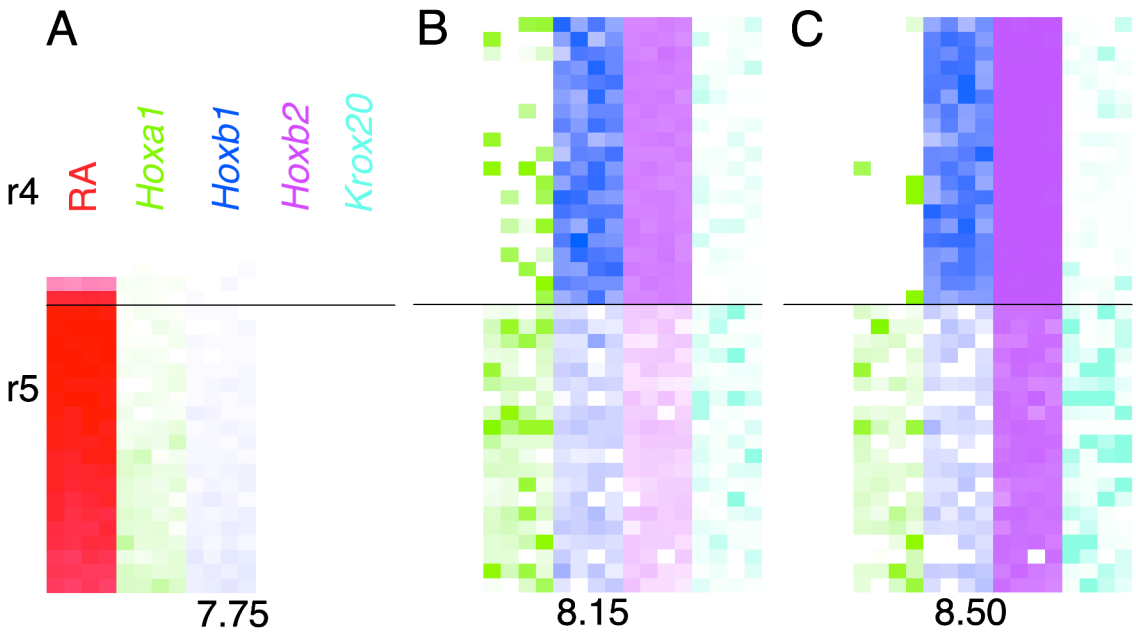


**Figure 3.9 Simulated expression patterns after inactivation of the 5' *Hoxb1* RARE (A-C)** Selected frames from the computer generated time-lapse movie RAREmutant.mov. By turning off the 5' RARE, there is a change in the levels of *Hoxa1* expression in r5. This occurs because the 3' and 5' RAREs are in effect fighting for the RAR/RXR dimers. This intriguing result needs to be more fully investigated. As in the wild type, it is easy to see downstream effects from cells that have misfired, most notably the patches where *Hoxa1* or *Hoxb1* are continuing to repress *Krox20*. **(A)** The behavior of the system mimics the wild-type at 7.75 dpc because the 5' RARE does not kick in until 8 dpc. **(B)** By 8.15 dpc, the expression of *Hoxb1* is still noticeable in r5, but the levels are low enough to allow *Krox20* expression to take hold. **(C)** The levels of *Krox20* in r5 are higher than in the wild type (Figure 3.7E). The effects of the *Hoxb1* RAREs

not having to compete for the dimers is clear by 8.5 dpc as evidenced by the higher levels of *Hoxa1* as compared to the wild type (Figure 3.7E).

## *Sensitivity Analysis*

A model that is presented with no analysis leaves something to be desired, and this section presents the results of a sensitivity analysis. There are two categories of conventional analysis possible: local and global sensitivity analysis. Local analysis is based upon evaluating the derivative of some output function with respect to any of the input variables at some fixed point in the space of the input variables. However, this approach is only really practical for linear models, and a local analysis is unable to gauge the impact of possible differences in the scales of the variations of the input variables. It has been recognized for several decades that when the model is nonlinear and the various input values are affected by uncertainties of different orders of magnitude, a global sensitivity analysis should be used (Cukier, 1973).

Recall that the simulation consists of over 75 input parameters, and the output consists of the quantities of 19 different molecular species for each of forty cells cell at each of the1080 time points, or over 800,000 outputs. Doing a sensitivity analysis over all these parameters would prove intractable. Because of this, the data was compacted before the analysis was run.

First of all, each of the 40 cells is assigned either an r4 or an r5 identity, and so the cells were grouped by their rhombomeric identity and the number of molecules for each species was averaged over all the cells. Next, since the movies and the experiments

are primarily concerned with the amount of messenger RNA that is in these cells, special attention was focused on the mRNA and how the variation in the parameters affected these quantities. Finally, instead of looking at 1080 time points, the data was downsampled to 54 time points (one for every 20 minutes instead of every minute).

## Measure of Importance

The global analysis initially tried is one that is based on a "measure of importance" called S. In this type of approach, all the parameters are varied simultaneously and the sensitivity of the output variables is measured over the entire range of each input parameter. It allows the output variance to be broken up into contributions due to individual parameters or combinations of parameters (Homma, 1996). As an illustrating example, let $\mathbf{y} = f(\mathbf{x})$ be the black box of the simulation to be evaluated, where $\mathbf{x} = (x_1, x_2, x_3)$, and **y** is an output vector of size m. Suppose the total variance of $f(\mathbf{x})$ is V. It is possible to write V as a sum of the variances that contribute to the total

$$V = V_1 + V_2 + V_3 + V_{12} + V_{23} + V_{13} + V_{123} \tag{3.51}$$

Then $S_1 = V_1 / V$ is the fraction of the total variance due to the parameter $x_1$ averaged over all the parameters and it is called the first order term for the parameter $x_1$. In a similar vein, $S_{12} = V_{12} / V$ is the fraction of the total variance due to the coupling of the parameters $x_1$ and $x_2$ and is called the second order term for the parameters $x_1$ and $x_2$. These variables can be combined to produce the sensitivity indices for each of the input variables by computing

$$S_{T,1} = S_1 + S_{12} + S_{13} + S_{123} \tag{3.52}$$

Calculating these variables is a straightforward, albeit time-consuming exercise. Notice that the $S_i$ are all positive and sum to one, with the most important factors having the largest contribution.

This analysis was performed on the model and the results were not surprising. In Table 3.3 are several sensitivity indices computed for the mRNA in each of the rhombomeres.

| | | $S_i$ value for mRNA for | | | | |
|---|---|---|---|---|---|---|
| Parameter | Rhombomere | *Hoxa1* | *Hoxb1* | *Hoxb2* | *Krox20* | Sum |
| $K_1$ | 4 | 0.25390 | 0.06763 | 0.04099 | 0.10119 | 4.18192 |
| | 5 | 0.04755 | -0.02082 | 0.01945 | 0.00594 | -0.04203 |
| $c_1$ | 4 | -0.33742 | -0.47741 | -0.47995 | -0.40615 | -5.43707 |
| | 5 | -0.37504 | -0.37020 | -0.49049 | -0.47657 | -6.16916 |
| $c_{13}$ | 4 | 0.34952 | 0.06243 | 0.04217 | 0.07847 | 4.78133 |
| | 5 | 0.36623 | -0.09437 | 0.02032 | 0.00078 | 1.03525 |
| $c_{26}$ | 4 | 0.11857 | 0.12154 | 0.06911 | 0.07454 | 3.90804 |
| | 5 | -0.03849 | 1.13157 | 0.02944 | 0.09681 | 1.58401 |

**Table 3.3 Sensitivity Analysis using the Measure of Importance**. This analysis does not appear to be one that can be employed for a simulation that is subject to stochastic variations.

In direct defiance of the theoretical analysis, the $S_i$ values are not all positive and they do not sum to one. The result of this analysis confirmed an important aspect of the model: the inherent fluctuations of the system can at times have stronger effects than a change in a parameter, and the stochasticity of the simulation plays a synergistic role with the change of the parameters. Accordingly, this type of analysis does not seem to address the question at hand, and it another type of analysis was used to examine the effects of changing the parameters.

## Excess Variance

Because the simulation is fundamentally subject to fluctuations, it is challenging to determine the effect on the output due to a change in a parameter. But this can be addressed using an excess variance based analysis. Let $v_j(\mathbf{x},t)$ denote an output of interest from the simulation at time *t* with input vector **x** and random number seed *j*. Let $v(\mathbf{x};x_i,t)$, denote the output from the simulation at time t with the input value $x_i$ perturbed but all other inputs the same, and the default random number seed. Computing the mean of the squared difference of these values,

$$E_j\left[\left(v_j(\mathbf{x},t)-v(\mathbf{x};x_i,t)\right)^2\right] \tag{3.53}$$

yields a response curve. This value is a consistent estimator (*i.e.*, the probability of the estimated value and the true value of the population parameter not lying within any arbitrary positive constant c units of each other approaches zero as the sample size tends

to infinity), and identifies the parameters that have an important effect in contributing to the output values of interest.

This calculation was performed for the levels of mRNA for *Hoxa1*, *Hoxb1*, *Hoxb2* and *Krox20*. The analysis was only performed for the $c_\mu$ values because previous investigations while building the model had shown that these were the most important in determining the system behavior. The analysis was performed for each of the 4 target variables, for each of the rhombomeres, and to allow for legibility of the plots, the $c_\mu$ values were examined 10 at a time. This resulted in a total of 48 figures, but in the interest of space, not all of the plots are shown. Typical plots of these results are shown in Figures 3.10, 3.11 and 3.12 below, and the results of the entire investigation are summarized in Table 3.4.

Figure 3.10 shows the normal state of affairs; none of the $c_\mu$ $(\mu = 40 \ldots 49)$ values plays a significant role in the expression of the messenger RNA for *Hoxb1* in rhombomere 4. But compare this plot to Figure 3.11. In this figure it is clear that $c_{53}$ plays a noticeable role on the level of mRNA for *Hoxb1* in rhombomere 4.

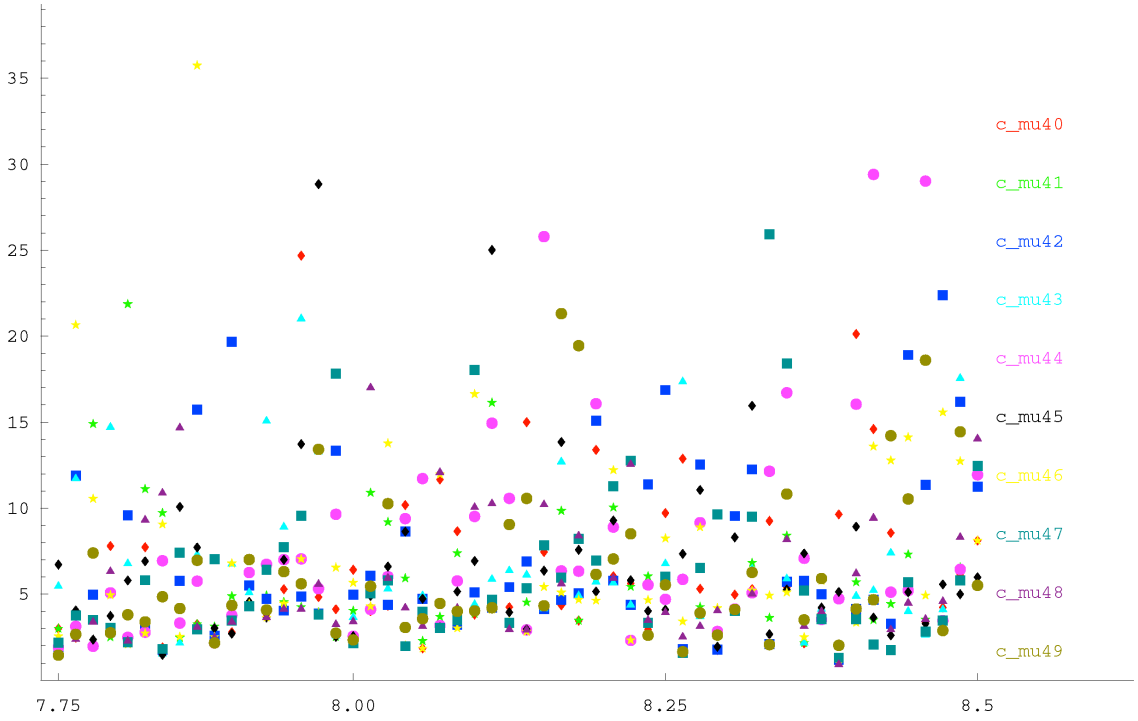


**Figure 3.10** Effects of $c_\mu$ values on mRNA for *Hoxb1* expression in rhombomere 4. The legend denotes the color of the response for a particular parameter, and in this instance none of the parameters has a significant effect. The x axis is time (dpc), and the y axis is the response value (computed in 3.4).

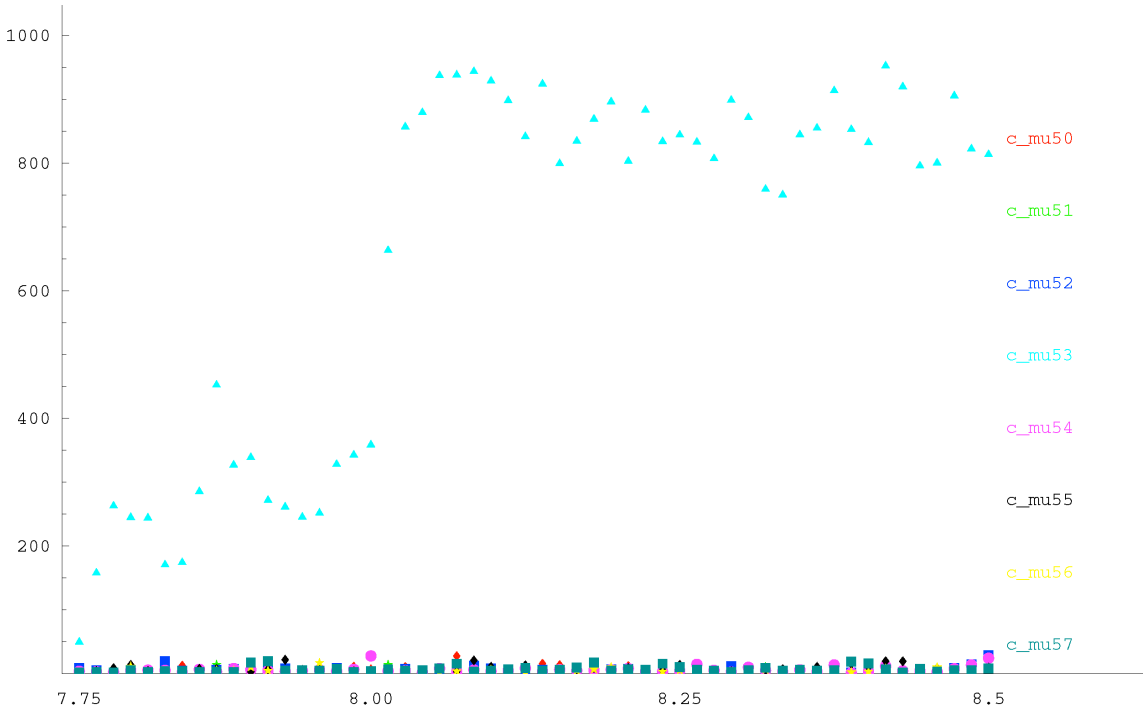


**Figure 3.11**: Effects of $c_\mu$ variables on the amount of mRNA for *Hoxb1* in rhombomere 4. The parameter $c_{53}$, which is part of the auto-regulatory loop, is by far the dominant parameter in this set. The x axis is time, and the y axis is the mean response values (computed in 3.4).

Looking at the list of values, $c_{53}$ is the stochastic rate coefficient for the formation of the *Hoxb1* protein/pbx/end complex, *i.e.*, $c_{53}$ is part of the auto-regulatory loop for *Hoxb1*, and it is no surprise that this parameter makes a difference in the expression of mRNA for *Hoxb1*. Compare this to Figure 3.12, which shows the effects of the same $c_\mu$ values on the mRNA for *Hoxb1*, but this time in rhombomere 5 in which there is no auto-

regulatory loop for *Hoxb1*. The contributions of the values are lower overall, and the repression mechanisms that turns on at day 8.0 makes a noticeable difference.

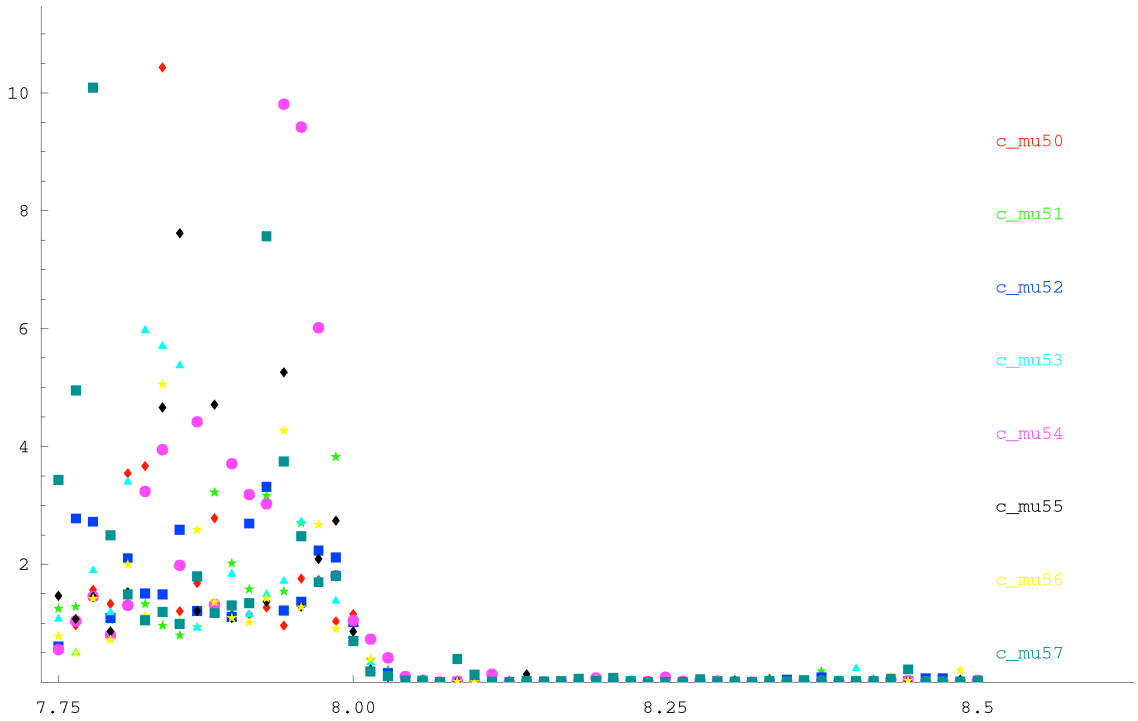


**Figure 3.12**: Effects of $c_\mu$ variables on the amount of mRNA for *Hoxb1* in rhombomere 5. Notice that none of the parameters has a major effect on the mRNA levels, and when the repression mechanisms start at 8 dpc, all of the effects virtually vanish. The x axis is time, and the y axis is the mean response values (computed in 3.4).

The $c_\mu$ values that play a role on the levels of the target variable are not surprising. For instance, the transcription of mRNA for *Hoxa1* from the activated form of the gene is important in both rhombomeres.

| Target | Rhombomere | Significant $c_\mu$ value | Related Function |
|---|---|---|---|
| *Hoxa1* | 4 | $c_7$ | ActivateA1 |
| | | $c_9$ | TranscribeA1 |
| | | $c_{10}$ | DecaymA1 |
| | 5 | $c_7$ | ActivateA1 |
| | | $c_9$ | TranscribeA1 |
| | | $c_{49}$ | DecayDimer |
| *Hoxb1* | 4 | $c_{16}$ | DecaymB1 |
| | | $c_{25}$ | TranscribeAutoB1 |
| | | $c_{53}$ | Complexb1 |
| | 5 | $c_{15}$ | TranscribeB1 |
| | | $c_{16}$ | DecaymB1 |
| *Hoxb2* | 4 | $c_{12}$ | Decaya1 |
| | | $c_{29}$ | TranscribeB2 |
| | | $c_{30}$ | DecaymB2 |
| | | $c_{53}$ | Complexb1 |
| | 5 | $c_{16}$ | DecaymB1 |
| | | $c_{29}$ | TranscribeB2 |
| | | $c_{30}$ | DecaymB2 |
| *Krox20* | 4 | $c_{17}$ | Translate SuperB1 |
| | | $c_{53}$ | Complexb1 |
| | 5 | $c_{25}$ | TranscribeAutoB1 |
| | | $c_{37}$ | DecaymKrox |

**Table 3.4**: **Effects of $c_\mu$ variables on the mRNA**. None of these variables is a great surprise. For instance, the parameters that change the mRNA for *Hoxb1* in r4 more than 20% above the baseline are the ones that affect the rate of decay of the mRNA for *Hoxb1*, the strength of the auto-regulatory loop, and the rate of Hoxb1/Prep complex formation. This last one might seem a little odd at first, until it is noted that the formed complex is required for the triggering of the auto-regulatory loop.

## *Summary*

The stochastic simulation model captures the timing of several *Hox* gene expression patterns in wild-type animals, and *in silico* simulations performed as a check of key interactions produced results similar to *in vivo* experiments. In addition, the *in silico* experiments yield intriguing results that bear further investigation in the laboratory.

The model simulations suggest that a transitory early release of RA may be sufficient to initiate the *Hox* genes. During the investigation of functions for modeling the RA source, it became clear that initiation of the network only required the RA source to stay on for as few as 3 minutes. All that was needed was enough RA to bind the receptors in r4 and r5 and proper expression of the target genes was the result. This refinement of the RA gradient hypothesis fits well with recent work on blocking RAR with a chemical antagonist in which the authors made a careful study of concentration and time dependent effects of the blocking agent using morphology and gene expression as assays. Chick embryos treated with the agent at HH stage 6 (Hamburger and Hamilton, 1951) do not express *Krox20* in r5, but treatment at HH stage 7 permits r5 expression (Dupe and Lumsden, 2001). Thus, the *Krox20* insensitivity to a later change in RA fits well with our model predictions: once the network was established early on proper r5 expression of *Krox20* was evident.

## *References for Chapter 3*

Allegretto, E. A., McClurg, M. R., Lazarchik, S. B., Clemm, D. L., Kerner, S. A., Elgort, M. G., Boehm, M. F., White, S. K., Pike, J. W., and Heyman, R. A. (1993). Transactivation properties of retinoic acid and retinoid X receptors in mammalian cells and yeast. Correlation with hormone binding and effects of metabolism. *J Biol Chem* **268,** 26625-33.

Arkin, A., Ross, J., and McAdams, H. H. (1998). Stochastic kinetic analysis of developmental pathway bifurcation in phage lambda-infected *Escherichia coli* cells. *Genetics* **149,** 1633-48.

Barrow, J. R., Stadler, H. S., and Capecchi, M. R. (2000). Roles of Hoxa1 and Hoxa2 in patterning the early hindbrain of the mouse. *Development* **127,** 933-44.

Bell, E., Wingate, R. J., and Lumsden, A. (1999). Homeotic transformation of rhombomere identity after localized Hoxb1 misexpression. *Science* **284,** 2168-71.

Berggren, K., McCaffery, P., Drager, U., and Forehand, C. J. (1999). Differential distribution of retinoic acid synthesis in the chicken embryo as determined by immunolocalization of the retinoic acid synthetic enzyme, RALDH-2. *Dev. Biol.* **210,** 288-304.

Berthelsen, J., Zappavigna, V., Ferretti, E., Mavilio, F., and Blasi, F. (1998a). The novel homeoprotein Prep1 modulates Pbx-Hox protein cooperativity. *Embo J* **17,** 1434-45.

Berthelsen, J., Zappavigna, V., Ferretti, E., Mavilio, F., and Blasi, F. (1998b). The novel homeoprotein Prep1 modulates Pbx-Hox protein cooperativity. *Embo. J.* **17,** 1434-45.

Cukier, R. I., Fortuin, C.M., Schuler, K.E., Petschek, A.G., and Schaibly, J.K. (1973). Study of the sensitivity of coupled reaction systems to uncertainties in rate coefficients, part I. *Journal of Chemical Physics* **59,** 3873-3878.

Davidson, E. H. (1986). "Gene activity in early development." Academic Press, Orlando.

Davidson, E. H. (2001). "Genomic Regulatory Systems." Academic Press, San Diego.

Depoix, C., Delmotte, M. H., Formstecher, P., and Lefebvre, P. (2001). Control of retinoic acid receptor heterodimerization by ligand-induced structural transitions. A novel mechanism of action for retinoid antagonists. *J. Biol. Chem.* **276,** 9452-9.

Duboule, D. (1994). Guidebook to the Homeobox Genes. Sambrook & Tooze, Oxford.

Dupe, V., and Lumsden, A. (2001). Hindbrain patterning involves graded responses to retinoic acid signalling. *Development* **128,** 2199-208.

Ferretti, E., Marshall, H., Popperl, H., Maconochie, M., Krumlauf, R., and Blasi, F. (2000). Segmental expression of Hoxb2 in r4 requires two separate sites that integrate cooperative interactions between Prep1, Pbx and Hox proteins. *Development* **127,** 155-66.

Frasch, M., Chen, X., and Lufkin, T. (1995). Evolutionary-conserved enhancers direct region-specific expression of the murine Hoxa-1 and Hoxa-2 loci in both mice and Drosophila. *Development* **121,** 957-74.

Fraser, S., Keynes, R., and Lumsden, A. (1990). Segmentation in the chick embryo hindbrain is defined by cell lineage restrictions. *Nature* **344,** 431-5.

Gallera, J. (1971). Primary induction in birds. *Adv. Morphogenet.* **9,** 149-180.

Gavalas, A., and Krumlauf, R. (2000). Retinoid signalling and hindbrain patterning. *Curr. Opin. Genet. Dev.* **10,** 380-386.

Gilbert, S. F. (1997). "Developmental Biology." Sinauer Associates, Sunderland, Mass.

Gillespie, D. T. (1977). Exact Stochastic Simulation of Coupled Chemical Reactions. *Journal of Physical Chemistry* **81,** 2340-2361.

Green, N. C., Rambaldi, I., Teakles, J., and Featherstone, M. S. (1998). A conserved C-terminal domain in PBX increases DNA binding by the PBX homeodomain and is not a primary site of contact for the YPWM motif of HOXA1. *J. Biol. Chem.* **273,** 13273-9.

Guthrie, S., and Lumsden, A. (1991). Formation and regeneration of rhombomere boundaries in the developing chick hindbrain. *Development* **112,** 221-9.

Hamburger, V., and Hamilton, H. (1951). A series of normal stages in the development of the chick embryo. *J. Morph.* **88,** 49-92.

Hill, A. V. (1910). Possible effects of the aggregation of the molecules of haemoglobin on its dissociation curves. *J. Physiol.* **40,** iv-viii.

Homma, T. a. S., A. (1996). Importance measures in global sensitivity analysis of nonlinear models. *Reliability Engineering and System Safety* **52,** 1-17.

Kalter, H., and Warkany, J. (1959). Experimental production of congenital malformations in mammals by metabolic procedure. *Physiology Review* **39,** 69-115.

Kastner, J. C., Solomon, J. E., and Fraser, S. E. (2002). Modeling a Hox Gene network *in silico* using a Stochastic Simlulation Algorithm. *Developmental Biology* **246,** 122-131.

Langston, A. W., and Gudas, L. J. (1992). Identification of a retinoic acid responsive enhancer 3' of the murine homeobox gene Hox-1.6. *Mech. Dev.* **38,** 217-27.

Lauffenburger, D. A., and Linderman, J. J. (1993). "Receptors." Oxford University Press, Oxford.

Leid, M., Kastner, P., and Chambon, P. (1992a). Multiplicity generates diversity in the Retinoic Acid signaling pathways. *Trends in Biochemical Sciences* **17,** 427-433.

Leid, M., Kastner, P., R., L., Nakshatri, H., Saunders, M., Zacharewski, T., Chen, J. Y., Staub, A., Garnier, J. M., Mader, S., and Chambon, P. (1992b). Purification, cloning, and RXR identity of the HeLa-cell factor with which RAR or TR heterodimers to bind target sequences efficiently. *Cell* **68,** 377-395.

Lufkin, T. (1997). Transcriptional regulation of vertebrate Hox genes during embryogenesis. *Crit. Rev. Eukaryot. Gene. Expr.* **7,** 195-213.

Lumsden, A., and Krumlauf, R. (1996). Patterning the vertebrate neuraxis. *Science* **274,** 1109-15.

Maconochie, M., Nonchev, S., Morrison, A., and Krumlauf, R. (1996). Paralogous Hox genes: function and regulation. *Annu. Rev. Genet.* **30,** 529-56.

Maconochie, M. K., Nonchev, S., Studer, M., Chan, S. K., Popperl, H., Sham, M. H., Mann, R. S., and Krumlauf, R. (1997). Cross-regulation in the mouse HoxB complex: the expression of Hoxb2 in rhombomere 4 is regulated by Hoxb1. *Genes Dev.* **11,** 1885-95.

Maden, M. (1999). Heads or tails? Retinoic acid will decide. *Bioessays* **21,** 809-12.

Mader, S., Chen, J. Y., Chen, Z., White, J., Chambon, P., and Gronemeyer, H. (1993). The patterns of binding of RAR, RXR and TR homo- and heterodimers to direct

repeats are dictated by the binding specificites of the DNA binding domains. *Embo. J.* **12,** 5029-41.

Marshall, H., Studer, M., Popperl, H., Aparicio, S., Kuroiwa, A., Brenner, S., and Krumlauf, R. (1994). A conserved retinoic acid response element required for early expression of the homeobox gene Hoxb-1. *Nature* **370,** 567-71.

McGinnis, W., Garber, R. L., Wirz, J., Kuroiwa, A., and Gehring, W. J. (1984). A Homologous Protein-Coding Sequence in *Drosophila* Homeotic Genes and Its Conservation in Other Metazoans. *Cell* **37,** 403-408.

McGinnis, W., and Krumlauf, R. (1992). Homeobox genes and axial patterning. *Cell* **68,** 283-302.

Murphy, P., and Hill, R. E. (1991). Expression of the mouse labial-like homeobox-containing genes, Hox 2.9 and Hox 1.6, during segmentation of the hindbrain. *Development* **111,** 61-74.

Neuteboom, S. T., and Murre, C. (1997). Pbx raises the DNA binding specificity but not the selectivity of antennapedia Hox proteins. *Mol. Cell. Biol.* **17,** 4696-706.

Niederreither, K., Subbarayan, V., Dolle, P., and Chambon, P. (1999). Embryonic retinoic acid synthesis is essential for early mouse post-implantation development. *Nat. Genet.* **21,** 444-8.

Nonchev, S., Maconochie, M., Vesque, C., Aparicio, S., Ariza-McNaughton, L., Manzanares, M., Maruthainar, K., Kuroiwa, A., Brenner, S., Charnay, P., and Krumlauf, R. (1996a). The conserved role of Krox-20 in directing Hox gene expression during vertebrate hindbrain segmentation. *Proc Natl Acad Sci U S A* **93,** 9339-45.

Nonchev, S., Vesque, C., Maconochie, M., Seitanidou, T., Ariza-McNaughton, L., Frain, M., Marshall, H., Sham, M. H., Krumlauf, R., and Charnay, P. (1996b). Segmental expression of Hoxa-2 in the hindbrain is directly regulated by Krox-20. *Development* **122,** 543-54.

Pellerin, I., Schnabel, C., Catron, K. M., and Abate, C. (1994). Hox proteins have different affinities for a consensus DNA site that correlate with the positions of their genes on the hox cluster. *Mol. Cell. Biol.* **14,** 4532-45.

Petkovich, M., Brand, N. J., Krust, A., and Chambon, P. (1987). A human retinoic acid receptor which belongs to the family of nuclear receptors. *Nature* **330,** 444-450.

Phelan, M. L., Rambaldi, I., and Featherstone, M. S. (1995). Cooperative interactions between HOX and PBX proteins mediated by a conserved peptide motif. *Mol Cell Biol* **15,** 3989-97.

Popperl, H., Bienz, M., Studer, M., Chan, S. K., Aparicio, S., Brenner, S., Mann, R. S., and Krumlauf, R. (1995). Segmental expression of Hoxb-1 is controlled by a highly conserved autoregulatory loop dependent upon exd/pbx. *Cell* **81,** 1031-42.

Ronshaugen, M., McGinnis, N. and McGinnis, W. (2002). Hox protein mutation and macroevolution of the insect body plan. *Nature*.

Rorsman, P., Eliasson, L., Renström, E., Gromada, J., Barg, S., and Göpel, S. (2000). The Cell Physiology of Biphasic Insulin Secretion. *News Physiol. Sci.* **15,** 72-77.

Schneider-Maunoury, S., Topilko, P., Seitandou, T., Levi, G., Cohen-Tannoudji, M., Pournin, S., Babinet, C., and Charnay, P. (1993). Disruption of Krox-20 results in alteration of rhombomeres 3 and 5 in the developing hindbrain. *Cell* **75,** 1199-214.

Sham, M. H., Vesque, C., Nonchev, S., Marshall, H., Frain, M., Gupta, R. D., Whiting, J., Wilkinson, D., Charnay, P., and Krumlauf, R. (1993). The zinc finger gene Krox20 regulates HoxB2 (Hox2.8) during hindbrain segmentation. *Cell* **72,** 183-96.

Studer, M., Gavalas, A., Marshall, H., Ariza-McNaughton, L., Rijli, F. M., Chambon, P., and Krumlauf, R. (1998). Genetic interactions between Hoxa1 and Hoxb1 reveal new roles in regulation of early hindbrain patterning. *Development* **125,** 1025-36.

Studer, M., Popperl, H., Marshall, H., Kuroiwa, A., and Krumlauf, R. (1994). Role of a conserved retinoic acid response element in rhombomere restriction of Hoxb-1. *Science* **265,** 1728-32.

Swindell, E. C., Thaller, C., Sockanathan, S., Petkovich, M., Jessell, T. M., and Eichele, G. (1999). Complementary domains of retinoic acid production and degradation in the early chick embryo. *Dev. Biol.* **216,** 282-96.

Van Allen, M. I., Kalousek, D. K., Chernoff, G. F., Juriloff, D., Harris, M., McGillivray, B. C., Yong, S. L., Langlois, S., MacLeod, P. M., Chitayat, D., Friedman, J. M., Wilson, R. D., McFadden, D., Pantzar, J., Ritchie, S., and Hall, J. G. (1993). Evidence for multi-site closure of the neural tube in humans. *Am. J. Med. Genet.* **47,** 723-743.

Wilkinson, D. G. (1993). Molecular mechanisms of segmental patterning in the vertebrate hindbrain and neural crest. *Bioessays* **15,** 499-505.

Wilkinson, D. G., Bhatt, S., Cook, M., Boncinelli, E., and Krumlauf, R. (1989). Segmental expression of Hox-2 homoeobox-containing genes in the developing mouse hindbrain. *Nature* **341,** 405-409.

# Chapter 4: Experiments

> Our real teacher has been and still is the embryo who is, incidentally, the only teacher who is always right.
>
> - Viktor Hamburger, 1968

## *Introduction*

When it comes to modeling biological systems, it is hardly ever the case that the modeler and the experimentalist are the same person. Instead, the work is usually done in collaboration. This leads to difficulties in that the modeler and the experimentalist don't always understand the intricacies and sticking points of the other discipline. Another problem is that the data used to build the model is not always the data ideally desired. For example, the binding coefficients listed in Table 3.1 were measured in cell cultures and not in chick or mouse. These facts lead the author to design and perform experiments relevant to the *Hox* system under investigation. Not only would the experiments be focused on testing and clarifying elements of the *Hox* model, they would also allow for better understanding of the problems and pitfalls in performing experiments in the biology lab.

In order to build the model it was necessary to make several assumptions. This chapter highlights one of those assumptions and describes an experiment that was performed to investigate and clarify an aspect of the model, namely the response of *Hoxa1* to retinoic acid (RA). This was accomplished by introducing a perturbation to the normal distribution of RA in the embryo.

The experiment described in this chapter was not the only model related experiment designed and pursued. In Appendix A, the reader will find the description of another experiment that was pursed. However, it turned out to be much more difficult than initially thought. This is not a rare occurrence in biology, and was one of the most important lessons about lab work that the author learned. While it is not possible to draw any definitive conclusions from the experiment in Appendix A, a great deal of work was done in paving the way for a continued investigation.

Before describing the retinoic acid perturbation experiment, there is a brief digression into the development of a method that made the experiments easier to perform.

## *Vital Stain*

Any sort of work on early chick and quail embryos is complicated by the fact that they are nearly transparent and very difficult to see against the yellow yolk. By HH stage 9 (Figure 3.1 7) there are enough signs in the surrounding tissue (the position of the area opaca for instance) to enable harvesting, but in order to easily perform other work including electroporation (described in Appendix A) or bead implantation (described below), something needs to be done in order to see the embryo.

A typical solution is to use a mixture of 10% of India ink in a balanced salt buffer, and when this is injected beneath the embryo there is enough contrast to easily see the embryo. The problem with this mixture is that India ink is known to be toxic, especially to younger embryos. If it is used in situations where the eggs are placed back into the incubator for more than a few hours, there will always be a decrease in viability. This is especially true after manipulation that is hard on the embryo, like electroporation (in

which electricity is delivered to the embryo) or bead implantation (in which the egg is open for a long time and the neural tube is ripped). Despite these known problems, there were no readily identifiable solutions presented in the literature, but an inquiry of other laboratory members suggested a possible solution. It came in the form of an ancient stash of pale blue food coloring. Using this as a vital stain increased the survival dramatically, but the contrast was poor and it was still very difficult to see the embryo. Nonetheless, this suggested that food coloring might be a good solution. Two different sources of food coloring were acquired; powder from Spectra Colors Corp, and liquid from the local supermarket. Along with India ink, these were used in an experiment to compare the resulting contrast and subsequent embryo viability.

Fertile chicken eggs from a local supplier (AA Laboratories) were incubated at 38° C until stages 4-6, usually between 36 and 40 hours. After removal from the incubator, the eggs were rinsed with 75% alcohol and 3 ml of albumin was removed. The egg was windowed and a few drops of Hanks' Buffered Salt Solution (HBSS) were added to the embryo to keep it moist. Approximately 100 mL of vital stain was injected under the embryo, and the resulting contrast was noted. The egg was then resealed with packing tape and replaced into the incubator. The embryos were harvested after 24 hours and assayed for viability. The results of this experiment are summarized in Table 4.1 below. It should be mentioned that eggs are not always resealed successfully, and some of the morphology problems and deaths are certainly due to the embryo drying out. This was a problem that applied to all of the experiments equally, and so these numbers were not separated out.

| Solution used | # Injected | Viable | % Viable |
|---|---|---|---|
| 10% India ink in HBSS | 9 | 7 | 78 % |
| Dec-a-Cake | 7 | 1 | 14% |
| 10% Dec-a-Cake in HBSS | 7 | 5 | 71% |
| Stock pale blue | 8 | 8 | 100% |
| 10% Spectra Red #40 In HBSS | 6 | 5 | 83% |
| 10% Spectra Blue #1 in HBSS | 5 | 5 | 100% |
| 10% Spectra Blue and Red in HBSS | 8 | 7 | 88% |

**Table 4.1 Vital stain results.** Viable is defined as embryos that are alive and look to have normal morphology. All of the solutions were diluted or mixed with HBSS. India ink actually faired better than expected. This was probably helped by the use of a freshly opened bottle: there is anecdotal evidence that using old ink decreases viability. The India ink solution also affected the surrounding tissue of an embryo, and there were often clumps of ink globules visible beneath the embryo. The Dec-a-Cake solution was the worst of the bunch, almost certainly due to the preservatives included, and while the diluted mix was much better than straight, it is still on the same level as India Ink. The stock pale blue provided excellent viability, but the contrast was very poor. The different mixes of the Spectra F.D&C. food coloring all resulted in good viability, and the contrast from the Blue and Red combination was very strong.

Since this experiment, the author has used food coloring exclusively for all experiments and the viability has been much better. In addition, the use of food coloring as a vital stain has collected a steady following in the Fraser and Bronner-Fraser laboratories and a half dozen people use it regularly. It has also been used at the Stowers Institute for Medical Research, and a member of the House Ear Institute used it to

successfully perform an experiment that was otherwise unsuccessful using India ink (A. Collazo, personal communication).

Having to solve the problem with the vital stain was just one of the many examples of the issues that need to be resolved before the experiment of interest can be performed.

## *Retinoic Acid Bead*

As mentioned in Chapter 3, the act of building the model caused a shift in thinking about how the system might become initiated. It became clear that a constant source of RA is not needed, and in fact a constant source leads to simulation results that are in disagreement with laboratory results. To better understand the connection between RA and *Hoxa1*, an experiment was undertaken to introduce RA into the system and determine the effects on *Hoxa1* expression. *Hoxa1* was picked as the assay because it is the first *Hox* gene to appear and unpublished work has shown that culturing embryos in the presence of RA causes a broad pattern of expression (R. Krumlauf, personal communication). In addition, RA appears to be the sole input to *Hoxa1*, as opposed to *Hoxb1* which also has a retinoic acid response element, but is also cross regulated by *Hoxa1* and auto-regulated.

There are a variety of methods for introducing RA into a biological system. These include oral administration (Pasqualetti et al., 2001), bathing an embryo in a culture medium containing RA (Godsave et al., 1998), or using a bead soaked in RA (Eichele et al., 1984). Using a bead is particularly attractive as it provides an effective way to deliver a local release. But the most important aspect of the bead is the local

delivery helps create an artificial gradient that can be used to test the connection between *Hoxa1* and RA, and, in particular, whether the implementation chosen for the transcription of *Hoxa1* is supported.

## Embryos

Instead of using eggs from the local supplier, fertile pathogen free chicken eggs were acquired from Charles River Laboratories. The change in egg supplier occurred because eggs from the local supplier were unreliable: many were unfertilized, and the development was inconsistent. Before the change, a great deal of time was spent dealing with eggs that were substandard. On a typical day only 2 dozen of 5 dozen eggs pulled from the incubator would be usable. The River Laboratories eggs were significantly more expensive (~$20 a dozen vs. $3.50 a dozen for AA Laboratory eggs), but they were consistently reliable, both in fertility and development time. This was yet another object lesson on the difficulty of laboratory work.

The eggs were incubated at 38°C until the proper stage of development, usually between 30 and 40 hours. The eggs were rinsed with 75% alcohol and 3 ml of albumin was removed. The eggs were windowed and a solution of .1% food coloring (equal amounts of FD&C Red #40 and FD&C Blue #1 from Spectra Colors Corp.) in HBSS was injected beneath the blastoderm to provide contrast.

## Bead Preparation and Implantation

AG1-X2 ion-exchange resin beads (mesh size 200-400, for an effective size between 50 and 150 μm) were purchased in chloride form from BioRad. They were

rederivatized to formate form by inserting them into a column and rinsing with three bed volumes of1M formic acid. They were then rinsed with water until the wash was approximately pH 5. All-trans RA was purchased from Sigma corporation and a $10^{-2}$ M solution of RA in DMSO was made fresh each day. This was subsequently diluted to the working concentration of $10^{-3}$ M. It was learned through the course of these experiments that RA degrades very quickly, even when stored under argon in a -20°C freezer. The formate beads were soaked in a 10 μ L drop of RA solution for 20-40 minutes, then rinsed in a 10 μ L drop of tissue culture media 3 times for 5 minutes each. This final step helps remove the DMSO from the beads, and the red dye in the tissue culture media stains the beads, which in turn helps make placement easier. The beads were then implanted into the hindbrain or midbrain of an embryo using the technique described in the caption of Figure 4.1 below.

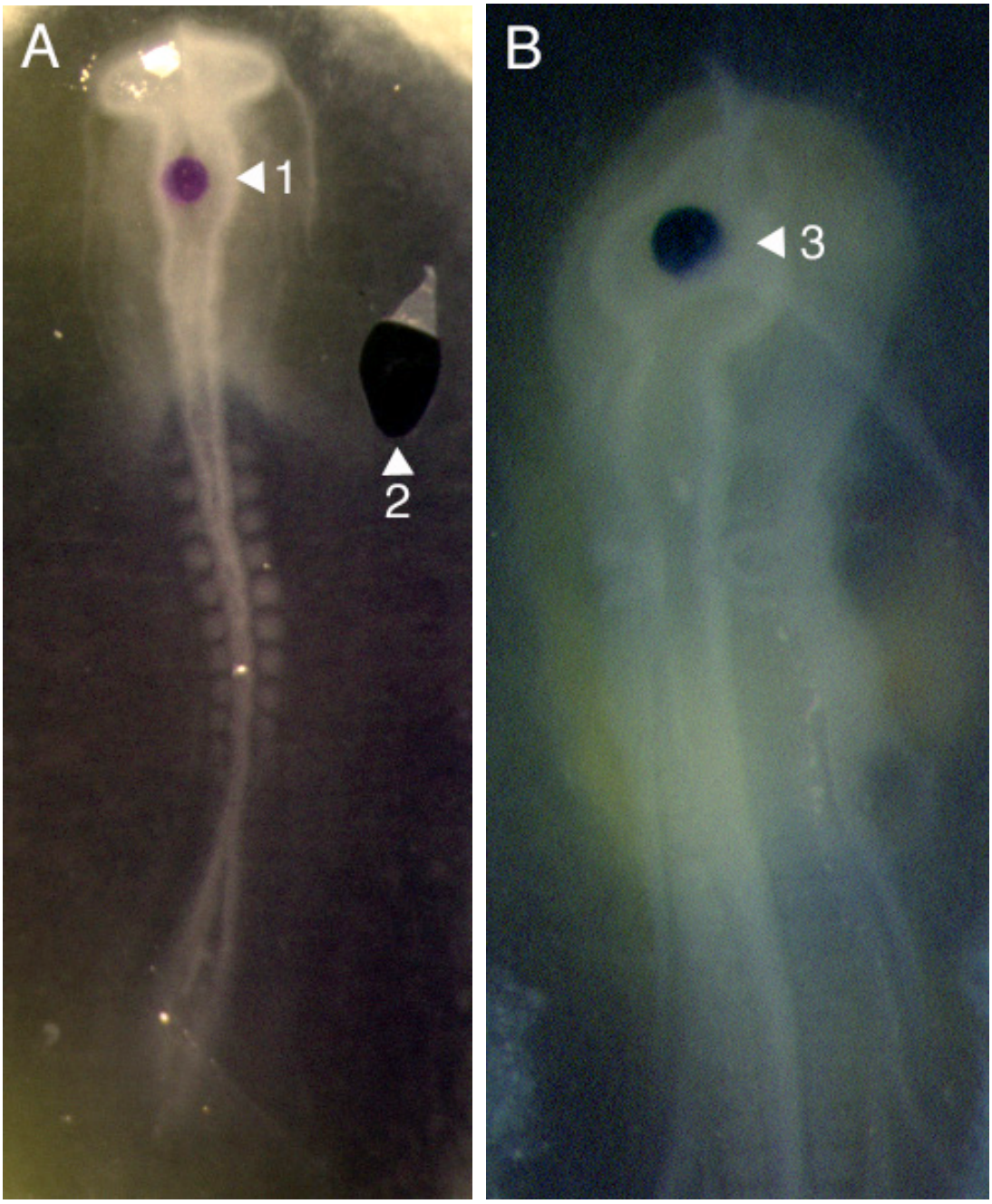


**Figure 4.1 Bead implantation. (A)** A 4x view of a stage 9 embryo with a bead (marked by 1) implanted into the midbrain of the embryo. To place the bead, a hole was torn in the vitelline membrane (marked by 2) using an electrolytically sharpened tungsten needle. The needle is then used to incise a small section of

the (potentially) closed neural tube at the mid and hindbrain level (see Figures 3.1.7 through 3.1.9). The bead is plucked from a dish with a pair of #5 forceps and placed into the hole then pushed under the vitelline membrane. After that, the bead is pushed from above the vitelline membrane into the neural tube and maneuvered into the desired position. Due to the surface tensions of the fluid, it is not possible to actually place the bead into the right position and expect it to stay there, especially if the vitelline membrane is completely removed. The white speck just anterior to the bead is a piece of eggshell that fell into the work area. **(B)** This 5x picture of a different embryo was taken 8 hours after bead implantation. The embryo is now at HH stage 12 and is starting to turn, but the bead (marked by 3) is still clearly visible in the midbrain.

After the bead implantation, the eggs were returned to the incubator for 6-8 hours. The embryos were then harvested and fixed in 4% paraformaldehyde solution either overnight at 4°C or for 1 hour at room temperature. After the paraformaldehyde treatment and a rinse in phosphate buffer saline (PBS), the embryos were dehydrated through a series of methanol/PBS washes, and were placed in a –20° C freezer for storage. Embryos stored in this manner can be kept in a freezer for upwards of a year, but in this case they were not in the freezer for more than a couple weeks. After storage, the embryos were re-hydrated with through a series methanol/PBS washes and subjected to *in situ* hybridization.

*In situ* hybridization is a molecular biology technique that allows the identification and localization of a particular nucleic acid sequence, in this case a specific

strand of messenger RNA. Recall that the Central Dogma of Molecular Biology states that mRNA is the ribonucleic acid transcribed from DNA and is the template from which a protein is translated. One method of detecting the mRNA for a particular protein in the organism is to create a probe: a complementary mRNA strand with specially modified nucleic acids. If the mRNA of interest is present in an organism, the probe will stick to it. The excess probe is then washed away, and an antibody to the modified nucleic acids is added to the mix. Finally, a dye that reacts to the antibody is added and the result is a visual readout on the location of the mRNA of interest.

Despite the brevity of the description, this process takes 4 days to complete, and so only one experiment can be performed a week. The complete protocol used can be found in Appendix C, and is a modified version of one described in the literature (Wilkinson, 1992).

The probe used for the assay was *Hoxa1*, and typical results are shown in Figure 4.2. The most striking feature of the expression pattern in Figure 4.2B is that there appears to be a gradient of expression in section of the neural folds marked by the arrows. This is, in fact, a real measurable gradient as seen in Figure 4.3.

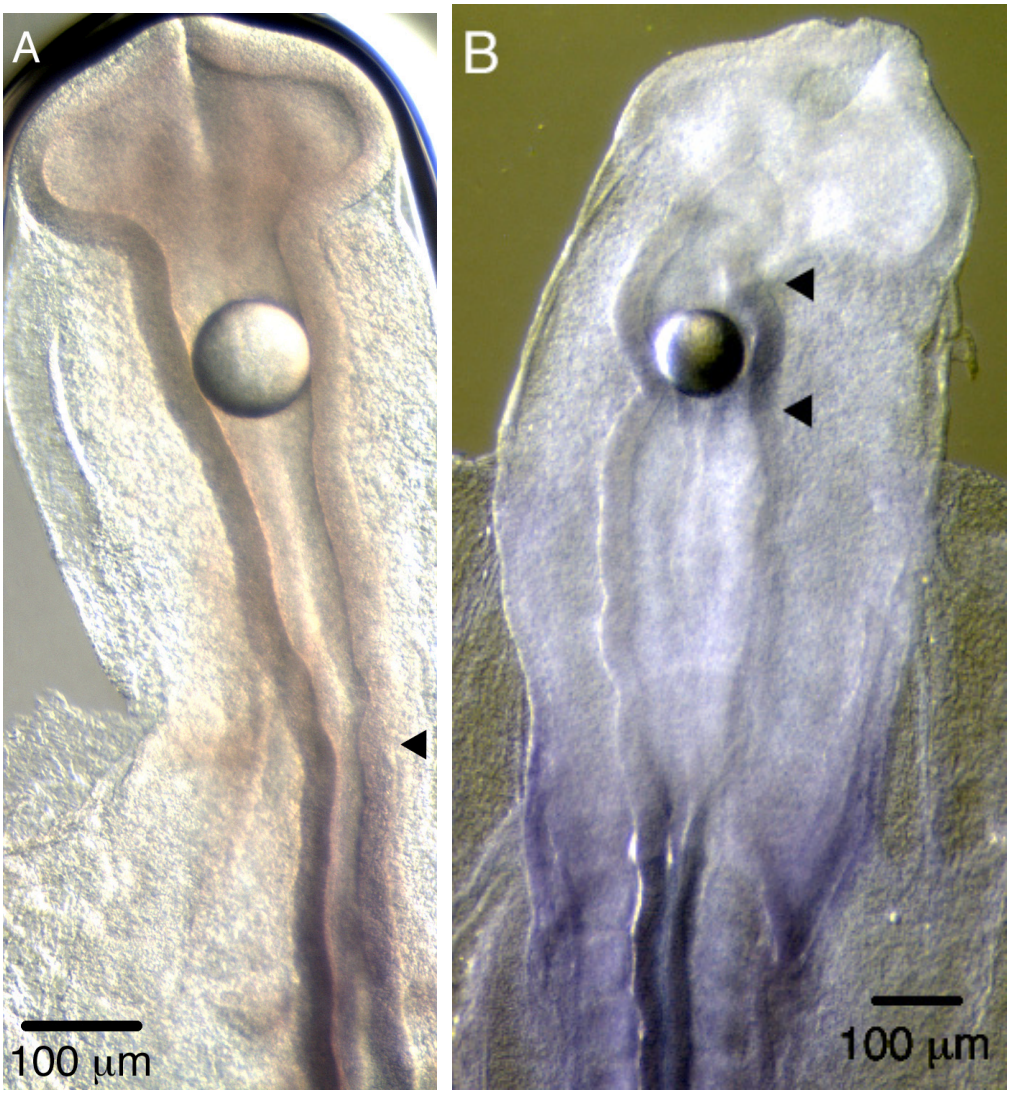


**Figure 4.2 Hoxa1 expression patterns. (A)** 6.3x picture of a stage 11 embryo stained for *Hoxa1*. The purple/blue stain marks the localization of the gene, and the deeper the color, the stronger the expression. A control bead soaked in only DMSO was implanted into the midbrain of a stage 9 embryo and collected at stage 11. This picture is a bit unique in that the bead remained in place through the entire *in situ* protocol. This is not often the case, as the bead usually becomes dislodged during one of the many washes. *Hoxa1* is clearly expressed (as

evidenced by the purple color) in the neural tube posterior to the point marked by the arrow. As expected, there is no *Hoxa1* expression near the bead **(B)** 5x picture of a stage 11 embryo stained for *Hoxa1*. An RA coated bead was implanted into the midbrain at stage 9. Notice the strong purple expression of *Hoxa1* in the area between the black arrows. This picture is typical of the results, but is particularly nice in that the bead stayed in place and the expression of *Hoxa1* near the bead is so prominent. If the bead is implanted at ages older than stage 10, there is a reduced chance that there will be any change in the expression of *Hoxa1*. This is compared to earlier stages when the hind and midbrain are still able to respond to the RA, and is consistent with other reports of RA perturbation experiments (Dupe and Lumsden, 2001; Gale et al., 1996).

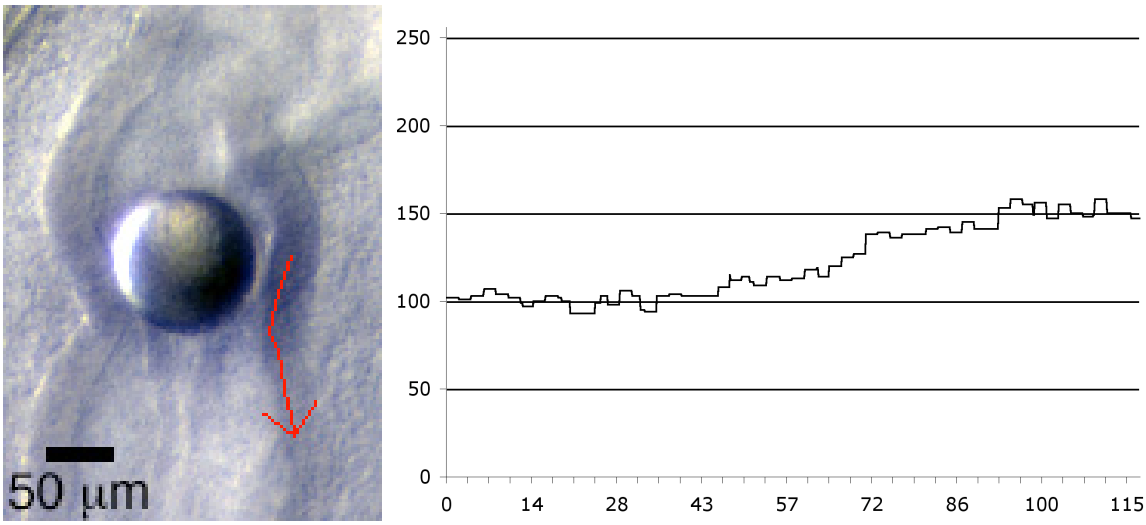


**Figure 4.3 *Hoxa1* expression near the RA bead.** This closeup of Figure 4.2 focuses on the expression of *Hoxa1* near the RA coated bead. There is an area of

expression just posterior to the bead that extends nearly 50 microns from the bead, and the expression in the neural tube is evident, especially on the right side. Using the 510LSM image analysis software from Zeiss, the change in intensity was measured along the red trajectory, and the results are show in the chart above. The ordinate is pixel intensity, and the abscissa is the microns along the path. Recall that lower intensities correspond to darker colors. Along the120 micron path the intensity pixels changes about 20%, with the first 40 microns holding relatively steady, followed by a gradual change starting before the bend in the red arrow. After a gradual change along the next 40 microns of the path, the intensity values level off to background intensity.

Determining the number of RA molecules on the bead can only be done in an indirect way. A study showed using radioactive RA that after about 30 minutes, approximately 25% of the radioactivity in the solution had been depleted (Eichele et al., 1984). Therefore, assuming a concentration of $10^{-3}$ M for the solution a theoretical maximum uptake by the bead is approximately $2.4 \times 10^{12}$ molecules. As for the depletion, approximately 40% of the RA is released by the bead in the next 8 hours (Eichele et al., 1984; Langer and Peppas, 1981). This means that the bead is, in effect, a saturating source with over $9.6 \times 10^{11}$ molecules released from the bead into the surrounding tissue during the 8 hours it is in the embryo.

## Bead Model

Modeling the effects of the RA soaked bead proceeded concurrently with the laboratory work. The simulation was modified to provide a constant saturating source of RA diffusing laterally into the tissue. At each time step anywhere from 20 to 2000 molecules of RA were introduced into each cell. This effectively provides a saturating source, because each of the cells contains approximately 2000 free receptors for the RA. The source is not symmetric, as it appears from the position of the bead that it is able to provide more RA to the anterior cells as compared to the posterior cells. Recall that the *Hoxa1* mRNA transcription was implemented used a combination of a Hill function and first order reaction. The accumulation of transcription factors (in this case the bound RAR/RXR dimers), would lead to the activation of the *Hoxa1* gene, and once this occurred the gene was activated and mRNA could be transcribed. But with the large accumulation of bound dimers provided by the constant source of RA, there was little chance that the gene would become unactivated. If a bound dimer dissociated from the gene, another was present to take its place. This implementation does not allow for a differential in *Hoxa1* expression due to varying amounts of RA. This results in an indiscriminate up-regulation of *Hoxa1* as shown in Figure 4.4A below.

Because the results did not accurately capture the new data, the model required a change to incorporate the data gathered from the embryo. This is in accordance with the quote from Hamburger at the beginning of the chapter. Therefore, the model was changed so that the *Hoxa1* mRNA was transcribed using a proportionality function (*i.e.*, the probability of transcription of *Hoxa1* mRNA proportional to the number of bound RAR/RXR dimers present) instead of the sequential Hill equation to activate the gene,

and a first-order reaction transcription from the activated gene. The other change to the model was the deletion of the parameters for the activation/unactivation of the *Hoxa1* gene. After making these changes, the model was able to capture the results that were gathered in the laboratory, as seen in Figure 4.4B.

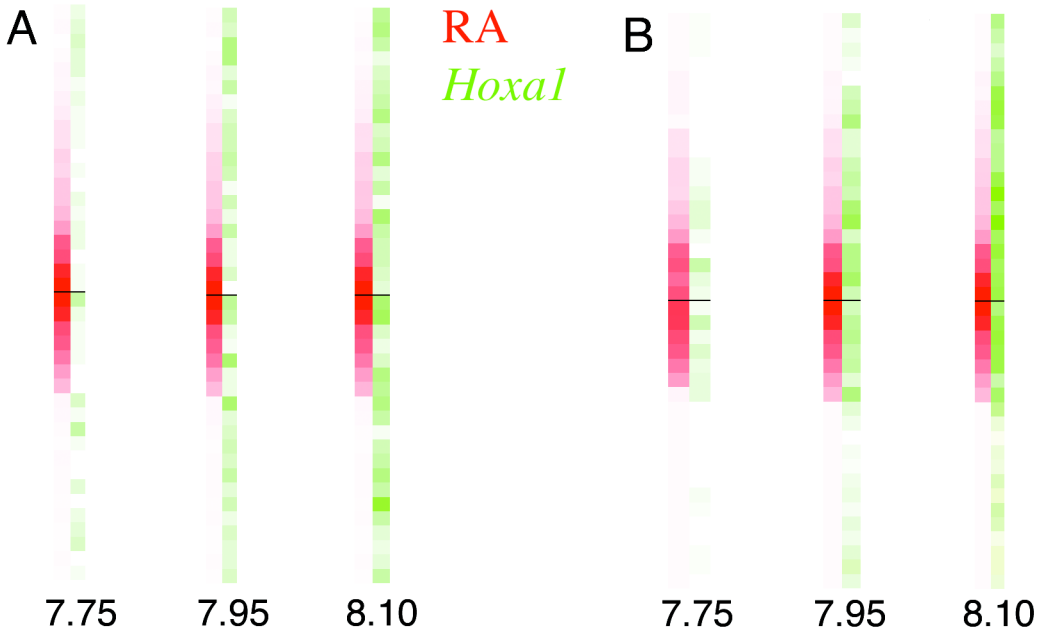


**Figure 4.4 *Hoxa1* expression from a constant RA source. (A)** Notice that there is no visible change in the levels of mRNA for *Hoxa1* due to the differing levels of RA. Increasing the number of free receptors by an order of magnitude does not affect the qualitative results. Because the bead was kept in the embryo for only 6-8 hours, the model was stopped after 8.1 dpc. **(B)** After making a change that ties the transcription of *Hoxa1* to the number of transcription factors present, the model now captures the type of behavior seen in the lab, namely more RA leads, in general, to a stronger expression of *Hoxa1* mRNA. There is still moderate

expression in the anterior (top) end of the figure however, maybe more than one would hope. This is not terribly surprising however, given that 200 molecules of RA are introduced at each time step and they have a cumulative effect. But in the posterior (bottom) section of the figure, in which there are 10 times fewer RA molecules introduced at each time step than in the anterior end, the expression is lower in general. Most importantly, the strongest expression of *Hoxa1* mRNA in Figure 4.4B is nearest the largest collection of RA, *i.e.,* the center of the figure. This is not true in Figure 4.4A: More RA does not in general lead to a stronger expression of *Hoxa1* mRNA.

The changes to the function for the transcription of *Hoxa1* were made to the baseline model, and the wild-type scenario was run again. The results of the simulation are shown in Figure 4.5 below.

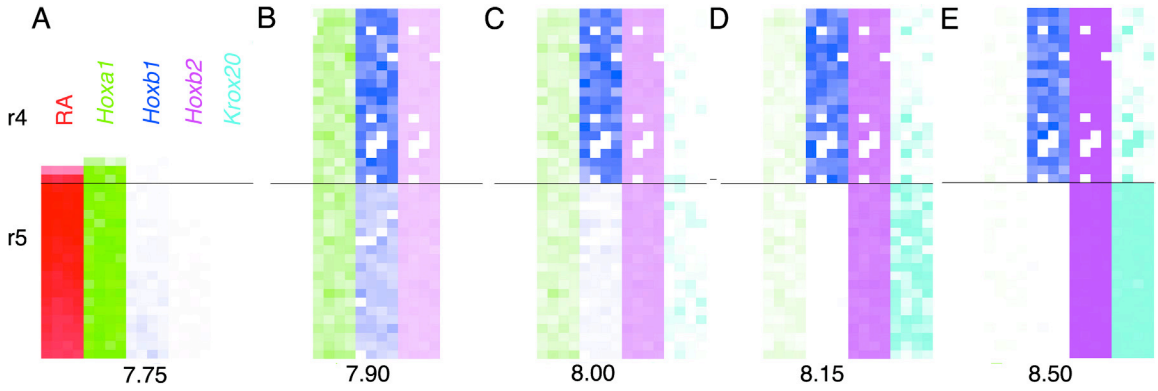

**Figure 4.5 Wild type.** This time slices in this picture are exactly the same as in Figure 3.7. Notice that the results are qualitatively the same. The only changes to the parameters were the deletion of the cell division, and a change in the transcription rate of *Hoxa1*. It might be tempting to make a comment about the number of blank cells in the second column of the *Hoxb1* and *Hoxb2*, but any conclusions would be erroneous; the only difference between that column and the first column is the random number seed used. This results shows that the model was insensitive to this change in the implementation of *Hoxa1* transcription.

An experiment that was relevant to the model under investigation provided a test of one of the key interactions of the model. The resulting data led to a change in the implementation of the RA and *Hoxa1* connection. The gradient of expression resulting from an RA coated bead has not been reported in the literature, and this novel result continues to support the view that RA concentration plays a role in the patterning of the hindbrain.

## *References for Chapter 4*

Dupe, V., and Lumsden, A. (2001). Hindbrain patterning involves graded responses to retinoic acid signalling. *Development* **128,** 2199-208.

Eichele, G., Tickle, C., and Alberts, B. (1984). Microcontgrolled Release of Biologically Active Compounds in Chick Embryos: Beads of 200 micron Diameter for the Local Release of Retinoids. *Analytical Biochemistry* **142,** 542-555.

Gale, E., Prince, V., Lumsden, A., Clarke, J., Holder, N., and Maden, M. (1996). Late effects of retinoic acid on neural crest and aspects of rhombomere. *Development* **122,** 783-93.

Godsave, S. F., Koster, C. H., Getahun, A., Mathu, M., Hooiveld, M., van der Wees, J., Hendriks, J., and Durston, A. J. (1998). Graded retinoid responses in the developing hindbrain. *Dev*. *Dyn*. **213,** 39-49.

Langer, R. S., and Peppas, N. A. (1981). Present and future applications of biomaterials in controlled drug delivery systems. *Biomaterials* **2,** 201-214.

Pasqualetti, M., Neun, R., Davenne, M., and Rijli, F. (2001). Retinoic acid rescues inner ear defects in *Hoxa1* deficient mice. *Nature Genetics* **29,** 34-39.

Wilkinson, D. G. (1992). Whole mount *in situ* hybridization of vertebrate embryos. *In* "In situ hybridization: A Practical Approach" (D. G. Wilkinson, Ed.), pp. 75-83. IRL Press, Oxford.

# Chapter 5: Summary

> Simple interactions can have consequences that are not predictable by intuition based on biological experience alone.
>
> - Lee Segel, 1980

The stochastic simulation algorithm model captures the timing of several *Hox* gene expression patterns in wild-type animals, and *in silico* simulations performed as a check of key interactions produced results similar to *in vivo* experiments. During the course of building the model, the *in silico* investigations suggested that an experiment concerning the connection of retinoic acid and *Hoxa1* would be enlightening. A new experiment was designed to investigate the interaction of these elements *in vivo*, and the corresponding experiment was performed in the model. The resulting data suggested that an implementation decision was incorrect. Based on these results the model was modified to encompass the new data, without losing the fit to the original data set.

In addition, the *in silico* experiments yield intriguing predictions that have yet to be thoroughly examined biologically. For example, the mutation experiments in which 5' RARE is mutated predicts that *Krox20* expression is down-regulated in rhombomere 5 (Figure 3.9C). The simulation also suggests that when *Hoxb1* is mutated, there is an up-regulation of *Krox20* in rhombomere 4, and a down-regulation of *Hoxb2* and *Krox20* in r5 (Figure 3.8C). The formal nature of the model calls attention to these simple test

experiments, and checking predictions will lead to valuable insight into the regulatory network.

If the model predictions are correct, the tool will allow a deeper investigation into the nature of the components and allow researchers to ask more complicated questions about the nature of the interactions. On the other hand, if the model predictions turn out to be incorrect (as was the case in Chapter 4), the experimental data leads to a refinement of the model that incorporates the new results. The revision will then offer different predicted relationships that will stimulate further experiments. This investigation will ultimately lead to a better predictive tool for the next round of experiments. Indeed, this is one of the great strengths of the simulation: as the components of the model are given greater support, it can be used to perform *in silico* experiments to identify the *in vivo* experiments that will be the most enlightening.

In addition to serving as an organizational tool for presenting newly established interactions, the model can also be used to investigate hypothesized molecular interactions. This was the case for the *Krox20*/*Hoxb1* connection that was the basis for the experiment described in Appendix A. Using it for this purpose will allow researchers to explore the consequences on the network as molecular connections are added or removed. The simulation itself is designed in a way to make modifications easily, and adding new pieces is a modular process. This will inevitably need to occur as new data are presented which require updating the regulatory network (Figure 3.5) accordingly. An example of this is work currently in progress that seems to suggest *Krox20* contains an auto-regulatory element (P. Charnay, personal communication).

It should also be possible to extend this model in ways that are not only spatial and temporal, but which incorporate more of the known biochemistry of the system. For example, extending the model to include the next segment anteriorly, rhombomere 3, would allow an investigation into the early r3 expression of *Krox20* (Schneider-Maunoury et al., 1993). On the temporal front, it would be instructive to include the proper mechanisms to capture later events such as the progressive down-regulation of *Hoxb2* in r3 by 10.5 dpc (Maconochie et al., 1997).

Biochemical improvements could include adding more genes, implementation of the mRNA modification and transport steps, and a better characterization of the genes or cofactors. Adding *Hoxa2* is an obvious choice because of the connection to the genes already in the network: it has been shown that *Krox20* is directly involved in the transcriptional activation of *Hoxa2 (Schneider-Maunoury et al., 1997).* New information concerning these genes appears on a regular basis and that provides the information for a better characterization. For instance, it has recently been observed an early low level of *Hoxb2* expression in rhombomere 5 appears to be due to a retinoic acid response element on the *Hoxb1* 3' RARE (R. Krumlauf, personal communication). All of these improvements will allow for a better understanding of the interaction and timing of the events.

There is also reason to believe that the model also can play an important role in explaining differences between species; for example *Hoxb2* expression in r3 and r5 is much lower in chick than in mouse (Vesque et al., 1996). The differences may be due to regulatory sequences that have yet to be fully characterized, and which can be easily updated in the model once they are known. It has also been suggested that this may be

influenced by different basal transcription rates between the species (R. Krumlauf, personal communication). Once the mechanisms for *Hoxb2* regulation are in place, it would be possible to use the model to explore this issue. An investigation addressing this would include changing the basal transcription rates, the binding affinity parameters, and experimenting with different transcription factors configurations.

## Conclusion

This thesis has shown that a tight coupling of modeling and experimental work provides a valuable framework for investigating biological problems; a framework that will become even more valuable as the amount of data increases. The act of constructing the model identified interesting biology questions, and the answer to one of those questions was used to enhance the model. Once the model was complete, the *in silico* experiments continued to identify potentially interesting biological questions.

The investigation into the early *Hox* genes also shows the success of using a stochastic simulation algorithm to model a gene regulatory network. This is especially important in situations where the fluctuations in the system appear to be a factor, because the stochastic approach is able to incorporate them in a physically intuitive and meaningful way. This investigation has also demonstrated that the SSA methodology has a wider applicability than the previous intracellular investigations. It can be adapted to encompass intercellular interactions, and the use of a priority queue to time order the multi-cellular system is an important addition to the method. The laboratory work stimulated by the model has yielded important biological results. The repression experiment in Appendix A shows that, as it stands, the construct does not successfully

repress *Hoxb1*. The RA perturbation experiment in Chapter 4 suggests that the response of *Hoxa1* to RA is concentration dependant.

It is expected that continued efforts in refining and using these sorts of models will result in a greater understanding of how computer simulations can be used to produce new biological insights. It is hoped that the success of this model will encourage more biologists to investigate the benefits of computer modeling in general, and stochastic simulation in particular. There is evidence that this work is already being noticed in the biology community: the author recently discovered that an article destined for the journal *Developmental Biology* referenced this work.

In a lesson for the mathematicians, this work also demonstrates a common problem with working in biology, one that was addressed in the general comments about modeling in the first chapter. There are too many "right" models, and the available laboratory data does not always allow for the ability to distinguish between them. This was the case with the first incarnation of the model: using a Hill function to produce an activated form of *Hoxa1* was reasonable choice given the information in the literature. Also supporting this choice were the results of the model: the simulation reproduced the wild type expression pattern, and computer perturbations yielded results similar to their laboratory counterparts. When new data were generated that tested this component, it was shown that the original implementation was not correct, and the model was changed to capture the dependence of *Hoxa1* transcription the quantity of transcription factors in a more explicit way. The new model is therefore better in so far as it captures more of the laboratory data. However, as is seen in the similarity between Figures 3.6 and 4.5, the models cannot be distinguished from each other on the basis of the output alone. This

shows the importance of the laboratory work in generating data that clarifies aspects of the model.

Finally, the systems biologists should see this work as a successful example of what they have been preaching: an integrative approach to biology problems will provide insight into how the systems behave. Insight that is not possible from approaching the problem using modeling or laboratory experiments alone. As more such successful interconnected effort appear, it is hoped that both biologists and mathematicians will look beyond the difficulties of interdisciplinary work that is mentioned in the quote from David Botstein at the beginning of Chapter 3, and instead focus on its enormous benefits to both fields.

## *References for Chapter 5*

Maconochie, M. K., Nonchev, S., Studer, M., Chan, S. K., Popperl, H., Sham, M. H., Mann, R. S., and Krumlauf, R. (1997). Cross-regulation in the mouse HoxB complex: the expression of Hoxb2 in rhombomere 4 is regulated by Hoxb1. *Genes Dev*. **11,** 1885-95.

Schneider-Maunoury, S., Seitanidou, T., Charnay, P., and Lumsden, A. (1997). Segmental and neuronal architecture of the hindbrain of Krox-20 mouse mutants. *Development* **124,** 1215-26.

Schneider-Maunoury, S., Topilko, P., Seitandou, T., Levi, G., Cohen-Tannoudji, M., Pournin, S., Babinet, C., and Charnay, P. (1993). Disruption of Krox-20 results in

alteration of rhombomeres 3 and 5 in the developing hindbrain. *Cell* **75,** 1199-214.

Vesque, C., Maconochie, M., Nonchev, S., Ariza-McNaughton, L., Kuroiwa, A., Charnay, P., and Krumlauf, R. (1996). Hoxb-2 transcriptional activation in rhombomeres 3 and 5 requires an evolutionarily conserved cis-acting element in addition to the Krox-20 binding site [published erratum appears in EMBO J 1996 Dec 16;15(24):7188]. *Embo. J.* **15,** 5383-96.

# Appendix A: *Hoxb1* perturbation

Perturbation experiments have been well worked out in some systems (most notably yeast and *Drosophila*), but they are harder to do in higher organisms. This is not to say that they are impossible, and the following section describes an experiment that was designed to investigate a component of the *Hox* network.

As mentioned previously, a group has presented a model that asserts that *Hoxa1* and *Hoxb1* are involved with the repression of *Krox20* (Barrow et al., 2000). This supposition was implemented in the baseline model presented in Chapter 3. However, there are reasons to believe that their model might not be accurate. In the $Hoxb1^{null}$ mutant there are no changes in the level of *Krox20*, and in the $Hoxa1^{null}/Hoxb1^{null}$ double mutant embryos there is no sign of *Krox20* in rhombomere 3 and reduced expression rhombomere 5. In addition, the $Hoxa1^{null}$ mutant mouse shows reduced levels of Krox20 in rhombomere 5 (Gavalas et al., 1998; Studer et al., 1998). Part of the difficultly in interpreting these results is that the rhombomeres in these mutants are often altered. For instance, in the $Hoxa1^{null}/Hoxb13'RARE^{null}$ mutant, a territory with new characteristics forms in the place of rhombomere 4 (Gavalas et al., 2001). But the rhombomere alteration does not always occur; rhombomere 3 appears to be normal (using both visual and *in situ* assays) in the $Hoxa1^{null}/Hoxb1^{null}$ double mutant (Studer et al., 1998). Taken together, the evidence does not seem to support the model of *Hoxb1* and *Hoxa1* playing a role in repressing *Krox20*. In an effort to test this conjecture, an experiment was designed to perturb the system using a specially designed piece of DNA. This

experiment would directly address the predicted changes in *Krox20* expression due to the knockout of *Hoxb1*

A detailed study of the *Drosophila* protein Engrailed showed that it was able to repress transcription activity (Han and Manley, 1993), and it has been fused to other proteins to provide a dominant negative like activity in a system. However, this fusion has been done primarily in *Xenopus* and fish (*cf.* LaBonne and Bronner-Fraser, 2000; Vignali et al., 2000).

Starting with the CS2+ vector (R. Rupp and D. Turner), Heather Marshall from the Stowers Institute for Medical Research inserted the cDNA for *Hoxb1* into the polylinker between the BamH1 and Xho1 sites. Into this construct she inserted the Engrailed repressor in frame at the unique XmaIII site of *Hoxb1*. The modified protein with the Engrailed repressor would attach to the *Hoxb1* binding domain and would in turn repress the expression of *Hoxb1* due to the auto regulatory loop. In addition, it would repress any gene that *Hoxb1* could attach to.

Her original plan for this construct was to use the construct for mRNA fish injections. This is a relatively easy procedure for a variety of reasons, not the least of which is that one cell fish embryos are easily harvest and manipulated. But, after making this construct, Dr. Marshal did not ever use it in fish and she provided it to the author for use in a chick perturbation experiment.

Introducing this DNA in a way that it becomes active would be a fantastic test of the model. If the DNA could in fact repress *Hoxb1* expression, then assaying for *Krox20* would allow for another piece of evidence concerning the supposed *Hoxb1-Krox20* connection.

On the modeling side, the results of the *Hoxb1*/Eng construct produces results that are very similar to those of the *Hoxb1* mutant in Chapter 3. This is evident in the results shown in Fig A.1 below.

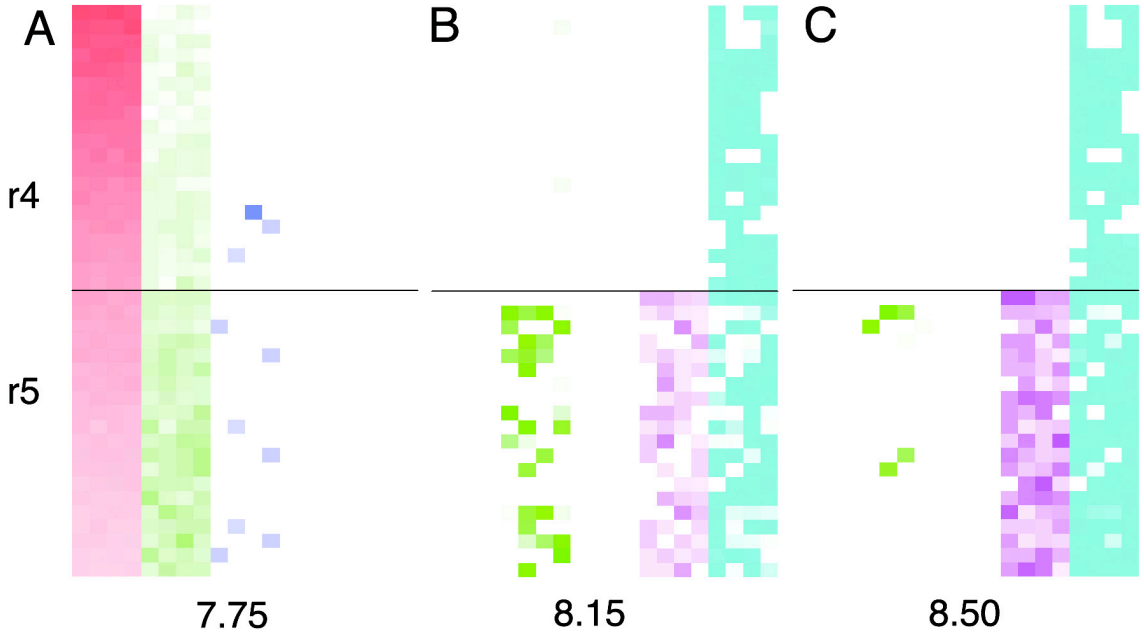


**Figure A.1 *Hoxb1*/Eng model results.** These results are very similar to the *Hoxb1* mutant presented in Chapter 3 (Figure 3.5). There is a near total down-regulation of *Hoxb1* which, combined with the normal fading of *Hoxa1*, allows for the up-regulation of *Krox20* in rhombomere 4. *Hoxb2* does appear in r5 due to the *Krox20* up-regulation, but is absent in r4 because of the lack of *Hoxb1*. These simulations were run before making the change to the mechanism for transcribing *Hoxa1* brought about by the work described in Chapter 4.

The similarities between the *Hoxb1* mutant in Chapter 3 and the *Hoxb1*/Eng construct are not unexpected at all. In both cases the dominant effect is the repression of

*Hoxb1*. However, in the *Hoxb1*/Eng construct, there is an occasional low level of the *Hoxb1* product still as not all of the system would be bound. Therefore, the lack of *Hoxb1* and *Hoxa1* would result in an expansion of *Krox20*. It is this expansion of *Krox20* that the experiment was designed to test.

## *Electroporation*

Introducing the *Hoxb1*/Eng construct cannot be accomplished in the same manner as a 1-cell fish injection, but a different technique that accomplishes the same result, namely having the foreign DNA incorporated into the organism, can be undertaken. Electroporation is a technique for introducing foreign DNA into cells. The method involves breaking down the membrane of cell walls through the use of an electric pulse. In addition to creating holes in the membrane, the electrical gradient drives the negatively charged DNA into the holes in the membrane. There are a variety of variables that contribute to the effectiveness of the electroporation including the size and placement of the electrodes, but by far the most important component is the duration and voltage of the pulse. In general, 3-5 pulses of 50 microsecond duration and between 7 and 25 volts works well. Excellent technical reviews can be found in (Itasaki et al., 1999; Swartz et al., 2001). One very nice benefit of the electroporation is that because of the electrical gradient, only cells on the positive side of the neural tube have their cell walls broken down, and only one half of the embryo is exposed to the DNA. This provides an excellent internal control, as one side of the embryo is experiment, and the other side is normal.

Electroporation can be a tricky procedure, and in order to check that the electroporation worked correctly, an additional control was required. To this end, an IRES GFP construct was purchased from Clontech. The internal ribosome entry site (IRES) is a sequence of DNA that a ribosome recognizes and will attach to. This leads to the creation of green fluorescent protein (GFP) mRNA, and when it is translated into proteins, a cell that has incorporated this DNA will glow when excited with the proper wavelength of light.

To create this construct, the Clontech IRES-GFP module was removed using the restriction enzymes XhoI and XbaI then cloned into the *Hoxb1* Engrailed construct just after the stop codon for the *Hoxb1*. The result is a bi-cistronic message: one that has two gene products (in this case the *Hoxb1*/Eng repressor and the GFP) from adjacent stretches of the same mRNA. In general the message from the second coding region will be weaker than the first. If the electroporation is successful, the GFP will be glowing and that also signifies that the *Hoxb1*/Eng fusion protein has been created. It is important to remember that it does not provide any information about if the *Hoxb1*/Eng repressor message is working correctly.

## Embryos

Fertile pathogen free chicken eggs were acquired from Charles River Laboratories and incubated at 38°C until the proper stage of development, usually between 28-34 hours. The eggs were rinsed with 75% alcohol and 3 mL of albumin was removed. The eggs were windowed and a solution of .1% food coloring (equal amounts of FD&C Red

40 and FD&C Blue 1 from Spectra Colors Corp.) in Hanks' Balanced Salt Solution (HBSS) was injected beneath the blastoderm to provide contrast.

After windowing the eggshell and injecting the dye, a small amount of HBSS was added to the top of the embryos to keep it moist. A tungsten needle was then used to cut a hole in the vitelline membrane near the hindbrain of the embryo. A solution of ~1ug/ul of the Hoxb1/Eng/IRES GFP construct (with a small amount of Fast Green dye included to make the injection easier to see) was injected using a quartz micropipette into the lumen of the neural tube. Electrodes made from platinum wire were laid flat on the area opaca, parallel and lateral to the embryo. About 1-2 mm of contact was made with the area opaca, and the electrodes were approximately 5 mm apart. 3-5 current pulses of 20-40 V and 50-100 ms duration were applied. More HBSS or Ringer was added to the top of the embryo, the egg was resealed with packing take and placed back into the incubator for 8 hours. The embryos were then screened for fluorescence using a Leica microscope. Positive embryos (Figure A.2) were harvested and fixed in 4% paraformaldehyde (PFA) solution overnight at 4°C. The next day they were dehydrated through a series of methanol washes, and were placed in a –20°C freezer for storage. Embryos stored in this manner can be kept in a freezer for upwards of a year, but in this case they were not in the freezer for more than a couple months. After storage, the embryos were re-hydrated into phosphate buffer saline (PBS) and whole mount *in situ* hybridization was performed.

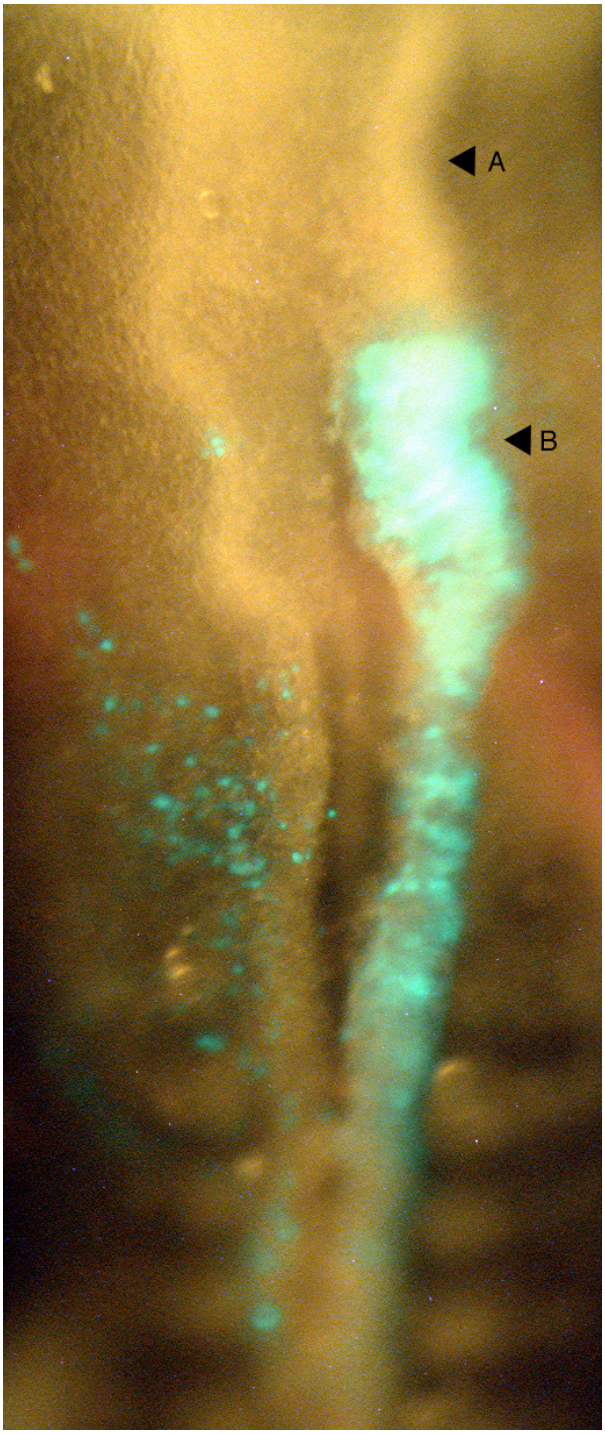


**Figure A.2 Glowing hindbrain.** This picture shows a 6.3x magnification of the hindbrain of a stage 11 embryo after electroporation at stage 7. This picture was taken *in ovo* during the screening process for embryos in which the *Hoxb1*/Eng/GFP construct was successfully incorporated into the cells during

electroporation. The midbrain/hindbrain boundary is marked by **(A)**, and the boundary between the third and fourth rhombomeres is marked by **(B)**. Despite the name of GFP, the cells in this instance were colored cyan in Photoshop to provide contrast to the anatomy of the embryo.

This experiment was performed using the embryos that fluoresced strongly (as in Figure A.2) and after performing the *in situ* for *Hoxb1*, the gene that would be affected if the construct were working correctly, there was no visible difference between the control and the experimental embryos.

With the negative results from this experiment, it became clear that something wasn't working right, but it could be a variety of things. First of all, the construct may not be working at all, or it may not have been introduced into the embryo at an early enough time point. If this occurred, the *Hoxb1*/Eng construct would not be able to shut down the system that was already adequately initiated. The later problem was the logical one to test, and it was initially presumed to be easier. It turns out that it was anything but.

In an experiment initially performed by Kristen Correia at the Stowers Institute, the original construct with the CMV enhancer was electroporated into chicks at HH stage 4 (Hamburger and Hamilton, 1951). This experiment is very different than the one described earlier as there is no neural tube to hold the DNA solution. The entire procedure is described below.

After performing a dozen of these electroporations, it was reported that there was only very low level of fluorescence. Further investigation into this led to the conclusion that the CMV enhancer doesn't work very well at these early stages of development, and

the CS enhancer should be used instead. CS is a combination of the CMV enhancer and the Chick Beta actin promoter.

The backbone of his LZRS-CA-H2B-YFP construct (originally used for the creation of a retrovirus for infection of quail) was the basis for the new Hoxb1/Eng repressor (courtesy R. Lansford). The cloning was done in two stages. The H2B-YFP module was removed using the NotI and XhoI enzymes, and the Clontech IRES GFP module was inserted into the LZRS-CA backbone after the band was isolated using a double digestion with XhoI and Not1. At this point the Hoxb1/Eng module was removed from the CS2+ construct using BamH1 and XhoI, commercially available Xho linkers (NE Biolabs) were added, and the resulting DNA fragment was cloned into the Xho site of the LZRS-CA-IRES-GFP. In yet another example of the difficulty in laboratory work, the work described in this paragraph took over six weeks to perform.

The construct (hereafter called CS-*Hoxb1*/Eng) was tested by transfection into two different cell lines and the results are shown in Figure A.3.

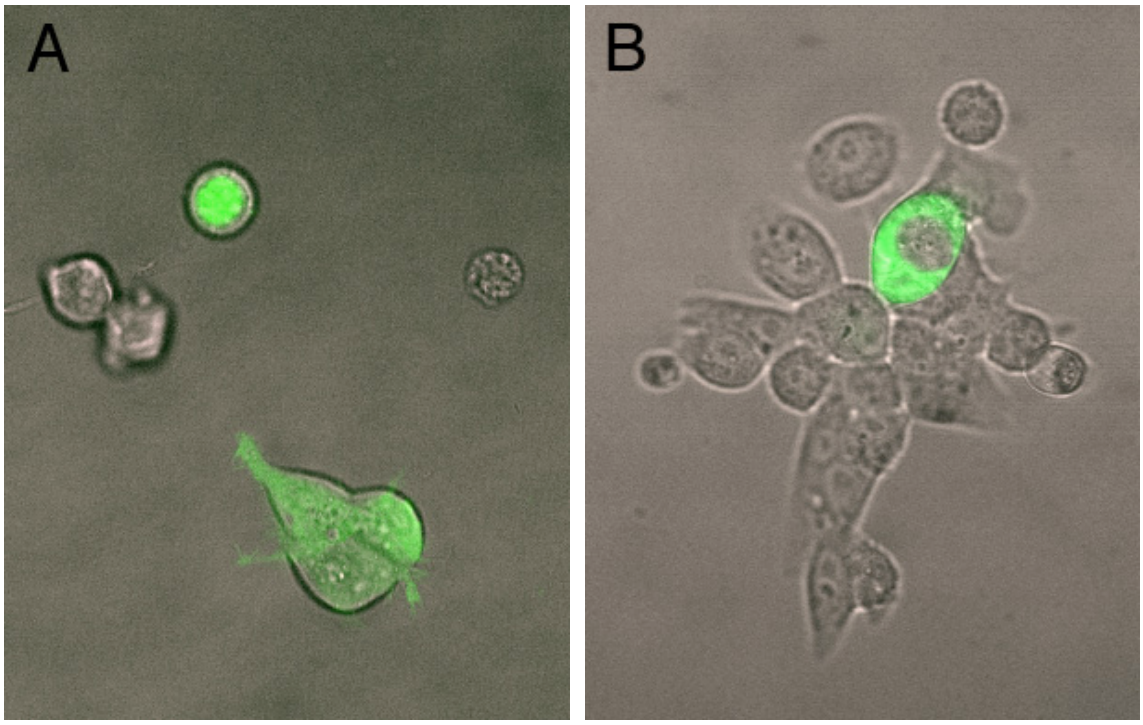


**Figure A.3 Transfected cells.** 63x magnification pictures of **(A)** Chinese hamster ovary and **(B)** 293 GPG cells transfected with the CA-*Hoxb1*/Eng construct. The cells were transfected using a 1 μ gram/μ liter DNA solution combined with Superfect (Qiagen). Superfect is a specially designed dendrimer that forms a complex with the DNA of interest and enters the cell using negatively charged receptors (Qiagen, 2000; Tang et al., 1996). Both these images were acquired on a Zeiss 410 inverted microscope.

The successful glow to the cells was promising, and the author visited the Stowers Institute for medical research to learn how to electroporate chick embryos at a young age. As mentioned previously, the procedure for this type of electroporation is much more difficult since there is no neural tube to contain the DNA. Instead, the DNA must be injected between the vitelline membrane and the embryo directly over the node. If the

DNA spreads beyond the translucent border of the embryo, it was injected on top of the vitelline membrane. If the DNA is localized in a small area, the yolk was injected.

The electroporation must be done with electrodes oriented such that the positive terminal is inserted into the yolk beneath the embryo, and the negative terminal is on top of the grove containing the DNA. Figure A.4A shows the major landmarks in a stage 4 embryo which provides an orientation for this process, and Figure A.4 B is a picture of the custom electrodes that were created for the electroporation.

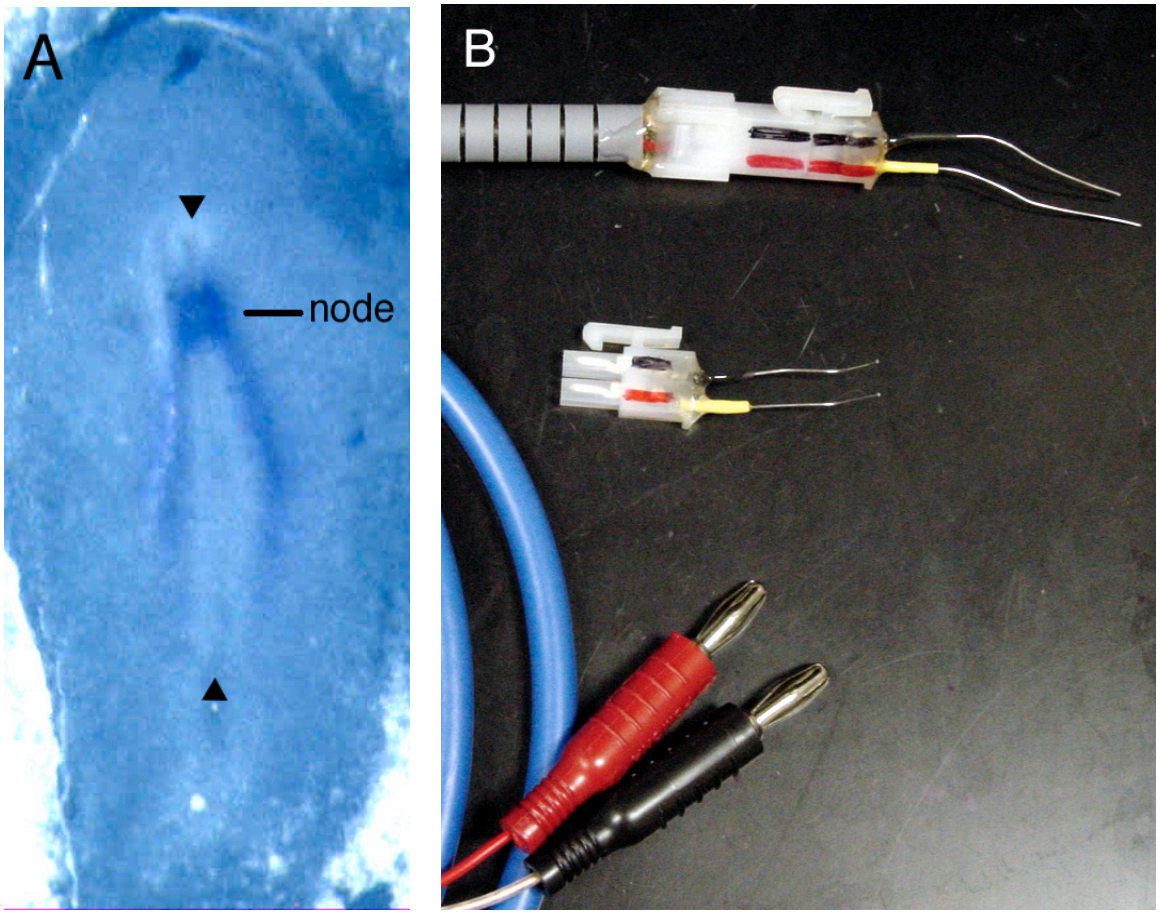

**Figure A.4 Stage 4+ embryo. (A)** This embryo, stained for the gene *Fringe* (probe courtesy of C. Tabin), highlights the important landmarks of a late stage 4 embryo. Anterior is towards the top of the page, posterior is toward the bottom. The primitive groove extends along the anterior/posterior axis between the arrows. Hensen's node (which is strongly expressing *Fringe*, as evidenced by the dark blue color) is clearly marked, and the neural folds (which eventually fold over and join together to become the neural tube) are on either side of the neural grove and are also expressing Fringe toward the anterior part of the embryo **(B)** Electrodes that were used for the electroporation experiments. These were built using 24 gauge platinum wire from a Caltech supply room, and others parts from a local electronic supply (MarVac). The electrodes are designed in such a way to make the configuration easily changeable for the proper application. The electrodes at the top are for stage 4/5 electroporation in which the electrodes need to be above and below the embryo, while the electrodes in the middle of the picture are for later stages in which the neural tube is more fully formed.

A construct containing CA driving GFP was used as a control for testing the efficacy of the technique using the equipment in the lab. An example of a control embryo is shown in Figure A.5.

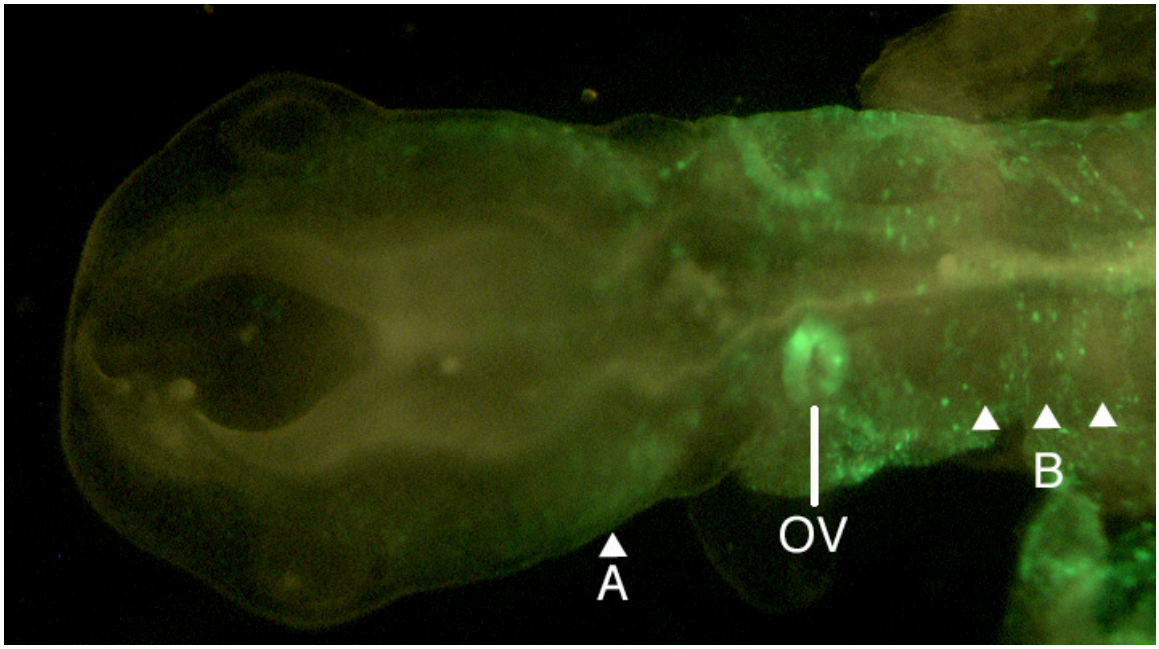


**Figure A.5 CA-GFP embryo.** This embryo was electroporated with CA driving GFP at stage 4 and collected at stage 12+. The early electroporation was done in the primitive streak and the node and the neural cells are strongly labeled. This picture is a 6.3x view of taken under a long pass GFP filter that allows both the fluorescent and white light to pass. The otic vesicle (OV) is clearly highlighted, as are streams of neural crest cells populating the head (A). A collection of vertical cells at B rings three somites. Note that the expression is much stronger than the hindbrain in Figure A.2.

Over 70 embryos were electroporated with the CS-*Hoxb1*/Eng construct at the Stowers Institute, but none of them glowed under the fluorescent scope. This was problematic, especially considering the results of the tissue culture tests, but there are several possible reasons why they didn't glow. The most likely one is that the second

message from the bi-cistronic structure wasn't effective. Despite the lack of fluorescence, the embryos were harvested and prepared for *in situ* hybridization as described in Chapter 4.

The probe used was *Hoxb1* (courtesy of Robb Krumlauf), and the initial results were potentially promising as they looked very different from wild type, both seen in Figure A.6.

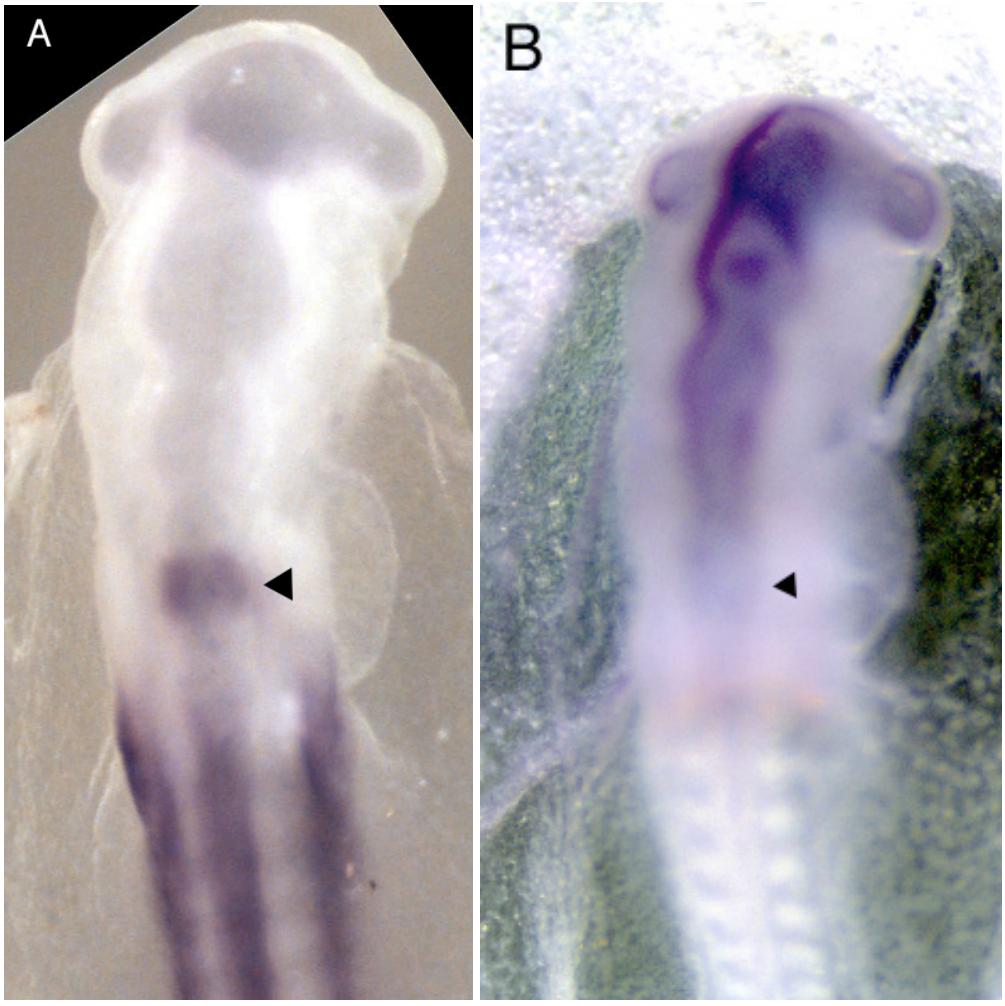


**Figure A.6 *Hoxb1* expression in wild-type (A)** and **CS-*Hoxb1*/Eng (B)** These embryos, stained for the gene *Hoxb1* shows a potential difference in the control and experimental embryos. In the wild-type embryo, the r4 (marked by the black

arrow) and neural tube expression is clearly seen. In addition, there is no expression in the mid or forebrain which is as expected. In the experimental embryo in **(B)** there is no expression in r4 (marked by the black arrow), no expression in the neural tube, but there appears to be expression in the mid and forebrain.

This pattern seen in Figure A.6 was visible in a half dozen embryos, but there were some concerns that the pattern was not real and was, in fact, an artifact brought about by a staining condensate that collected in the neural tube. To check this concern, an antibody treatment was performed on the embryos. The antibodies used would probe for GFP and *Krox20*. The GFP antibody (courtesy of H. McBride) was used to test if the electroporation of the construct worked, as it might be the case that the level of expression of the GFP was just too low to visualize with light microscopy. The *Krox20* antibody (purchased from Covance) was used as the experimental assay, as it would test the connection between the *Hoxb1* and *Krox20*.

For the antibody procedure, the embryos were rinsed in PBS 3 times for 5 minutes each. The PBS was then replaced with a 1:50 concentration of primary antibody in PBS and rocked at room temperature for 2 hours. Another set of 3 PBS rinses followed, followed by a 1:50 concentration of secondary antibody in PBS. After two hours of room temperature rocking with the secondary, the embryos are given their final set of PBS rinses. The secondary antibody contained the fluorescent tag CY-3, so the embryos were screened under a fluorescent microscope.

The GFP antibody worked well against the control embryos that had been electroporated with CA-GFP, and there was clear co-localization between the different colors. However, there was no sign of fluorescence from the experimental embryos that were electroporated with CA-*Hoxb1*/Eng. In addition, the *Krox20* antibody didn't work in any of the embryos, control or experiment. This could be for a variety of reasons, but the first line of investigation should be the fixation process. For instance, when using HNK-1, an antibody which stains migratory neural crest cells, it has been determined that anything more than a 10-minute fix in 4% PFA solution will cause the staining to fail, as will using Bouins' fixation (H. McBride, personal communication). This is unlike *in situ* hybridization in which the amount of fixation time or the fixative used is not an issue. Indeed, leaving embryos in 4% PFA for 2 days at room temperature does not noticeably affect the *in situ* hybridization (data not shown).

While considering the next step to take in this investigation, caveats to this experiment appeared from Andy Groves, a former post-doc at Caltech who is now at the House Ear Institute. Using a construct that is very similar to the CS-*Hoxb1*/Eng repressor, Dr. Groves used a Dlx engrailed repressor construct to look at ear formation. After electroporating this construct in at stage 4, they observed a very clear phenotype: the failure to form a proper ear. As a control they created a variation of their Dlx-engrailed-repressor construct that cannot bind DNA due to point mutations in their homeobox. It was a surprise and a disappointment to discover that the same phenotype appeared. Dr. Groves is still investigating this phenomenon, and has been provided the CS-*Hoxb1*/Eng construct to use in his experiments.

This means that a rigorous control for the CS-*Hoxb1*/Eng experiment would require the electroporation of a construct with a point mutation in the homeobox. This fact, combined with the numerous issues that still required troubleshooting, forced this experiment to be put on hold. It should certainly not be considered a failure, as significant steps have been accomplished in making this new construct to test this connection between *Krox20* and *Hoxb1*.

# Appendix B: Protocols

## *Whole Mount* In Situ *Hybridization*

This protocol is designed for young (<20 somites) chick embryos that do not need to be treated with Proteinase K. It is based on a protocol supplied by Helen McBride who also drew upon information from Reinhard Koester, Andy Groves, and David Wilkinson (Wilkinson, 1992). All solution volumes (except the pre-hyb solution) are the amount needed for each vial being processed, assuming that no more than 10 ml will be used for each wash. Multiply accordingly if necessary.

### General Comments

For performing *in situs*, there are a few ways to handle the samples. Doing everything with 15 ml Falcon tubes is possible, but defiantly the archaic way to go. 12 well culture chambers are nice in that it is much easier to transfer liquids in and out. In addition, empty wells can be used to discard your waste liquids so you can rescue any embryos that may have inadvertently been sucked up. Another way is to use Reactivials

with the cover insert replaced by a fine nylon mesh (Shandon biopsy bags), a method developed by the author. This fluid can then be poured out without losing the embryos. In addition, a suction device can be used, but be warned, it is still possible to suck up the embryos through the mesh. Forcing liquid back in with a pipet is easy. The embryos can stick to the bag, but if this happens, they can be pushed back into the tube with fluid pressure.

## Day 1 Rehydration and Hybridization

## Solutions

### *PBT*

| | | |
|---|---|---|
| Tween 20 | .1% | 500ul |
| 1x PBS | | 500ml |
| Final Volume | | 500.5ml |

Use PBT the same week as you add the Tween.

### *Pre-hybridization solution*

| | | |
|---|---|---|
| Formamide | 50% | 25 ml |
| Depc SSC | 5x | 12.5 ml 20x Depc SSC |
| yeast RNA | 50ug/ml | 125 ul of 20 mg/ml tRNA |
| SDS | 1% | .5 g |
| Heparin | 50 ug/ml | .0025 g |
| Depc $H_2O$ | | 12.375 ml |
| Final Volume | | 50 ml |

Pre-hyb mix can be stored for several weeks at -20°C. Make sure to use SSC made with Depc $H_2O$.

### *Hybridization solution*

Pre-hybridization buffer with probe added. A final volume of 1 ug probe per ml is typical. My probes are washed off a Qiagen column with 50 ul water and added to 150 ul of pre-hyb. I then add 2 ul of this to 100 ul of pre-hyb to make the hybridization solution.

## Protocol

1) Re-hydrate embryos through a Methanol series (75%; 50%; 25% Methanol/PBT) for 5-20 minutes each and wash 2x 5 minutes in PBT.
2) Add .5 ml pre-warmed pre-hyb buffer and swirl embryos around.
3) Replace with 1 ml warmed pre-hyb buffer and let rock at 65°C for 1-2 hours.
4) Replace with .5 ml of hyb solution and rock overnight at 65°C.

## Day 2 Post-hybridization Washes and Antibody Incubation

## Solutions

### *Wash Solution 1*

| | | |
|---|---|---|
| Formamide | 50% | 15 ml |
| SSC, pH 4.5 | 5x | 7.5 ml 20x SSC |
| SDS | 1% | 3 ml 10% SDS |
| ddH20 | | 4.5 ml |
| Final volume | | 30 ml |

### *Wash Solution 2*

| | | |
|---|---|---|
| Formamide | 50% | 15 ml |

| | | |
|---|---|---|
| SSC, pH 4.5 | 2x | 3 ml 20x SSC |
| SDS | .2% | 600 ul 10% SDS |
| ddH20 | | 11.4 ml |
| Final volume | | 30 ml |

***Mab+Lev; Tween***

| | | |
|---|---|---|
| Maleic Acid disodium salt | 100 mM | 2.4 g |
| NaCl | 150 mM | 1.315 g |
| Tween 20 | .1% | 150 ul |
| Levamisole | 2 mM | .07224 g |
| ddH20 | | 150 ml |
| Final volume | | 150 ml |

Add the Levamisole and Tween 20 on the day of use and filter. Levamisole is a phosphatase inhibitor that should inhibit the native alkaline phosphatase and thus reduce the background, but opinions vary as to the effectivenes of this treatment. In general, it won't hurt, but it you forget to add it, you may not even notice the difference. The Mab solution can be used the next day.

***Antibody (Ab) block solution***

| | | |
|---|---|---|
| Blocking Powder | 2% | .16 g |
| Sheep serum | 10% | 800 ul |
| Mab+Lev; Tween | | 8 ml |
| Final volume | | 8.8 ml |

Heat at 65°C with frequent mixing. After the powder dissolves, cool at 4°C until needed.

***Antibody solution***

| | | |
|---|---|---|
| Anti-dig Ab | .1% | 2.5 ul |
| Blocking Solution | | 2.5 ml |
| Final volume | | 2.5 ml |

Use chilled blocking buffer. Store at 4°C until needed. Pre-absorb Ab in block solution for 1 hour before placing with embryos.

## Protocol

Be careful during these washes. The embryos seem to be especially transparent and they are prone to float and stick.

1) Wash 3x 20 minutes with pre-warmed solution 1 at 65°C with rocking.
2) Wash 3x 20 minutes with pre-warmed solution 2 at 65°C with rocking.
3) Wash embryos 3x 5 minutes in Mab+Lev;Tween at room temperature with rocking.
4) Pre-block embryos in 5 ml Ab block solution for 2 hours at room temperature with rocking.
5) Replace with 2.5 ml Ab mixture. Rock gently overnight at 4° C.

## Day 3 Post Antibody Washes

## Solutions

### *Mab+Lev; Tween*

| | | |
|---|---|---|
| Maleic Acid disodium salt | 100 mM | 2.4 g |
| NaCl | 150 mM | 1.315 g |
| Tween 20 | .1% | 150 ul |

| | | |
|---|---|---|
| Levamisole | 2 mM | .07224 g |
| ddH20 | | 150 ml |
| Final volume | | 150 ml |

## Protocol

By Day 3 and the Mab washes, the embryos tend to sink and not stick to the sides of the vials.

1) Wash 3 x 5 minutes in Mab+Lev; Tween at room temp with rocking.
2) Wash 5 x 30-60 minutes in Mab+Lev; Tween at room temp with rocking.
3) Wash overnight in Mab+Lev; Tween at 4° C with rocking. Note, you can also wash at room temp for 2 hours with rocking and continue onto day 4.

## Day 4 Alkaline Phosphatase Detection

## Solutions

### *NTMT*

| | | |
|---|---|---|
| NaCl | 100 mM | 600 ul 5M NaCL |
| Tris, pH 9.5 | 100 mM | 3 ml 1M Tris |
| MgCl2 | 50 mM | 1.5 ml 1M MgCl2 |
| Tween 20 | .1% | 30 ul |
| Levamisole | 2 mM | .0144 g |
| ddH2O | | 24.87 ml |
| Final volume | | 30 ml |

Add the Levamisole and Tween 20 on the day of use.

### *Staining solution*

| | | |
|---|---|---|
| Tween 20 | .1% | 2 ul |
| Levamisole | 2 mM | .000996 g |
| BMPurple | | 2 ml |
| Final volume | | 2 ml |

## Protocol

1) Wash 3 x 10 minutes in NTMT at room temperature with rocking.
2) Replace NTMT with 1 ml of Staining solution.
3) Cover with aluminum foil and let stain for at room temperature with rocking.
4) Check for staining completion. In can be difficult to determine when the stains are done. As a rule of thumb, staining will take at least two hours, but you can continue staining until the background starts to come up. In general, a dissection microscope should be used to judge the staining intensity. With most probes, stain can proceed overnight at 4C with no problems. To speed the reaction, the solution can be replaces several times when you see a precipitate forming. Samples that will be sectioned will need to be over stained.
5) Rinse 2 x 5 minutes in PBT when staining is judged complete.
6) Post-fix in 4% paraformaldehyde for 1 hour at room temperature or overnight at 4°C.
7) Wash 2 x in PBT. If proceeding to gelatin embedding, proceed as normal. For storage or paraffin section, dehydrate in methanol series and store.

## Stock Solutions

The following stock solutions are all computed for a final volume of 100 ml.

### Depc H20

| | | |
|---|---|---|
| Depc | .1 | 100 ul |
| ddH20 | | 100 ml |
| Final volume | | 100.1 ml |

Add Depc and let the solution sit overnight. Autoclave the next day.

### 5M NaCl

| | | |
|---|---|---|
| NaCl | 5 M | 29.22 g |
| ddH20 | | 100 ml |
| Final volume | | 100 ml |

Mix well and autoclave.

### 1M Tris, pH 9.5

| | | |
|---|---|---|
| Tris (base) | 1 M | 12.11 g |
| ddH20 | 100 ml | |
| Final volume | 100 ml | |

Mix well. The pH will initially be around 11. Add hydrochloric acid to reduce pH to 9.5.

### 1M MgCl2

| | | |
|---|---|---|
| MgCl2 | 1 M | 20.33 g |
| DdH20 | | 100 ml |
| Final volume | | 100 ml |

Mix well and autoclave.

## 20x SSC pH 4.5

| | | |
|---|---|---|
| NaCl | 3 M | 17.5 g |
| NaCitrate | 300 mM | 8.82 g |
| DdH20 | | 100 ml |
| | OR | |
| Depc-H20 | | 100 ml |
| Final volume | | 100 ml |

pH with Citric acid to pH 4.5. If this is to be used for the Pre-hyb mix, use Depc-H$_2$0.

## 10% SDS

| | | |
|---|---|---|
| SDS | 10% | 10 g |
| ddH20 | | 100 ml |
| Final volume | | 100 ml |

Mix well and autoclave.

## Chemicals

| | | |
|---|---|---|
| Anti-dig Ab | Boehringer | BM 1093 274 |
| Blocking Solution | Boehringer | BM 1096 176 |
| Formamide | Fisher | BP227-500 |
| Heparin | Sigma | H8514 |
| Maleic Acid disodium salt | Sigma | M9009 |
| yeast RNA | Boehringer | 109 495 |

## *Electrode Construction*

Strip both ends of the 16 gauge wire and solder on banana plugs. Thread the other end of the 16 gauge wire though a holder. Put the platinum wire into the middle of

the copper strands and solder. The 16 gauge wire works well since the plastic coating forces the electrodes to be about 4 mm apart. Use a continuity meter to check that the connection is solid and that there isn't cross talk between the red and black sides. Apply non-conducting epoxy to the end of the electrodes. Make sure that there is enough to cover the joint between the platinum wire and the speaker wire.

## Appendix C: Model Source Code

What follows below is the complete C source code for the model. The source code can be found on the CD-ROM as well.

### *main.c*

```
/*************************************************************************
        This stochastic reaction-diffusion code is designed to study the
        problem of the binding of Retinoic acid binds to the retinoic acid
        receptors and the subsequent creation of the early members of the
        hox family: HoxA1, HoxA2, HoxB1 and Krox20

        Retinoic acid is assumed to be produced at a "point-source"
        located at the caudal section of the hindbrain and its distribution
        is determined by diffusion.

        This code requires Hox.h as its header file. 'Hox.c' furnishes
        all the routines that implement the physical effects of the reactions
        used as well as the diffusion code for RA,

        These functions are accessed by the Reaction[]() and Diffusion[]()
        functions, which are implemented as function arrays; this makes the
        bookeeping quite simple.

        Usage: a.out seed [val] stop [val] write [val]

*************************************************************************/
```

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <string.h>
#include <sys/stat.h>
#include <unistd.h>
#include <fcntl.h>
#include <sys/types.h>
#include <sys/times.h>
#include <time.h>
#include "Hox.h"
#include "header.h"
#include "UpdateAmu.h"
#include "ranlib.h"


char reactiontypes[NUM_FUNCS][25] = {
        "MakeRA",                   /* 0 */
        "BindRA",                   /* 1 */
        "DecayRA",                  /* 2 */
        "MakeRAR",                  /* 3 */
        "DecayRAR",                 /* 4 */
        "UnbindBRAR",               /* 5 */
        "DecayBRAR",                /* 6 */
        "ActivateA1",               /* 7 */
        "UnActivateA1",             /* 8 */
        "TranscribeA1",             /* 9 */
        "DecaymA1",                 /* 10 */
        "TranslateA1",              /* 11 */
        "Decaya1",                  /* 12 */
        "ActivateB1",               /* 13 */
        "UnActivateB1",             /* 14 */
        "TranscribeB1",             /* 15 */
        "DecaymB1",                 /* 16 */
        "TranslateB1",              /* 17 */
        "SuperActivateB1",          /* 18 */
        "UnSuperActivateB1",        /* 19 */
        "TranscribeSuperB1",        /* 20 */
        "RepressB1",                /* 21 */
        "UnRepressB1",              /* 22 */
        "AutoActivateB1",           /* 23 */
        "UnAutoActivateB1",          /* 24 */
        "TranscribeAutoB1",          /* 25 */
        "Decayb1",                  /* 26 */
        "ActivateB2",               /* 27 */
        "UnActivatedB2",            /* 28 */
```

```
        "TranscribeB2",         /* 29 */
        "DecaymB2",             /* 30 */
        "TranslateB2",          /* 31 */
        "Decayb2",              /* 32 */
        "Divide",               /* 33 */
        "ActivateKrox",         /* 34 */
        "UnActivateKrox",       /* 35 */
        "TranscribeKrox",       /* 36 */
        "DecaymKrox",           /* 37 */
        "RepressKrox",          /* 38 */
        "UnRepressKrox",        /* 39 */
        "TranslateKrox",        /* 40 */
        "Decaykrox",            /* 41 */
        "BindRXR",              /* 42 */
        "MakeRXR",              /* 43 */
        "DecayRXR",             /* 44 */
        "UnbindBRXR",           /* 45 */
         "DecayBRXR",           /* 46 */
         "Dimerize",            /* 47 */
        "UnDimerize",           /* 48 */
        "DecayDimer",           /* 49 */
        "Complexa1",            /* 50 */
        "Uncomplexa1",          /* 51 */
        "Decaya1Complex",       /* 52 */
        "Complexb1",            /* 53 */
        "Uncomplexb1",          /* 54 */
        "Decayb1Complex",       /* 55 */
        "MakeComplex",          /* 56 */
        "DecayComplex"};        /* 57 */


int
main(int argc,char * argv[])
{
        extern int DEBUG;
        void    srand48();
        double drand48();
        int     i,k,mu,x_i,switch_flag;
        struct stat buf;
        int     write_flag,count;
        int     diff_count,reac_count, result;
        int    diffusions, reactions, failures;
        long    seedval;
        float   a_summ,T,delta_t,t_stop,t_write;
        float   a0,r2a0;
        double  r1,r2;
```

```
        float   *image,*dummy;
        float    prob;
        int      a,b;
        char    name_buff[80];
        clock_t time;
        long double  d_time;
        CELL *          voxel;
        int     channels[NUM_FUNCS];
        int       border;
        int     counter;
        char * input;
        char * output;
        float tmp;
        float ktmp;
        FILE    *COUNT;
        FILE    *BORDER;
        FILE *fp;

        time = clock();
        seedval = 13;
        t_stop = 5000.0;
        t_write = 1.0;
        delta_t = 10.0;
        result = 0;
        x_i = 0;
        DEBUG = 0;
        input = (char *) NULL;

/******************** Get setup data from command line ********************/

        output = strdup("output");
        switch (argc) {
                case 1:
                        break;
                case 2:
                        input = strdup(argv[1]);
                        fp = fopen(input,"r");
                        seedval = (long) getValue(fp,"seed:");
                        t_stop = (float) getValue(fp,"stop:");
                        delta_t = (float) getValue(fp,"delta_t:");
                        fclose(fp);
                        break;
                case 3:
                        input = strdup(argv[1]);
                        output = strdup(argv[2]);
                        fp = fopen(input,"r");
```

```
                seedval = (long) getValue(fp,"seed:");
                t_stop = (float) getValue(fp,"stop:");
                delta_t = (float) getValue(fp,"delta_t:");
                fclose(fp);
                break;
        case 7:
                if (strcmp(argv[1],"seed") == 0) {
                        seedval = (double)atof(argv[2]);
                        t_stop = atof(argv[4]);
                        delta_t = atof(argv[6]);
                }
                else {
                        printf("Syntax error....\n");
                        exit (1);
                }
                break;
        default:
                printf("Syntax error...\n");
                exit (1);
                break;
}

/********************* Setup Initial Conditions ***************************/

/* Check that the output directory exists */
        sprintf(name_buff,"%s.%li/",output,seedval);
        if(stat(name_buff,&buf) == -1) {
                printf("Directory %s doesn't exist\n",name_buff);
                mkdir(name_buff,511);
        }
        else {
                printf("Directory %s exists\n",name_buff);
        }

        if ((COUNT = fopen("count","w")) == NULL) {
                printf("Cannot open data file count\n");
                exit (1);
        }

        sprintf(name_buff,"%s.%li/border",output,seedval);
        if ((BORDER = fopen(name_buff,"w")) == NULL) {
                printf("Cannot open data file count\n");
                exit (1);
        }

        srand48(seedval);
```

```
/*      Setup Initial Conditions for "Concentrations"        */

        NX = 40;
        dummy = (float *) malloc(2*NX*sizeof(float));
        image = (float *) malloc(2*NX*sizeof(float));
        voxel = init(2*NX);
        read_inputs(input, &initial_ra, &initial_rar, &D_ra,
             &a1hill, &b1hill, &b1auto,&rephill,&b2hill,C_mu,K);
/* Zero out some of the late events */

        tmp = C_mu[repressB1];
        ktmp = C_mu[activateKrox];
        C_mu[activateKrox] = 0.0;
        C_mu[repressB1] = 0.0;

        border = 19;
        prob = .01;
        for (i = 0; i < NX; i++) {
                a = (int) ignbin((long) initial_rar,prob);
                b = (int) ignbin((long) 10,.5);
                if(b <= 5)
                        voxel[i].rar = initial_rar+a;
                else
                        voxel[i].rar = initial_rar-a;
                a = (int) ignbin((long) initial_rar,prob);
                b = (int) ignbin((long) 10,.5);
                if(b <= 5)
                        voxel[i].rxr = initial_rar+a;
                else
                        voxel[i].rxr = initial_rar-a;

                voxel[i].plex = initial_rar/4;

/* genes */
                voxel[i].A1 = voxel[i].B2 = voxel[i].B1 = voxel[i].Krox = 1;
                dummy[i] = (float)i;

/* initial rhombomere identities */
                if(i < 20) {
                        voxel[i].id = R5;
                }
                else if ((i >= 20) && (i < 40)) {
                        voxel[i].id = R4;
                }
                else {
                        voxel[i].id = R3;
```

```
                }

                for (k = 0; k < NUM_FUNCS; k++) {
                        voxel[i].a_mu[k] = 0.0;
                }

/* Divide */
        voxel[i].a_mu[divide] = C_mu[divide];
        voxel[i].a_mu[makeRAR] = C_mu[makeRAR];
        voxel[i].a_mu[makeRXR] = C_mu[makeRXR];
        voxel[i].a_mu[decayRAR] = C_mu[decayRAR]*initial_rar;
        voxel[i].a_mu[makeComplex] = C_mu[makeComplex];
        voxel[i].a_mu[decayComplex] = C_mu[decayComplex];
           }
           for (k = 0; k < NUM_FUNCS; k++) {
                channels[k] = 0;
           }

/* Setup RA Source */

           counter = 0;
           voxel[0].ra = initial_ra;
           voxel[0].d_ra = voxel[0].ra*D_ra;
           update_cmu0(voxel,0.0);
           Update[bindRAR](voxel);
           Update[bindRXR](voxel);
           Update[decayRA](voxel);

           voxel[0].a_mu[divide] = 0.0;


/*****************************************************************/

           T = 0.0;
           write_flag = 0;
           count = 0;
           diff_count = 0;
           reac_count = 0;
           reactions = diffusions = failures = 0;


           while (T < t_stop) {                 /* start main loop */
                a0 = 0.0;
                for (i = 0; i < NX; i++) {   /* compute sum of react/diff params */
                        for (k = 0; k < NUM_FUNCS; k++) {
                                a0 += (voxel+i)->a_mu[k];    /* reaction values */
```

```
                        }
                        a0 += (voxel+i)->d_ra;
                }
                if (a0 == 0) {
                        printf("Unknown Error: a0 = 0...\n");
                        exit(1);
                }
                r1 = drand48();
                r2 = drand48();
                T += -log(r1)/a0;
/****          Check T to see if it is time to write data files       ****/
                if (T >= t_write) {

/* Start the late events  */
if(T >= 21000.5 && T <= 21500.5) {
        C_mu[repressB1] = tmp;
C_mu[activateKrox] = ktmp;
}



                        printf("Write Image Data.  Sim Time = %4.2e secs\n",T);
                        sprintf(name_buff,"%s.%li/%s.%d.dat",output,seedval,"ra",count);
                        for (i = 0; i < NX; i++) {
                                image[i] = voxel[i].ra;
                        }
                        write_gnu_data_file(image,dummy,NX,name_buff,2);

        sprintf(name_buff,"%s.%li/%s.%d.dat",output,seedval,"brar",count);
                        for (i = 0; i < NX; i++) {
                                image[i] = voxel[i].brar;
                        }
                        write_gnu_data_file(image,dummy,NX,name_buff,2);

        sprintf(name_buff,"%s.%li/%s.%d.dat",output,seedval,"brxr",count);
                        for (i = 0; i < NX; i++) {
                                image[i] = voxel[i].brxr;
                        }
                        write_gnu_data_file(image,dummy,NX,name_buff,2);

        sprintf(name_buff,"%s.%li/%s.%d.dat",output,seedval,"hoxa1",count);
                        for (i = 0; i < NX; i++) {
                                image[i] = voxel[i].a1;
                        }
                        write_gnu_data_file(image,dummy,NX,name_buff,2);
```

```
sprintf(name_buff,"%s.%li/%s.%d.dat",output,seedval,"thoxa1",count);
                for (i = 0; i < NX; i++) {
                        image[i] = voxel[i].a1 + voxel[i].a1plex;
                }
                write_gnu_data_file(image,dummy,NX,name_buff,2);

sprintf(name_buff,"%s.%li/%s.%d.dat",output,seedval,"rar",count);
                for (i = 0; i < NX; i++) {
        image[i] = voxel[i].rar;
                }
                write_gnu_data_file(image,dummy,NX,name_buff,2);

sprintf(name_buff,"%s.%li/%s.%d.dat",output,seedval,"rxr",count);
                for (i = 0; i < NX; i++) {
        image[i] = voxel[i].rxr;
                }
                write_gnu_data_file(image,dummy,NX,name_buff,2);

sprintf(name_buff,"%s.%li/%s.%d.dat",output,seedval,"dimer",count);
                for (i = 0; i < NX; i++) {
        image[i] = voxel[i].dimer;
                }
                write_gnu_data_file(image,dummy,NX,name_buff,2);

sprintf(name_buff,"%s.%li/%s.%d.dat",output,seedval,"hoxb1",count);
                for (i = 0; i < NX; i++) {
        image[i] = voxel[i].b1;
                }
                write_gnu_data_file(image,dummy,NX,name_buff,2);

sprintf(name_buff,"%s.%li/%s.%d.dat",output,seedval,"plex",count);
                for (i = 0; i < NX; i++) {
        image[i] = voxel[i].plex;
                }
                write_gnu_data_file(image,dummy,NX,name_buff,2);

sprintf(name_buff,"%s.%li/%s.%d.dat",output,seedval,"a1plex",count);
                for (i = 0; i < NX; i++) {
        image[i] = voxel[i].a1plex;
                }
                write_gnu_data_file(image,dummy,NX,name_buff,2);

sprintf(name_buff,"%s.%li/%s.%d.dat",output,seedval,"b1plex",count);
                for (i = 0; i < NX; i++) {
        image[i] = voxel[i].b1plex;
```

```
                }
                write_gnu_data_file(image,dummy,NX,name_buff,2);

sprintf(name_buff,"%s.%li/%s.%d.dat",output,seedval,"thoxb1",count);
                for (i = 0; i < NX; i++) {
        image[i] = voxel[i].b1plex + voxel[i].b1;
                }
                write_gnu_data_file(image,dummy,NX,name_buff,2);

sprintf(name_buff,"%s.%li/%s.%d.dat",output,seedval,"hoxb2",count);
                for (i = 0; i < NX; i++) {
        image[i] = voxel[i].b2;
                }
                write_gnu_data_file(image,dummy,NX,name_buff,2);

sprintf(name_buff,"%s.%li/%s.%d.dat",output,seedval,"krox20",count);
                for (i = 0; i < NX; i++) {
        image[i] = voxel[i].krox;
                }
                write_gnu_data_file(image,dummy,NX,name_buff,2);

sprintf(name_buff,"%s.%li/%s.%d.dat",output,seedval,"mhoxa1",count);
                for (i = 0; i < NX; i++) {
        image[i] = voxel[i].mA1;
                }
                write_gnu_data_file(image,dummy,NX,name_buff,2);

sprintf(name_buff,"%s.%li/%s.%d.dat",output,seedval,"mhoxb1",count);
                for (i = 0; i < NX; i++) {
        image[i] = voxel[i].mB1;
                }
                write_gnu_data_file(image,dummy,NX,name_buff,2);

sprintf(name_buff,"%s.%li/%s.%d.dat",output,seedval,"mhoxb2",count);
                for (i = 0; i < NX; i++) {
        image[i] = voxel[i].mB2;
                }
                write_gnu_data_file(image,dummy,NX,name_buff,2);

sprintf(name_buff,"%s.%li/%s.%d.dat",output,seedval,"mkrox20",count);
                for (i = 0; i < NX; i++) {
        image[i] = voxel[i].mKrox;
                }
                write_gnu_data_file(image,dummy,NX,name_buff,2);
                t_write += delta_t;
                d_time = (clock()-time)/CLOCKS_PER_SEC;
```

```
                        printf("Reac Count = %d  Diff Count =
%d\n",reac_count,diff_count);
                        printf("Incremental CPU Time = %4.2Le secs \n\n",d_time);
                        count++;
                        if(reac_count > 400000) {
                                printf("I'm stuck! Bailing out\n");
                                exit(1);
                        }
                        reac_count = 0;
                        diff_count = 0;
                        time = d_time;
                        for (k = 0; k < NUM_FUNCS; k++) {
                                fprintf(COUNT,"a_mu[%s (%d)] called %d times\n",
                                        reactiontypes[k],k,channels[k]);
                        }
                        fprintf(COUNT,
                                        "\nReac Count = %d Diff Count = %d  Fail Count =
%d\n",
                                        reactions,diffusions,failures);
                        fflush(COUNT);

                        fprintf(BORDER,"%d\n",border);
                        fflush(BORDER);
                }
                r2a0 = r2*a0;
                a_summ = 0;
                mu = 0;
                switch_flag = 0;
                for (i = 0; i < NX; i++) {
                        x_i = i;
                        for (k = 0; k < NUM_FUNCS; k++) {
                                mu = k;
                                a_summ += voxel[i].a_mu[k];
                                if (a_summ >= r2a0) {
                                        switch_flag = 1;
                                        goto React;
                                }
                        }
                        a_summ += voxel[i].d_ra;
                        if (a_summ >= r2a0) {
                                switch_flag = 2;
                                mu = 0;
                                goto React;
                        }
                }
React:  if (switch_flag != 0) {
```

```
                        switch (switch_flag) {
                                case 1:
if(DEBUG) {
        printf("calling a_mu[%d] in cell %d\n",mu,x_i);
}
                                        channels[mu] += 1;
                                        if(mu == divide) {
                                                (voxel+x_i)->a_mu[divide] = 0.0;
                                                printf("Cell %d is dividing!\n",x_i);
                                                if(x_i <=  border)
                                                        border++;
                                                NX++;
                                                result = Reaction[mu](voxel+x_i,NX);
                                voxel[x_i].a_mu[divide] = 0.0;
                                voxel[x_i+1].a_mu[divide] = 0.0;
                                        }
                                        else {
                                                result = Reaction[mu](voxel+x_i);
                                        }
                                        reac_count += 1;
                                        reactions +=1;
                                break;
                                case 2:
if(DEBUG) {
        printf("calling diffusion in cell %d\n",x_i);
}
                                        result = Diffusion[mu](voxel+x_i);
                                        diff_count += 1;
                                        diffusions +=1;
                                        break;
                        }
                        update_cmu0(voxel,T);
                }
                if(!result) {
/* Back out the time */
                        T -= -log(r1)/a0;
                        if(switch_flag == 1) {
                                printf("Error: Called a_mu[%s (%d)] = %f in cell %d\n",
                                        reactiontypes[mu],mu,voxel[x_i].a_mu[mu],x_i);
                        }
                        failures += 1;
                        exit(1);
                }
        }           /* End Main Loop */
```

```
        printf("Reac Count = %d  Diff Count = %d, Fail Count = %d\n",
        reactions,diffusions,failures);
        fclose(COUNT);
        fclose(BORDER);
        exit (0);
}
```

### *inputs.c*

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include "header.h"


double
getValue(FILE * fp,char * tag) {
  char buf[80];
  char *ptr;
  char *res;
  double val;

  res = fgets(buf,80,fp);

/* Look for the line with the right tag */
  while(!(ptr = strstr(buf,tag)) && res) {
    res = fgets(buf,80,fp);
  }
  if(!res) {
   printf("The tag %s was not found\n",tag);
   exit(1);
  }

/* Now that we are on the right line, look for the colon */

  while(*ptr != ':') {
    ptr++;
  }

/* Move past the colon */
  ptr++;

/* the next thing is the value we want. */
  val = atof(ptr);

/* Rewind the stream to the beginning of the file */
```

```
  rewind(fp);

  return val;
}

void
read_inputs(char * filename,int * init_ra, int * init_rar, float *D_ra, double *a1hill,
        double *b1hill, double *b1auto, double *rephill, double *b2hill, float c_mu[],
        float K[])
{

  FILE * fp;
  char buf[20];
  int i;

  fp = fopen(filename,"r");

  *init_ra = (int) getValue(fp,"initial_ra");
  *init_rar = (int) getValue(fp,"initial_rar");

  *D_ra   = (float) getValue(fp,"D_ra");
  *a1hill = (double) getValue(fp,"a1hill");
  *b1hill = (double) getValue(fp,"b1hill");
  *b1auto = (double) getValue(fp,"b1auto");
  *rephill = (double) getValue(fp,"rephill");
  *b2hill = (double) getValue(fp,"b2hill");

// read in the production rate values
  for(i = 0; i < NUM_FUNCS; i++) {
     sprintf(buf,"c_mu%d",i);
     c_mu[i] = (float) getValue(fp,buf);
  }
  for(i = 0; i < 7; i++) {
     sprintf(buf,"K%d",i);
     K[i] = (float) getValue(fp,buf);
  }
}
```

## *II.c*

```
#include <stdio.h>
#include <strings.h>
#include <string.h>
#include <stdlib.h>
#include <math.h>
```

```
#include "ranlib.h"
#include "Hox.h"
#include "header.h"
#include "UpdateAmu.h"


void
update(CELL *c) {
   int i;
   for(i = 0; i < NUM_FUNCS; i++) {
      Update[i](c);
   }
}

void
print_cell(FILE * DATA,CELL * c) {
        if(c->num == 0) {
                fprintf(DATA,"num\tra\trar\trxr\tbrar\tbrxr\tdimer\t");
                fprintf(DATA,"a1\tb1\tb2\tkrox\tmA1\tmB1\tmB2\t");
                fprintf(DATA,"mKrox\tplex\ta1plex\tb1plex\n");
        }
        fprintf(DATA,"%d\t%d\t%d\t%d\t%d\t%d\t%d\t%d\t%d\t%d\t%d\t%d\t%d\t%d\t%d\t%d\t%d\t
%d\t%d\t%d\t%d\n",c->num+1, c->ra, c->rar, c->rxr, c->brar, c->brxr, c->dimer,
                        c->a1, c->b1, c->b2, c->krox, c->mA1, c->mB1, c->mB2,
                        c->mKrox, c->plex, c->a1plex, c->b1plex);
}


void
divide_resources(CELL * mom, CELL * daughter) {
    int a;

    float prob = .5;
    a = (int) ignbin((long) mom->a1 ,prob);
    daughter->a1 -= a;
    mom->a1 = a;

    a = (int) ignbin((long) mom->mA1 ,prob);
    daughter->mA1 -= a;
    mom->mA1 = a;

    a = (int) ignbin((long) mom->b1 ,prob);
    daughter->b1 -= a;
    mom->b1 = a;

    a = (int) ignbin((long) mom->mB1 ,prob);
```

```
daughter->mB1 -= a;
mom->mB1 = a;

a = (int) ignbin((long) mom->b2 ,prob);
daughter->b2 -= a;
mom->b2 = a;

a = (int) ignbin((long) mom->mB2 ,prob);
daughter->mB2 -= a;
mom->mB2 = a;

a = (int) ignbin((long) mom->mKrox ,prob);
daughter->mKrox -= a;
mom->mKrox = a;

a = (int) ignbin((long) mom->krox ,prob);
daughter->krox -= a;
mom->krox = a;

a = (int) ignbin((long) mom->brar ,prob);
daughter->brar -= a;
mom->brar = a;

a = (int) ignbin((long) mom->brxr ,prob);
daughter->brxr -= a;
mom->brxr = a;

a = (int) ignbin((long) mom->dimer ,prob);
daughter->dimer -= a;
mom->dimer = a;

a = (int) ignbin((long) mom->rar ,prob);
daughter->rar -= a;
mom->rar = a;

a = (int) ignbin((long) mom->plex ,prob);
daughter->plex -= a;
mom->plex = a;

a = (int) ignbin((long) mom->a1plex ,prob);
daughter->a1plex -= a;
mom->a1plex = a;

a = (int) ignbin((long) mom->b1plex ,prob);
daughter->b1plex -= a;
mom->b1plex = a;
```

```
    a = (int) ignbin((long) mom->rxr ,prob);
    daughter->rxr -= a;
    mom->rxr = a;
#ifdef DEBUG
    printf("mom has %d, daughter has %d rar\n",a,daughter->rar);
#endif
}

void
add(CELL *head, CELL *new)
{
   CELL *ptr;
   ptr = head;
   while((ptr->next != (CELL*) NULL) && (new->num > ptr->next->num)) {
        ptr = ptr->next;
   }
   if(ptr->next == (CELL*) NULL) {
     ptr->next = new;
     new->next =  (CELL *) NULL;
   }
   else {
     new->next = ptr->next;
     ptr->next = new;
   }
}

int
Divide(CELL *afterme, int NX)
{
   CELL *ptr;
   int i;
   int ncells;

   int flag = 0;
   ptr = afterme;
   ncells = NX-afterme->num;
   for(i = 0; i < ncells; i++) {
      (afterme+i)->num++;
   }
   memmove(afterme+1,afterme, ncells*sizeof(CELL));
   memcpy(afterme,afterme+1,sizeof(CELL));
   divide_resources(afterme,afterme+1);

   update(afterme);
   update(afterme+1);
```

```
   afterme->num--;
   flag = 1;
   return flag;
}

CELL*
init(int size) {
   int i;
   CELL * head;
   head = (CELL*) calloc(size,sizeof(CELL));
   for(i = 0; i < size-1; i++) {
     (head+i)->num = i;
     (head+i)->next = (head+i+1);
   }
   (head+(size-1))->num = size-1;
   (head+(size-1))->next = (CELL *) NULL;
   return head;
}
```

### *write_gnu_data_file.c*

```
#include <stdio.h>
#include <string.h>
#include <math.h>
#include <malloc.h>

void
write_gnu_data_file(float *array,float *farray,int length,char*
fname,int d_flag)
{
       int     jj;
       float   norm_dist;
       FILE    *fpo;

       if ((fpo = fopen(fname,"w")) == NULL) {
         printf("Cannot open gnu data file %s\n",fname);
         exit (0);
       }

       for (jj = 0; jj < length; jj++) {
         if (d_flag == 1)
            norm_dist = (float)jj/(float)length;
          else if (d_flag == 2)
              norm_dist = (float)jj+1.0;
         else
            norm_dist = farray[jj];
```

```
        fprintf(fpo,"%0.3f    %0.5f\n",norm_dist,array[jj]);
      }
        fprintf(fpo,"\n");
        fclose(fpo);
}
```

## *header.h*

```
#ifndef __HEADER_H
#define __HEADER_H
#include "Hox.h"

void update();
void divide_resources();
void print_cell();
void add();
int Divide();
CELL* init();
double getValue();
void read_inputs();
void write_gnu_data_file();

#endif
```

## *Hox.h*

```
/*
 *      Header file for stochastic simulation of study of Retinoic Acid
 *      diffusion and the production of the early members of the hox
 *      family using the extended Gillespie formulation for 1-dimensional
 *      reaction-diffusion.
 *
 */

#ifndef __HOX_H_
#define __HOX_H_

#defineNUM_FUNCS 58
#define PERMS 0666

typedef enum { R3,R4,R5 } rhombomere;
typedef enum {A,B} Repressor;
```

```
/* Note that the convention adopted is that the uppercase letters stand
 * for the DNA and the lowercase stand for proteins
 */

typedef struct cell {
      int  num;                 /* Cell number */
      rhombomere  id;           /* The identity of the cell */
      int  ra;                  /* number of unbound RA molecules */
      int  rar;                 /* number of RA receptors */
      int  rxr;                 /* number of RA receptors */
      int  brar;                /* number of bound RA molecules  */
      int  brxr;                /* number of bound RA molecules  */
      int  dimer;               /* number of rar/rxr dimers */
      int  A1;                  /* number of A1 genes */
      int  actA1;               /* number of activated A1 genes */
      int  B1;                  /* number of B1 genes */
      int  actB1;               /* number of activated B1 genes */
      int  superactB1;          /* number of super activated B1 genes */
      int  repB1;               /* number of repressed B1 genes */
      int  autoB1;              /* number of auto activated B1 genes */
      int  B2;                  /* number of B2 genes */
      int  actB2;               /* number of activated B2 genes */
      int  Krox;                /* number of Krox20 genes */
      int  actKrox;             /* number of activated Krox20 genes */
      int  repKrox;             /* number of repressed Krox20 genes */
      Repressorkrep;            /* what the current repressor for krox is */
      int  plex;                /* number of complexes availble */
      int  mA1;                 /* number of A1 mRNA */
      int  mB1;                 /* number of B1 mRNA */
      int  mB2;                 /* number of B2 mRNA */
      int  mKrox;               /* number of Krox20 mRNA */
      int  a1;                  /* number of a1 proteins */
      int  a1plex;              /* number of a1+pbx+prep molecules */
      int  b1;                  /* number of b1 proteins */
      int  b1plex;              /* number of a1+pbx+prep molecules */
      int  b2;                  /* number of b2 proteins */
      int  krox;                /* number of krox20 proteins */
      float   d_ra;             /* Retinoic acid diffusion coefficient */
      float   a_mu[NUM_FUNCS];/* Reaction probabilities */
      struct cell *next; /* Pointer to the next cell */
} CELL;


enum {makeRA,               /* 0 */
   bindRAR,                 /* 1 */
   decayRA,                 /* 2 */
```

```
makeRAR,            /* 3 */
decayRAR,           /* 4 */
unbindBRAR,         /* 5 */
decayBRAR,          /* 6 */
activateA1,         /* 7 */
unActivateA1,       /* 8 */
transcribeA1,       /* 9 */
decaymA1,           /* 10 */
translateA1,        /* 11 */
decaya1,            /* 12 */
activateB1,         /* 13 */
unActivateB1,       /* 14 */
transcribeB1,       /* 15 */
decaymB1,           /* 16 */
translateB1,        /* 17 */
superActivateB1,    /* 18 */
unSuperActivateB1,  /* 19 */
transcribeSuperB1,  /* 20 */
repressB1,          /* 21 */
unRepressB1,        /* 22 */
autoActivateB1,     /* 23 */
unAutoActivateB1,   /* 24 */
transcribeAutoB1,   /* 25 */
decayb1,            /* 26 */
activateB2,         /* 27 */
unActivateB2,       /* 28 */
transcribeB2,       /* 29 */
decaymB2,           /* 30 */
translateB2,        /* 31 */
decayb2,            /* 32 */
divide,             /* 33 */
activateKrox,       /* 34 */
unActivateKrox,     /* 35 */
transcribeKrox,     /* 36 */
decaymKrox,         /* 37 */
repressKrox,        /* 38 */
unRepressKrox,      /* 39 */
translateKrox,      /* 40 */
decaykrox,          /* 41 */
bindRXR,            /* 42 */
makeRXR,            /* 43 */
decayRXR,           /* 44 */
unbindBRXR,         /* 45 */
decayBRXR,          /* 46 */
dimerize,           /* 47 */
unDimerize,         /* 48 */
```

```
    decayDimer,             /* 49 */
    complexa1,              /* 50 */
    unComplexa1,            /* 51 */
    decaya1Complex,         /* 52 */
    complexb1,              /* 53 */
    unComplexb1,            /* 54 */
    decayb1Complex,         /* 55 */
    makeComplex,            /* 56 */
    decayComplex};          /* 57 */


typedef int (*REACTION)();

/******************** Function Declarations ***************************/

int     update_cmu0(CELL*,float);

int     MakeRA(CELL *);             /* a_mu[0] */
int     BindRAR(CELL *);            /* a_mu[1] */
int     DecayRA(CELL *);            /* a_mu[2] */
int     MakeRAR(CELL *);            /* a_mu[3] */
int     DecayRAR(CELL *);           /* a_mu[4] */
int     UnbindBRAR(CELL *);         /* a_mu[5] */
int     DecayBRAR(CELL *);          /* a_mu[6] */
int     ActivateA1(CELL *);         /* a_mu[7] */
int     UnActivateA1(CELL *);       /* a_mu[8] */
int     TranscribeA1(CELL *);       /* a_mu[9] */
int     DecaymA1(CELL *);           /* a_mu[10] */
int     TranslateA1(CELL *);        /* a_mu[11] */
int     Decaya1(CELL *);            /* a_mu[12] */
int     ActivateB1(CELL *);         /* a_mu[13] */
int     UnActivateB1(CELL *);       /* a_mu[14] */
int     TranscribeB1(CELL *);       /* a_mu[15] */
int     DecaymB1(CELL *);           /* a_mu[16] */
int     TranslateB1(CELL *);        /* a_mu[17] */
int     SuperActivateB1(CELL *);    /* a_mu[18] */
int     UnSuperActivateB1(CELL *);/* a_mu[19] */
int     TranscribeSuperB1(CELL *);/* a_mu[20] */
int     RepressB1(CELL *);          /* a_mu[21] */
int     UnRepressB1(CELL *);        /* a_mu[22] */
int     AutoActivateB1(CELL *);     /* a_mu[23] */
int     UnAutoActivateB1(CELL *);/* a_mu[24] */
int     TranscribeAutoB1(CELL *);/* a_mu[25] */
int     Decayb1(CELL *);            /* a_mu[26] */
int     ActivateB2(CELL *);         /* a_mu[27] */
int     UnActivateB2(CELL *);       /* a_mu[28] */
```

```
int     TranscribeB2(CELL *);       /* a_mu[29] */
int     DecaymB2(CELL *);           /* a_mu[30] */
int     TranslateB2(CELL *);        /* a_mu[31] */
int     Decayb2(CELL *);            /* a_mu[32] */
int     Divide(CELL *, int);        /* a_mu[33] */
int     ActivateKrox(CELL *);       /* a_mu[34] */
int     UnActivateKrox(CELL *);     /* a_mu[35] */
int     TranscribeKrox(CELL *);     /* a_mu[36] */
int     DecaymKrox(CELL *);         /* a_mu[37] */
int     RepressKrox(CELL *);        /* a_mu[38] */
int     UnRepressKrox(CELL *);      /* a_mu[39] */
int     TranslateKrox(CELL *);      /* a_mu[40] */
int     Decaykrox(CELL *);          /* a_mu[41] */
int     BindRXR(CELL *);            /* a_mu[42] */
int     MakeRXR(CELL *);            /* a_mu[43] */
int     DecayRXR(CELL *);           /* a_mu[44] */
int     UnbindBRXR(CELL *);         /* a_mu[45] */
int     DecayBRXR(CELL *);          /* a_mu[46] */
int     Dimerize(CELL *);           /* a_mu[47] */
int     UnDimerize(CELL *);         /* a_mu[48] */
int     DecayDimer(CELL *);         /* a_mu[49] */
int     Complexa1(CELL *);          /* a_mu[50] */
int     Uncomplexa1(CELL *);        /* a_mu[51] */
int     Decaya1Complex(CELL *);     /* a_mu[52] */
int     Complexb1(CELL *);          /* a_mu[53] */
int     Uncomplexb1(CELL *);        /* a_mu[54] */
int     Decayb1Complex(CELL *);     /* a_mu[55] */
int     MakeComplex(CELL *);        /* a_mu[56] */
int     DecayComplex(CELL *);       /* a_mu[57] */

int     RA_Diffusion(CELL *);
/********************** Initializations *********************************/


static REACTION     Reaction[] = {
                        MakeRA,                 /* 0 */
                        BindRAR,                /* 1 */
                        DecayRA,                /* 2 */
                        MakeRAR,                /* 3 */
                        DecayRAR,               /* 4 */
                        UnbindBRAR,             /* 5 */
                        DecayBRAR,              /* 6 */
                        ActivateA1,             /* 7 */
                        UnActivateA1,           /* 8 */
                        TranscribeA1,           /* 9 */
                        DecaymA1,               /* 10 */
```

```
TranslateA1,            /* 11 */
Decaya1,                /* 12 */
ActivateB1,             /* 13 */
UnActivateB1,           /* 14 */
TranscribeB1,           /* 15 */
DecaymB1,               /* 16 */
TranslateB1,            /* 17 */
SuperActivateB1,        /* 18 */
UnSuperActivateB1,      /* 19 */
TranscribeSuperB1,      /* 20 */
RepressB1,              /* 21 */
UnRepressB1,            /* 22 */
AutoActivateB1,         /* 23 */
UnAutoActivateB1,       /* 24 */
TranscribeAutoB1,       /* 25 */
Decayb1,                /* 26 */
ActivateB2,             /* 27 */
UnActivateB2,           /* 28 */
TranscribeB2,           /* 29 */
DecaymB2,               /* 30 */
TranslateB2,            /* 31 */
Decayb2,                /* 32 */
Divide,                 /* 33 */
ActivateKrox,           /* 34 */
UnActivateKrox,         /* 35 */
TranscribeKrox,         /* 36 */
DecaymKrox,             /* 37 */
RepressKrox,            /* 38 */
UnRepressKrox,          /* 39 */
TranslateKrox,          /* 40 */
Decaykrox,              /* 41 */
BindRXR,                /* 42 */
MakeRXR,                /* 43 */
DecayRXR,               /* 44 */
UnbindBRXR,             /* 45 */
DecayBRXR,              /* 46 */
Dimerize,               /* 47 */
UnDimerize,             /* 48 */
DecayDimer,             /* 49 */
Complexa1,              /* 50 */
Uncomplexa1,            /* 51 */
Decaya1Complex,         /* 52 */
Complexb1,              /* 53 */
Uncomplexb1,            /* 54 */
Decayb1Complex,         /* 55 */
MakeComplex,            /* 56 */
```

```
                        DecayComplex};                  /* 57 */


static REACTION      Diffusion[] = { RA_Diffusion };

float    C_mu[NUM_FUNCS];
float K[7];

int      initial_ra;
int      initial_rar;
float    D_ra;
float    Kg;
double a1hill;
double b1hill;
double b1auto;
double  rephill;
double b2hill;
float    G1;
float    G2;
int NX;
int DEBUG;

#endif
```

## *Hox.c*

```
/*************************************************************************
        This file contains the functions which implement the
        reaction channels for the  RA/Hox study; it also contains
        the function required to implement the diffusion components
        of Retinoic Acid.

        Note that the a_mu and d_mu values are updated during these
        function calls.  No updating of these quantities is done in
        the main program.


*************************************************************************/
#include <math.h>
#include <string.h>
#include <stdio.h>
#include <stdlib.h>
#include "Hox.h"
#include "UpdateAmu.h"
```

```
#include "ranlib.h"

int
update_cmu0(CELL *c, float T)
{
        int flag = 0;

        if(c->num != 0) {
                fprintf(stderr,"Error in update_cmu0:");
                fprintf(stderr," Trying to update in cell %d",c->num);
                goto cleanup;
        }
        c->a_mu[makeRA] = C_mu[makeRA]*T*exp(-.005*T);
        flag = 1;

cleanup:
        return flag;
}

int
MakeRA(CELL * c) /*   *-> ra a_mu[0] */
{
        int flag = 0;
        int affected = 3;
        int todo[3] = {bindRXR,bindRAR,decayRA};
        int i;
        c->ra += 1;
        c->d_ra = D_ra*(c->ra);

        for(i = 0; i < affected; i++) {
                (Update)[todo[i]](c);
        }
        flag = 1;
        return flag;
}

int
BindRAR(CELL *c)    /* ra + rar -> brar a_mu[1] */
{
        int flag = 0;
        int affected = 7;
        int i;
        int todo[7]= {bindRAR,decayRA,bindRXR,decayRAR,unbindBRAR,
                                decayBRAR,dimerize};

        if(c->ra < 1 || c->rar < 1) {
```

```
                fprintf(stderr,"Error in BindRAR:");
                fprintf(stderr," Cell %d has %d ra and %d rar\n",c->num,c->ra,c->rar);
                goto cleanup;
        }

/* Change the relevant quantities */
        c->ra -= 1;
        c->rar -= 1;
        c->brar += 1;

        for(i = 0; i < affected; i++) {
                (Update)[todo[i]](c);
        }

        c->d_ra = D_ra*(c->ra);
        flag = 1;
cleanup:
        return flag;
}

int
DecayRA(CELL * c)               /*   ra->null  a_mu[2] */
{
        int flag = 0;
        int affected = 3;
        int todo[3] = {bindRAR,bindRXR,decayRA};
        int i;
        if(c->ra < 1) {
                fprintf(stderr,"Error in DecayRA:");
                fprintf(stderr," Cell %d has %d RA\n", c->num,c->ra);
                goto cleanup;
        }
        c->ra -= 1;
        for(i = 0; i < affected; i++) {
                (Update)[todo[i]](c);
        }
        c->d_ra = D_ra*(c->ra);
        flag = 1;

cleanup:
        return flag;
}

int
MakeRAR(CELL * c) /*    *-> rar a_mu[3] */
{
```

```
	int affected = 2;
	int todo[2] = {bindRAR,decayRAR};
	int flag = 0;
	int i;
	c->rar += 1;
	for(i = 0; i < affected; i++) {
		(Update)[todo[i]](c);
	}
	flag = 1;
	return flag;
}

int
DecayRAR(CELL * c)			/*   rar->null a_mu[4]*/
{
	int affected = 2;
	int todo[2] = {bindRAR,decayRAR};
	int flag = 0;
	int i;
	if(c->rar < 1) {
		fprintf(stderr,"Error in DecayRAR:");
		fprintf(stderr," Cell %d has %d RAR\n", c->num,c->rar);
		goto cleanup;
	}
	c->rar -= 1;
	for(i = 0; i < affected; i++) {
		(Update)[todo[i]](c);
	}
	flag = 1;

cleanup:
	return flag;
}

int
UnbindBRAR(CELL *c)			/* brar-> ra + rar -> a_mu[5] */
{
	int flag = 0;
	int affected = 7;
	int i;
	int todo[7] = {bindRAR,bindRXR,decayRA,decayRAR,unbindBRAR,
				decayBRAR,dimerize};
	if(c->brar < 1) {
		fprintf(stderr,"Error in UnbindBRA:");
		fprintf(stderr," Cell %d has %d brar.\n",c->num,c->brar);
		goto cleanup;
```

```
        }
/* Change the relevant quantities */
        c->ra += 1;
        c->rar += 1;
        c->brar -= 1;
        for(i = 0; i < affected; i++) {
                (Update)[todo[i]](c);
        }

        c->d_ra = D_ra*(c->ra);

        flag = 1;
cleanup:
        return flag;
}

int
DecayBRAR(CELL *c)                  /* brar-> null  a_mu[6] */
{
        int flag = 0;
        int affected = 3;
        int todo[3] = {unbindBRAR,decayBRAR,dimerize};
        int i;
        if(c->brar < 1) {
                fprintf(stderr,"Error in DecayBRA:");
                fprintf(stderr," Cell %d has %d brar.\n",c->num,c->brar);
                goto cleanup;
        }
/* Change the relevant quantities */
        c->brar -= 1;
        for(i = 0; i < affected; i++) {
                (Update)[todo[i]](c);
        }

        flag = 1;
        cleanup:

        return flag;
}

int
ActivateA1(CELL * c)                /*  brar + A1 -> actA1  a_mu[7];*/
{
        int flag = 0;
        int affected = 7;
        int i;
```

```
        int todo[7] = {unDimerize,decayDimer,activateA1,transcribeA1,
                activateB1,unActivateA1,repressB1};

        if(c->A1 < 1 || c->dimer < 1 || c->actA1 > 1) {
                fprintf(stderr,"Error in ActivateA1:");
                fprintf(stderr," Cell %d has %d A1, %d actA1 and %d dimer.\n",
      c->num,c->A1,c->actA1,c->dimer);
                goto cleanup;
        }
        c->A1 = 0;
        c->actA1 = 1;
        c->dimer -= 1;
        for(i = 0; i < affected; i++) {
                (Update)[todo[i]](c);
        }

        flag = 1;

cleanup:
        return flag;
}

int
UnActivateA1(CELL * c)                  /* actA1 -> A1,brar  a_mu[8]*/
{
        int flag = 0;
        int affected = 8;
        int todo[8] = {unDimerize,decayDimer,decayBRAR,activateA1,unActivateA1,
                        transcribeA1,activateB1,repressB1};
        int i;
        if(c->actA1 < 1) {
                fprintf(stderr,"Error in UnActivateA1:");
                fprintf(stderr," Cell %d has %d actA1\n", c->num,c->actA1);
                goto cleanup;
        }
        c->A1 = 1;
        c->actA1 = 0;
        c->dimer += 1;
        for(i = 0; i < affected; i++) {
                (Update)[todo[i]](c);
        }

        flag = 1;
        cleanup:
        return flag;
```

```
}

int
TranscribeA1(CELL * c)                  /* actA1 -> mA1  a_mu[9]*/
{

        int flag = 0;
        int affected = 9;
        int todo[9] = {unDimerize,decayDimer,activateA1,unActivateA1,
                        transcribeA1,translateA1,activateB1,repressB1,
                        decaymA1};
        int i;

        if(c->A1 < 1) {
                fprintf(stderr,"Error in TranscribeA1:");
                fprintf(stderr," Cell %d has %d A1\n", c->num,c->A1);
                goto cleanup;
        }

        c->mA1 += 1;
        for(i = 0; i < affected; i++) {
                (Update)[todo[i]](c);
        }

        flag = 1;
        cleanup:
        return flag;
}

int
DecaymA1(CELL * c)                      /* mA1 -> null  a_mu[10]*/
{

        int flag = 0;
        int affected = 3;
        int todo[3] = {transcribeA1,translateA1,decaymA1};
        int i;

        if(c->mA1 < 1) {
                fprintf(stderr,"Error in DecaymA1:");
                fprintf(stderr," Cell %d has %d mA1\n", c->num,c->mA1);
                goto cleanup;
        }

        c->mA1 -= 1;
        for(i = 0; i < affected; i++) {
```

```
                (Update)[todo[i]](c);
        }

        flag = 1;
cleanup:
        return flag;
}

int
TranslateA1(CELL * c)                   /* mA1 -> a1 a_mu[11]*/
{
        int flag = 0;
        int affected = 4;
        int todo[4] = {translateA1,decaya1,decaymA1,complexa1};
        int i;
        if(c->mA1 < 1) {
                fprintf(stderr,"Error in TranslateA1:");
                fprintf(stderr," Cell %d has %d mA1\n", c->num,c->mA1);
                goto cleanup;
        }

        c->a1 += 1;
        for(i = 0; i < affected; i++) {
                (Update)[todo[i]](c);
        }

        flag = 1;
cleanup:
                return flag;
}

int
Decaya1(CELL *c)                /* a1 -> null  a_mu[12]*/
{
        int flag = 0;
        int affected = 2;
        int i;
        int todo[2] = {decaya1,complexa1};
        if(c->a1 < 1) {
                fprintf(stderr,"Error in Decaya1:");
                fprintf(stderr," Cell %d has %d a1.\n",c->num,c->a1);
                goto cleanup;
        }
        c->a1 -= 1;
        for(i = 0; i < affected; i++) {
                (Update)[todo[i]](c);
```

```
	}

	flag = 1;
cleanup:
	return flag;
}


int
ActivateB1(CELL * c)			/*  brar + B1 -> actB1 a_mu[13]*/
{
	int i;
	int flag = 0;
	int affected = 11;
	int todo[11] = {unDimerize,decayDimer,activateB1,unActivateB1,
				activateA1,transcribeB1,superActivateB1,repressB1,
				autoActivateB1,activateB2,unActivateB2};

	if(c->B1 < 1 || c->dimer < 1) {
		fprintf(stderr,"Error in ActivateB1:");
		fprintf(stderr," Cell %d has %d B1 and %d dimer\n",
		c->num,c->B1,c->dimer);
		goto cleanup;
	}
	c->B1 = 0;
	c->dimer -= 1;
	c->actB1 = 1;

/* Occasionally, activate the b2 gene */
	i =  (int) ignbin((long) 10, .25);
	if(c->B2 == 1 && i <= 2) {
		c->actB2 = 1;
		c->B2 = 0;
	}
	for(i = 0; i < affected; i++) {
		(Update)[todo[i]](c);
	}

	flag = 1;

cleanup:
	return flag;
}

int
UnActivateB1(CELL * c)			/*  actB1 -> B1,brar a_mu[14]*/
```

```
{
        int flag = 0;
        int affected = 9;
        int i;
        int todo[9] = {unDimerize,decayDimer,activateB1,unActivateB1,
                        transcribeB1,superActivateB1,repressB1,autoActivateB1,
                                activateA1};
        if(c->actB1 < 1) {
                fprintf(stderr,"Error in UnactivateB1:");
                fprintf(stderr," Cell %d has %d actB1.\n",
               c->num,c->actB1);
                goto cleanup;
        }
        c->B1 = 1;
        c->dimer += 1;
        c->actB1 = 0;
        for(i = 0; i < affected; i++) {
                (Update)[todo[i]](c);
        }

        flag = 1;

cleanup:
        return flag;
}

int
TranscribeB1(CELL * c)                  /*  superactB1 -> mB1 a_mu[15]*/
{

        int flag = 0;
        int affected = 11;
        int i;
        int todo[11] = {unDimerize,decayDimer,activateB1,unActivateB1,
                        transcribeB1,superActivateB1,translateB1,repressB1,
                        autoActivateB1,decaymB1,activateA1};
        if(c->actB1 < 1) {
                fprintf(stderr,"Error in TranscribeB1:");
                fprintf(stderr," Cell %d has %d actB1.\n", c->num,c->actB1);
                goto cleanup;
        }
        c->mB1 += 1;

        for(i = 0; i < affected; i++) {
                (Update)[todo[i]](c);
        }
```

```
	flag = 1;

	cleanup:
	return flag;
	}

int
DecaymB1(CELL * c)			/* mA1 -> null  a_mu[16]*/
{

	int flag = 0;
	int affected = 3;
	int i;
	int todo[3] = {transcribeB1,translateB1,decaymB1};

	if(c->mB1 < 1) {
		fprintf(stderr,"Error in DecaymB1:");
		fprintf(stderr," Cell %d has %d mA1\n", c->num,c->mB1);
		goto cleanup;
	}

	c->mB1 -= 1;
	for(i = 0; i < affected; i++) {
		(Update)[todo[i]](c);
	}

	flag = 1;
cleanup:
	return flag;
}

int
TranslateB1(CELL * c)			/*  mB1 -> b1 a_mu[17]*/
{
	int flag = 0;
	int affected = 5;
	int i;
	int todo[5] = {translateB1,complexb1,decaymB1,decayb1,repressKrox};
	if(c->mB1 < 1) {
		fprintf(stderr,"Error in TranslateB1:");
		fprintf(stderr," Cell %d has %d mB1\n", c->num,c->mB1);
		goto cleanup;
	}
	c->b1 += 1;
	for(i = 0; i < affected; i++) {
```

```
                (Update)[todo[i]](c);
        }

        flag = 1;

cleanup:
        return flag;
}


int
SuperActivateB1(CELL *c)            /* actB1+a1 ->superactB1 a_mu[18] */
{
        int flag = 0;
        int affected = 9;
        int i;
        int todo[9] = {unActivateB1,decaya1,superActivateB1,transcribeB1,
                                unSuperActivateB1,transcribeSuperB1,autoActivateB1,
                                        decaya1Complex,unComplexa1};
        if(c->actB1 < 1 || c->a1plex < 1) {
                fprintf(stderr,"Error in SuperActivateB1:");
                fprintf(stderr," Cell %d has %d actB1 and %d a1plex\n",
                        c->num,c->actB1,c->a1plex);
                goto cleanup;
        }
        c->actB1 = 0;
        c->superactB1 = 1;
        c->a1plex -= 1;

        for(i = 0; i < affected; i++) {
                (Update)[todo[i]](c);
        }

        flag = 1;

cleanup:
        return flag;
}

int
UnSuperActivateB1(CELL *c)                  /*  superactB1 -> actB1, a1 a_mu[19]*/
{
        int flag = 0;
        int affected = 13;
        int i;
        int todo[13] = {decaya1,unActivateB1,superActivateB1,transcribeB1,
```

```
                                unSuperActivateB1,transcribeSuperB1,activateA1,

        decayDimer,unDimerize,repressB1,decaya1Complex,
                                        unComplexa1,repressB1};

        if(c->superactB1 < 1) {
                fprintf(stderr,"Error in UnSuperActivateB1:");
                fprintf(stderr," Cell %d has %d superactB1\n",c->num,c->superactB1);
                goto cleanup;
        }
//      c->actB1 = 1;
//      c->superactB1 = 0;
//      c->a1plex += 1;

        c->B1 = 1;
        c->superactB1 = 0;
        c->actB1 = 0;
        c->a1plex += 1;
        c->dimer += 1;

        for(i = 0; i < affected; i++) {
                (Update)[todo[i]](c);
        }

        flag = 1;
cleanup:
        return flag;

}

int
TranscribeSuperB1(CELL * c)                     /*  superactB1 -> mB1 a_mu[20]*/
{
        int flag = 0;
        int affected = 14;
        int i;
        int todo[14] = {unDimerize,decayDimer,activateA1,activateB1,
                                superActivateB1,unSuperActivateB1,transcribeSuperB1,
                                translateB1,repressB1,autoActivateB1,decaymB1,
                                        repressKrox,unComplexa1,decayb1};
        if(c->superactB1 < 1) {
                fprintf(stderr,"Error in TranscribeSuperB1:");
                fprintf(stderr," Cell %d has %d actB1.\n", c->num,c->superactB1);
                goto cleanup;
        }
        c->mB1 += 2;            /* This should be changed to 5 or so */
```

```
	for(i = 0; i < affected; i++) {
		(Update)[todo[i]](c);
	}
	flag = 1;

cleanup:
	return flag;
}

int
RepressB1(CELL * c)			/*  B1+brar -> repB1 a_mu[21]*/
{
	int flag = 0;
	int affected = 7;
	int i;
	int todo[7] = {unDimerize,decayDimer,activateA1,activateB1,
			unRepressB1,autoActivateB1,repressB1};
	if(c->B1 < 1 || c->dimer < 1) {
		fprintf(stderr,"Error in RepressB1:");
		fprintf(stderr," Cell %d has %d B1 and %d dimer\n",
						c->num,c->B1,c->dimer);
		goto cleanup;
	}
	c->repB1 = 1;
	c->B1 = 0;
	c->dimer -= 1;

	for(i = 0; i < affected; i++) {
		(Update)[todo[i]](c);
	}

	flag = 1;

cleanup:
	return flag;
}

int
UnRepressB1(CELL * c)			/*  B1 -> repB1,brar  a_mu[22]*/
{
	int flag = 0;
	int affected = 7;
	int i;
	int todo[7] = {unDimerize,decayDimer,activateA1,activateB1,
			unRepressB1,autoActivateB1,repressB1};
	if(c->repB1 < 1) {
```

```
                fprintf(stderr,"Error in UnRepressB1:");
                fprintf(stderr," Cell %d has %d repB1\n",c->num,c->repB1);
                goto cleanup;
        }
        c->repB1 = 0;
        c->B1 = 1;
        c->dimer += 1;

        for(i = 0; i < affected; i++) {
                (Update)[todo[i]](c);
        }

        flag = 1;

cleanup:
        return flag;
}

int
AutoActivateB1(CELL * c)            /*  B1+b1 -> autoB1 a_mu[23]*/
{
        int flag = 0;
        int affected = 8;
        int i;
        int todo[8] = {activateB1,repressB1,transcribeAutoB1,unComplexb1,
                        decayb1Complex,autoActivateB1,unAutoActivateB1,activateB2};

        if(c->B1 < 1 || c->b1plex < 1) {
           fprintf(stderr,"Error in AutoActivateB1:");
           fprintf(stderr," Cell %d has %d B1 and %d b1plex\n",
                        c->num,c->B1,c->b1plex);
           goto cleanup;
        }
        c->B1 = 0;
        c->b1plex -= 1;
        c->autoB1 = 1;
        for(i = 0; i < affected; i++) {
                (Update)[todo[i]](c);
        }
        flag = 1;
cleanup:
        return flag;
}

int
UnAutoActivateB1(CELL *c)                   /*  autoB1 -> B1, b1 a_mu[24]*/
```

```
{
        int flag = 0;
        int affected = 8;
        int i;
        int todo[8] = {activateB1,repressB1,transcribeAutoB1,unComplexb1,
                decayb1Complex,autoActivateB1,unAutoActivateB1,activateB2};

        if(c->autoB1 < 1) {
                fprintf(stderr,"Error in UnAutoActivateB1:");
                fprintf(stderr," Cell %d has %d autoB1\n",c->num,c->autoB1);
                goto cleanup;
        }
        c->B1 = 1;
        c->b1plex += 1;
        c->autoB1 = 0;
        for(i = 0; i < affected; i++) {
                (Update)[todo[i]](c);
        }
        flag = 1;
cleanup:
        return flag;
}

int
TranscribeAutoB1(CELL * c)                      /*  autoB1 -> mB1 a_mu[25]*/
{
        int flag = 0;
        int affected = 8;
        int i;
        int todo[8] = {activateB1,superActivateB1,translateB1,
                        unAutoActivateB1,repressB1,autoActivateB1,
                    transcribeAutoB1,decaymB1};
        if(c->autoB1 < 1) {
                fprintf(stderr,"Error in TranscribeAutoSuperB1:");
                fprintf(stderr," Cell %d has %d autoB1.\n", c->num,c->autoB1);
                goto cleanup;
        }
        c->mB1 += 2;            /* This should be changed to 5 or so */
        for(i = 0; i < affected; i++) {
                (Update)[todo[i]](c);
        }
        flag = 1;

cleanup:
        return flag;
}
```

```
int
Decayb1(CELL *c)                /*  b1 -> null a_mu[26]*/
{
        int flag = 0;
        int affected = 3;
        int i;
        int todo[3] = {decayb1,complexb1,repressKrox};
        if(c->b1 < 1) {
                fprintf(stderr,"Error in Decayb1:");
                fprintf(stderr," Cell %d has %d b1.\n",c->num,c->b1);
                goto cleanup;
        }
        c->b1 -= 1;
        for(i = 0; i < affected; i++) {
                (Update)[todo[i]](c);
        }

        flag = 1;
cleanup:
        return flag;
}

int
ActivateB2(CELL *c)                     /* B2 + b1 -> actB2 a_mu[27]*/
{
        int flag = 0;
        int affected = 8;
        int r4 = (c->id == R4);
        int r5 = (c->id == R5);
        int i;
        int todo[8] = {autoActivateB1,decayb1,activateB2,unActivateB2,
          transcribeB2,decaykrox,unComplexb1,decayb1Complex};
        if(r4) {
                if(c->B2 < 1 || c->b1plex < 1) {
                        fprintf(stderr,"Error in ActivateB2:");
                        fprintf(stderr," Cell %d has %d B2 and %d b1plex\n",
                                c->num,c->B2,c->b1plex);
                        goto cleanup;
                }
                c->b1plex -= 1;
        } else if(r5) {
                if(c->B2 < 1 || c->krox < 1) {
                        fprintf(stderr,"Error in ActivateB2:");
                        fprintf(stderr," Cell %d has %d B2 and %d krox\n",
                                c->num,c->B2,c->krox);
```

```
                goto cleanup;
            }
            c->krox -= 1;
        } else {
            fprintf(stderr,"Error in ActivateB2: Called in R3\n");
            //goto cleanup;
        }

        c->B2 = 0;
        c->actB2 = 1;
        for(i = 0; i < affected; i++) {
            (Update)[todo[i]](c);
        }

        flag = 1;
cleanup:
        return flag;
}

int
UnActivateB2(CELL *c)                   /* actB2-> B2 ,b1 a_mu[28]*/
{
        int flag = 0;
        int affected = 8;
        int r4 = (c->id == R4);
        int r5 = (c->id == R5);
        int i;
        int todo[8] = {autoActivateB1,decayb1,activateB2,unActivateB2,

transcribeB2,decaykrox,unComplexb1,decayb1Complex};

        if(c->actB2 < 1) {
            fprintf(stderr,"Error in UnActivateB2:");
            fprintf(stderr," Cell %d has %d actB2 and %d b1plex\n",
          c->num,c->actB2,c->b1plex);
            goto cleanup;
        }
        if(r4) {
            c->b1plex += 1;
        } else if (r5) {
            c->krox += 1;
        } else {
            fprintf(stderr,"Error in UnActivateB2: Called in R3\n");
        }
        c->B2 = 1;
        c->actB2 = 0;
```

```
	for(i = 0; i < affected; i++) {
		(Update)[todo[i]](c);
	}

	flag = 1;
cleanup:
	return flag;
}

int
TranscribeB2(CELL * c)			/*  actB2 -> mB2 a_mu[29]*/
{

	int flag = 0;
	int affected = 9;
	int i;
	int todo[9] = {activateB2,unActivateB2,transcribeB2,translateB2,

	decaymB2,decaykrox,decayb1Complex,unComplexb1,
					autoActivateB1};

	if(c->actB2 < 1) {
		fprintf(stderr,"Error in TranscribeB2:");
		fprintf(stderr," Cell %d has %d actB2.\n", c->num,c->actB2);
		goto cleanup;
	}
	c->mB2 += 1;

	for(i = 0; i < affected; i++) {
		(Update)[todo[i]](c);
	}

	flag = 1;

cleanup:
	return flag;
}

int
DecaymB2(CELL * c)			/* mA1 -> null  a_mu[30]*/
{

	int flag = 0;
	int affected = 3;
	int i;
	int todo[3] = {transcribeB2,translateB2,decaymB2};
```

```
	if(c->mB2 < 1) {
		fprintf(stderr,"Error in DecaymB2:");
		fprintf(stderr," Cell %d has %d mA1\n", c->num,c->mB2);
		goto cleanup;
	}

	c->mB2 -= 1;
	for(i = 0; i < affected; i++) {
		(Update)[todo[i]](c);
	}

	flag = 1;
cleanup:
	return flag;
}

int
TranslateB2(CELL * c)			/*  mB2 -> b2 a_mu[31]*/
{
	int flag = 0;
	int affected = 3;
	int i;
	int todo[3] = {translateB2,decayb2,decaymB2};
	if(c->mB2 < 1) {
		fprintf(stderr,"Error in TranslateB2:");
		fprintf(stderr," Cell %d has %d mB2\n", c->num,c->mB2);
		goto cleanup;
	}
	c->b2 += 1;
	for(i = 0; i < affected; i++) {
		(Update)[todo[i]](c);
	}

	flag = 1;

cleanup:
	return flag;
}

int
Decayb2(CELL *c)		/*  b2 -> null a_mu[32]*/
{
	int flag = 0;
	int affected = 1;
	int i;
```

```
	int todo[1] = {decayb2};
	if(c->b2 < 1) {
		fprintf(stderr,"Error in Decayb2:");
		fprintf(stderr," Cell %d has %d b2.\n",c->num,c->b2);
		goto cleanup;
	}
	c->b2 -= 1;
	for(i = 0; i < affected; i++) {
		(Update)[todo[i]](c);
	}

	flag = 1;
cleanup:
	return flag;
}

/* a_mu[33] is Divide */

int
ActivateKrox(CELL *c)		/* a1 + Krox -> actKrox a_mu[34] */
{
	int flag = 0;
	int affected = 4;
	int i;
	int todo[4] = {activateKrox,unActivateKrox,transcribeKrox,repressKrox};
	if(c->Krox < 1) {
		fprintf(stderr,"Error in ActivateKrox:");
		fprintf(stderr," Cell %d has %d Krox.\n",
			c->num,c->Krox);
		goto cleanup;
	}
	c->Krox = 0;
	c->actKrox = 1;
	for(i = 0; i <  affected; i++) {
		(Update)[todo[i]](c);
	}

	flag = 1;
cleanup:
	return flag;
}

int
UnActivateKrox(CELL *c)
{
	int flag = 0;
```

```
	int affected = 4;
	int i;
	int todo[4] = {activateKrox,unActivateKrox,transcribeKrox,repressKrox};
	if(c->actKrox < 1) {
		fprintf(stderr,"Error in UnActivateKrox:");
		fprintf(stderr," Cell %d has %d Krox\n", c->num,c->Krox);
		goto cleanup;
	}
	c->Krox = 1;
	c->actKrox = 0;
	for(i = 0; i <  affected; i++) {
		(Update)[todo[i]](c);
	}

	flag = 1;
cleanup:
	return flag;
}

int
TranscribeKrox(CELL *c)
{
	int flag = 0;
	int affected = 6;
	int todo[6] = {activateKrox,unActivateKrox,transcribeKrox,repressKrox,
					translateKrox,decaymKrox};
	int i;

	if(c->actKrox < 1) {
		fprintf(stderr,"Error in TranscribeKrox:");
		fprintf(stderr," Cell %d has %d actKrox\n", c->num,c->actKrox);
		goto cleanup;
	}
	c->mKrox += 1;
	for(i = 0; i < affected; i++) {
		(Update)[todo[i]](c);
	}

	flag = 1;
	cleanup:
	return flag;
}

int
DecaymKrox(CELL *c)
{
```

```
	int flag = 0;
	int affected = 3;
	int todo[3] = {transcribeKrox,translateKrox,decaymKrox};
	int i;

	if(c->mKrox < 1) {
		fprintf(stderr,"Error in DecaymKrox:");
		fprintf(stderr," Cell %d has %d mKrox\n", c->num,c->mKrox);
		goto cleanup;
	}

	c->mKrox -= 1;
	for(i = 0; i < affected; i++) {
		(Update)[todo[i]](c);
	}

	flag = 1;
cleanup:
	return flag;
}

int
RepressKrox(CELL *c)
{
	int flag = 0;
	int affected = 10;
	int todo[10] = {decaya1,superActivateB1,activateKrox,unRepressKrox,

	unActivateKrox,transcribeKrox,repressKrox,complexb1,
					complexa1,decayb1};
	int i;

	if(c->Krox < 1 || (c->b1 < 1 && c->a1 < 1)) {
		fprintf(stderr,"Error in RepressKrox:");
		fprintf(stderr," Cell %d (%d) has %d Krox,%d a1 and %d b1\n",
					c->num,c->id+3,c->Krox,c->a1,c->b1);
		goto cleanup;
	}
	c->repKrox = 1;
	c->Krox = 0;

	if(c->a1 > c->b1) {
		c->a1 -= 1;
	   c->krep = A;
	} else {
		c->b1 -= 1;
```

```
            c->krep = B;
        }

        for(i = 0; i < affected; i++) {
                (Update)[todo[i]](c);
        }

        flag = 1;

cleanup:
        return flag;
}

int
UnRepressKrox(CELL *c)
{
        int flag = 0;
        int affected = 10;
        int i;
        int todo[10] = {decaya1,superActivateB1,activateKrox,unRepressKrox,

        unActivateKrox,transcribeKrox,repressKrox,complexb1,
                                        decayb1,complexa1};

        if(c->repKrox < 1) {
                fprintf(stderr,"Error in UnRepressKrox:");
                fprintf(stderr," Cell %d has %d Krox and %d a1\n",
                                                c->num,c->Krox,c->a1);
                goto cleanup;
        }
        c->repKrox = 0;
        c->Krox = 1;
        if(c->krep == A) {
                c->a1 += 1;
        } else {
                c->b1 += 1;
        }

        for(i = 0; i < affected; i++) {
                (Update)[todo[i]](c);
        }

        flag = 1;

cleanup:
        return flag;
```

```
}

int
TranslateKrox(CELL *c)
{
	int flag = 0;
	int affected = 4;
	int i;
	int todo[4] = {translateKrox,decaykrox,decaymKrox,activateB2};
	if(c->mKrox < 1) {
		fprintf(stderr,"Error in TranslateKrox:");
		fprintf(stderr," Cell %d has %d mKrox\n", c->num,c->mKrox);
		goto cleanup;
	}
	c->krox += 1;
	for(i = 0; i < affected; i++) {
		(Update)[todo[i]](c);
	}

	flag = 1;

cleanup:
	return flag;
}

int
Decaykrox(CELL *c)
{
	int flag = 0;
	int affected = 2;
	int i;
	int todo[2] = {decaykrox,activateB2};
	if(c->krox < 1) {
		fprintf(stderr,"Error in DecayKrox:");
		fprintf(stderr," Cell %d has %d krox.\n",c->num,c->krox);
		goto cleanup;
	}
	c->krox -= 1;
	for(i = 0; i < affected; i++) {
		(Update)[todo[i]](c);
	}

	flag = 1;
cleanup:
	return flag;
}
```

```
int
BindRXR(CELL *c)      /* a_mu[42] */
{
        int flag = 0;
        int affected = 7;
        int i;
        int todo[7]= {bindRAR,decayRA,decayRXR,bindRXR,unbindBRXR,
                                decayBRXR,dimerize};
        if(c->ra < 1 || c->rxr < 1) {
                fprintf(stderr,"Error in BindRXR:");
                fprintf(stderr," Cell %d has %d ra and %d rxr\n",c->num,c->ra,c->rxr);
                goto cleanup;
        }

/* Change the relevant quantities */
        c->ra -= 1;
        c->rxr -= 1;
        c->brxr += 1;

        for(i = 0; i < affected; i++) {
                (Update)[todo[i]](c);
        }

        c->d_ra = D_ra*(c->ra);
        flag = 1;
cleanup:
        return flag;

}

int
MakeRXR(CELL *c)  /* a_mu[43] */
{
        int affected = 2;
        int todo[2] = {bindRXR,decayRXR};
        int flag = 0;
        int i;
        c->rxr += 1;
        for(i = 0; i < affected; i++) {
                (Update)[todo[i]](c);
        }
        flag = 1;
        return flag;
}
```

```
int
DecayRXR(CELL *c)     /* a_mu[44] */
{
        int affected = 2;
        int todo[2] = {bindRXR,decayRXR};
        int flag = 0;
        int i;
        if(c->rxr < 1) {
                fprintf(stderr,"Error in DecayRXR:");
                fprintf(stderr," Cell %d has %d RXR\n", c->num,c->rxr);
                goto cleanup;
        }
        c->rxr -= 1;
        for(i = 0; i < affected; i++) {
                (Update)[todo[i]](c);
        }
        flag = 1;

cleanup:
        return flag;

}

int
UnbindBRXR(CELL *c)   /* a_mu[45] */
{
        int flag = 0;
        int affected = 7;
        int i;
        int todo[7]= {bindRAR,decayRA,decayRXR,bindRXR,unbindBRXR,
                                decayBRXR,dimerize};
        if(c->brxr < 1) {
                fprintf(stderr,"Error in UnbindBRXR:");
                fprintf(stderr," Cell %d has %d brxr.\n",c->num,c->brxr);
                goto cleanup;
        }
/* Change the relevant quantities */
        c->ra += 1;
        c->rxr += 1;
        c->brxr -= 1;
        for(i = 0; i < affected; i++) {
                (Update)[todo[i]](c);
        }

        c->d_ra = D_ra*(c->ra);
```

```
	flag = 1;
cleanup:
	return flag;

}

int
DecayBRXR(CELL *c)    /* a_mu[46] */
{
	int affected = 3;
	int todo[3] = {unbindBRXR,decayBRXR,dimerize};
	int flag = 0;
	int i;
	if(c->brxr < 1) {
		fprintf(stderr,"Error in DecayBRXR:");
		fprintf(stderr," Cell %d has %d brxr.\n",c->num,c->brxr);
		goto cleanup;
	}
/* Change the relevant quantities */
	c->brxr -= 1;
	for(i = 0; i < affected; i++) {
		(Update)[todo[i]](c);
	}

	flag = 1;
	cleanup:

	return flag;

}

int
Dimerize(CELL *c)     /* a_mu[47] */
{
	int flag = 0;
	int affected = 11;
	int i;
	int todo[11]= {unbindBRAR,decayBRAR,unbindBRXR,decayBRXR,
			activateA1,activateB1,repressB1,decayDimer,
			dimerize,unDimerize,transcribeA1};

	if(c->brar < 1 || c->brxr < 1) {
		fprintf(stderr,"Error in Dimerize:");
		fprintf(stderr," Cell %d has %d brar and %d brxr\n",
			c->num,c->brar,c->brxr);
```

```
                goto cleanup;
        }

/* Change the relevant quantities */
        c->brar -= 1;
        c->brxr -= 1;
        c->dimer += 1;

        for(i = 0; i < affected; i++) {
                (Update)[todo[i]](c);
        }

        flag = 1;
cleanup:
        return flag;
}

int
UnDimerize(CELL *c)   /* a_mu[48] */
{
        int flag = 0;
        int affected = 11;
        int i;
        int todo[11]= {unbindBRAR,decayBRAR,unbindBRXR,decayBRXR,dimerize,
                        activateA1,activateB1,repressB1,decayDimer,
                        unDimerize,transcribeA1};


        if(c->dimer < 1) {
                fprintf(stderr,"Error in UnDimerize:");
                fprintf(stderr," Cell %d has %d dimers\n", c->num,c->dimer);
                goto cleanup;
        }

/* Change the relevant quantities */
        c->brar += 1;
        c->brxr += 1;
        c->dimer -= 1;

        for(i = 0; i < affected; i++) {
                (Update)[todo[i]](c);
        }

        flag = 1;
cleanup:
        return flag;
```

```
}

int
DecayDimer(CELL *c)    /* a_mu[49] */
{
        int flag = 0;
        int affected = 6;
        int todo[6]= {activateA1,activateB1,repressB1,decayDimer,unDimerize,
                                transcribeA1};
        int i;

        if(c->dimer < 1) {
                fprintf(stderr,"Error in DecayDimer:");
                fprintf(stderr," Cell %d has %d dimers\n", c->num,c->dimer);
                goto cleanup;
        }

/* Change the relevant quantities */
        c->dimer -= 1;

        for(i = 0; i < affected; i++) {
                (Update)[todo[i]](c);
        }

        flag = 1;
cleanup:
        return flag;

}

int
Complexa1(CELL *c)      /* a_mu[50] */
{
        int flag = 0;
        int affected = 8;
        int i;
        int todo[8]= {complexb1,superActivateB1,decaya1Complex,
                                decayComplex,decaya1,complexa1,repressKrox,
                                unComplexa1};

        if(c->a1 < 1 || c->plex < 1) {
                fprintf(stderr,"Error in Complexa1:");
                fprintf(stderr," Cell %d has %d a1 and %d complexes\n"
                        ,c->num,c->a1,c->plex);
                goto cleanup;
```

```
        }

/* Change the relevant quantities */
        c->a1 -= 1;
        c->plex -= 1;
        c->a1plex += 1;

        for(i = 0; i < affected; i++) {
                (Update)[todo[i]](c);
        }

        flag = 1;
cleanup:
        return flag;
}

int
Uncomplexa1(CELL *c)    /* a_mu[51] */
{
        int flag = 0;
        int affected = 8;
        int i;
        int todo[8]= {complexb1,superActivateB1,decaya1Complex,
                                decayComplex,decaya1,complexa1,repressKrox,
                                unComplexa1};

        if(c->a1plex < 1) {
                fprintf(stderr,"Error in UnComplexa1:");
                fprintf(stderr," Cell %d has %d a1plex\n" ,c->num,c->a1plex);
                goto cleanup;
        }

/* Change the relevant quantities */
        c->a1 += 1;
        c->plex += 1;
        c->a1plex -= 1;

        for(i = 0; i < affected; i++) {
                (Update)[todo[i]](c);
        }

        flag = 1;
cleanup:
        return flag;
}
```

```
int
Decaya1Complex(CELL *c) /* a_mu[52] */
{
        int flag = 0;
        int affected = 3;
        int i;
        int todo[3]= {superActivateB1,decaya1Complex,unComplexa1};

        if(c->a1plex < 1) {
                fprintf(stderr,"Error in Decaya1Complex:");
                fprintf(stderr," Cell %d has %d a1plex\n" ,c->num,c->a1plex);
                goto cleanup;
        }

/* Change the relevant quantities */
        c->a1plex -= 1;

        for(i = 0; i < affected; i++) {
                (Update)[todo[i]](c);
        }

        flag = 1;
cleanup:
        return flag;
}

int
Complexb1(CELL *c)      /* a_mu[53] */
{
        int flag = 0;
        int affected = 9;
        int i;
        int todo[9]= {complexb1,autoActivateB1,activateB2,complexa1,
                                unComplexb1,decayb1Complex,decayb1,decayComplex,
                                repressKrox};

        if(c->b1 < 1 || c->plex < 1) {
                fprintf(stderr,"Error in Complexb1:");
                fprintf(stderr," Cell %d has %d b1 and %d complexes\n"
                        ,c->num,c->b1,c->plex);
                goto cleanup;
        }

/* Change the relevant quantities */
        c->b1 -= 1;
        c->plex -= 1;
```

```
	c->b1plex += 1;

	for(i = 0; i < affected; i++) {
		(Update)[todo[i]](c);
	}

	flag = 1;
cleanup:
	return flag;
}

int
Uncomplexb1(CELL *c)	/* a_mu[54] */
{
	int flag = 0;
	int affected = 9;
	int i;
	int todo[9]= {complexb1,autoActivateB1,activateB2,complexa1,
				unComplexb1,decayb1Complex,decayb1,decayComplex,
				repressKrox};

	if(c->b1plex < 1) {
		fprintf(stderr,"Error in UnComplexb1:");
		fprintf(stderr," Cell %d has %d b1\n",c->num,c->b1plex);
		goto cleanup;
	}

/* Change the relevant quantities */
	c->b1 += 1;
	c->plex += 1;
	c->b1plex -= 1;

	for(i = 0; i < affected; i++) {
		(Update)[todo[i]](c);
	}

	flag = 1;
cleanup:
	return flag;
}

int
Decayb1Complex(CELL *c)  /* a_mu[55] */
{
	int flag = 0;
	int affected = 4;
```

```
	int i;
	int todo[4]= {autoActivateB1,activateB2,unComplexb1,decayb1Complex};

	if(c->b1plex < 1) {
		fprintf(stderr,"Error in Decayb1Complex:");
		fprintf(stderr," Cell %d has %d b1\n",c->num,c->b1plex);
		goto cleanup;
	}

/* Change the relevant quantities */
	c->b1plex -= 1;

	for(i = 0; i < affected; i++) {
		(Update)[todo[i]](c);
	}

	flag = 1;
cleanup:
	return flag;
}

int
MakeComplex(CELL *c)
{
	int affected = 3;
	int todo[3] = {complexa1,complexb1,decayComplex};
	int flag = 0;
	int i;
	c->plex += 1;
	for(i = 0; i < affected; i++) {
		(Update)[todo[i]](c);
	}
	flag = 1;
	return flag;
}

int
DecayComplex(CELL *c)
{
	int affected = 3;
	int todo[3] = {complexa1,complexb1,decayComplex};
	int flag = 0;
	int i;
	c->plex -= 1;
	for(i = 0; i < affected; i++) {
		(Update)[todo[i]](c);
```

```
        }
        flag = 1;
        return flag;

}


int
RA_Diffusion(CELL * c)   /*  RA diffusion; Source located at xi = 0 */
{
        int ra;
        int flag = 0;

        if((c->next != (CELL*) NULL) && (c->ra > 0)) {
                ra = ((c+1)->ra += 1);
                (c+1)->d_ra = D_ra*ra;
                (c+1)->a_mu[bindRAR] = C_mu[bindRAR]*ra*(c+1)->rar;
                (c+1)->a_mu[bindRXR] = C_mu[bindRXR]*ra*(c+1)->rxr;
                (c+1)->a_mu[decayRA] = C_mu[decayRA]*ra;
                ra = (c->ra -= 1);
                c->d_ra = D_ra*ra;
                c->a_mu[bindRAR] = C_mu[bindRAR]*ra*c->rar;
                c->a_mu[bindRXR] = C_mu[bindRXR]*ra*c->rxr;
                c->a_mu[decayRA] = C_mu[decayRA]*ra;
                flag = 1;
        }
        else {
                printf("Can't diffuse in cell %d, have %d ra.\n",c->num,c->ra);
        }
        return flag;
}
```

## *UpdateAmu.h*

```
#include "Hox.h"

#ifndef __UPDATEAMU_H
#define __UPDATEAMU_H

void UpdateMakeRA(CELL *);
void UpdateBindRAR(CELL *);
void UpdateDecayRA(CELL *);
void UpdateMakeRAR(CELL *);
void UpdateDecayRAR(CELL *);
void UpdateUnBindBRAR(CELL *);
```

```
void UpdateDecayBRAR(CELL *);
void UpdateActivateA1(CELL *);
void UpdateUnActivateA1(CELL *);
void UpdateTranscribeA1(CELL *);
void UpdateDecaymA1(CELL *);
void UpdateTranslateA1(CELL *);
void UpdateDecaya1(CELL *);
void UpdateActivateB1(CELL *);
void UpdateUnActivateB1(CELL *);
void UpdateTranscribeB1(CELL *);
void UpdateDecaymB1(CELL *);
void UpdateTranslateB1(CELL *);
void UpdateSuperActivateB1(CELL *);
void UpdateUnSuperActivateB1(CELL *);
void UpdateTranscribeSuperB1(CELL *);
void UpdateRepressB1(CELL *);
void UpdateUnRepressB1(CELL *);
void UpdateAutoActivateB1(CELL *);
void UpdateUnAutoActivateB1(CELL *);
void UpdateTranscribeAutoB1(CELL *);
void UpdateDecayb1(CELL *);
void UpdateActivateB2(CELL *);
void UpdateUnActivatedB2(CELL *);
void UpdateTranscribeB2(CELL *);
void UpdateDecaymB2(CELL *);
void UpdateTranslateB2(CELL *);
void UpdateDecayb2(CELL *);
void UpdateDivide(CELL *);
void UpdateActivateKrox(CELL *);
void UpdateUnActivateKrox(CELL *);
void UpdateTranscribeKrox(CELL *);
void UpdateDecaymKrox(CELL *);
void UpdateRepressKrox(CELL *);
void UpdateUnRepressKrox(CELL *);
void UpdateTranslateKrox(CELL *);
void UpdateDecaykrox(CELL *);
void UpdateBindRXR(CELL *);
void UpdateMakeRXR(CELL *);
void UpdateDecayRXR(CELL *);
void UpdateUnbindBRXR(CELL *);
void UpdateDecayBRXR(CELL *);
void UpdateDimerize(CELL *);
void UpdateUnDimerize(CELL *);
void UpdateDecayDimer(CELL *);
void UpdateComplexa1(CELL *);
void UpdateUnComplexa1(CELL *);
```

```
void UpdateDecaya1Complex(CELL *);
void UpdateComplexb1(CELL *);
void UpdateUnComplexb1(CELL *);
void UpdateDecayb1Complex(CELL *);
void UpdateMakeComplex(CELL *);
void UpdateDecayComplex(CELL *);

typedef  void (*UpdateAmu)();

static UpdateAmu       Update[NUM_FUNCS] = {
UpdateMakeRA,
UpdateBindRAR,
UpdateDecayRA,
UpdateMakeRAR,
UpdateDecayRAR,
UpdateUnBindBRAR,
UpdateDecayBRAR,
UpdateActivateA1,
UpdateUnActivateA1,
UpdateTranscribeA1,
UpdateDecaymA1,
UpdateTranslateA1,
UpdateDecaya1,
UpdateActivateB1,
UpdateUnActivateB1,
UpdateTranscribeB1,
UpdateDecaymB1,
UpdateTranslateB1,
UpdateSuperActivateB1,
UpdateUnSuperActivateB1,
UpdateTranscribeSuperB1,
UpdateRepressB1,
UpdateUnRepressB1,
UpdateAutoActivateB1,
UpdateUnAutoActivateB1,
UpdateTranscribeAutoB1,
UpdateDecayb1,
UpdateActivateB2,
UpdateUnActivatedB2,
UpdateTranscribeB2,
UpdateDecaymB2,
UpdateTranslateB2,
UpdateDecayb2,
UpdateDivide,
UpdateActivateKrox,
UpdateUnActivateKrox,
```

```
UpdateTranscribeKrox,
UpdateDecaymKrox,
UpdateRepressKrox,
UpdateUnRepressKrox,
UpdateTranslateKrox,
UpdateDecaykrox,
UpdateBindRXR,
UpdateMakeRXR,
UpdateDecayRXR,
UpdateUnbindBRXR,
UpdateDecayBRXR,
UpdateDimerize,
UpdateUnDimerize,
UpdateDecayDimer,
UpdateComplexa1,
UpdateUnComplexa1,
UpdateDecaya1Complex,
UpdateComplexb1,
UpdateUnComplexb1,
UpdateDecayb1Complex,
UpdateMakeComplex,
UpdateDecayComplex};

#endif
```

### *UpdateAmu.c*

```
#include <math.h>
#include <stdlib.h>
#include <stdio.h>
#include "Hox.h"
#include "header.h"
#include "UpdateAmu.h"


void                                    /* 0 */
UpdateMakeRA(CELL *c)
{

}

void                                        /* 1 */
UpdateBindRAR(CELL *c)
{
```

```
        c->a_mu[bindRAR] = C_mu[bindRAR]*(c->ra)*(c->rar);
}

void                                    /* 2 */
UpdateDecayRA(CELL *c)
{
  c->a_mu[decayRA]   = C_mu[decayRA]*(c->ra);
}

void                                    /* 3 */
UpdateMakeRAR(CELL *c)
{

}

void
UpdateDecayRAR(CELL *c)         /* 4 */
{
   c->a_mu[decayRAR]  = C_mu[decayRAR]*(c->rar);
}

void
UpdateUnBindBRAR(CELL *c)       /* 5 */
{
        c->a_mu[unbindBRAR]  = C_mu[unbindBRAR]*(c->brar);
}

void
UpdateDecayBRAR(CELL *c)                /* 6 */
{
        c->a_mu[decayBRAR]  = C_mu[decayBRAR]*(c->brar);
}

void
UpdateActivateA1(CELL *c) /* 7 */
{
//    int dimer = c->dimer;
//    c->a_mu[activateA1]  = C_mu[activateA1]*pow((double)dimer,a1hill)/
//                        (K[1]+pow((double)dimer,a1hill))*(dimer)*(c->A1);
}

void
UpdateUnActivateA1(CELL *c)       /* 8 */
{
//    c->a_mu[unActivateA1] = C_mu[unActivateA1]*(c->actA1);
}
```

```
void
UpdateTranscribeA1(CELL *c) /* 9 */
{
//        c->a_mu[transcribeA1] = C_mu[transcribeA1]*(c->actA1);
/* This version is for the updated model */
          c->a_mu[transcribeA1] = C_mu[transcribeA1]*(c->dimer);
}

void
UpdateDecaymA1(CELL *c) /* 10 */
{
          c->a_mu[decaymA1] = C_mu[decaymA1]*(c->mA1);
}

void
UpdateTranslateA1(CELL *c)          /* 11 */
{
          c->a_mu[translateA1] = C_mu[translateA1]*(c->mA1);
}

void
UpdateDecaya1(CELL *c)                        /* 12 */
{
   c->a_mu[decaya1]  = C_mu[decaya1]*(c->a1);
}

void
UpdateActivateB1(CELL *c)           /* 13 */
{
   int dimer = c->dimer;
   c->a_mu[activateB1]  = C_mu[activateB1]*pow((double)dimer,b1hill)/
                     (K[2]+pow((double)dimer,b1hill))*dimer*c->B1;
}

void
UpdateUnActivateB1(CELL *c)                   /* 15 */
{
   c->a_mu[unActivateB1] = C_mu[unActivateB1]*(c->actB1);
}

void
UpdateTranscribeB1(CELL *c)                   /* 15 */
{
   c->a_mu[transcribeB1] = C_mu[transcribeB1]*(c->actB1);
```

```
}

void
UpdateDecaymB1(CELL *c)                    /* 16 */
{
        c->a_mu[decaymB1] = C_mu[decaymB1]*(c->mB1);
}

void
UpdateTranslateB1(CELL *c)                 /* 17 */
{
   c->a_mu[translateB1] = C_mu[translateB1]*(c->mB1);
}

void
UpdateSuperActivateB1(CELL *c)             /* 18 */
{
   int a1plex = c->a1plex;
   int id = (c->id == R4);
   c->a_mu[superActivateB1]  = id*C_mu[superActivateB1]*
                        pow((double)a1plex,a1hill)/
                        (K[3]+pow((double)a1plex,a1hill))* a1plex*(c->actB1);
}

void
UpdateUnSuperActivateB1(CELL *c)           /* 19 */
{
   c->a_mu[unSuperActivateB1] = C_mu[unSuperActivateB1]*(c->superactB1);
}

void
UpdateTranscribeSuperB1(CELL *c)           /* 20 */
{
   c->a_mu[transcribeSuperB1] = C_mu[transcribeSuperB1]*(c->superactB1);
}

void
UpdateRepressB1(CELL *c)             /* 21 */
{
   int dimer = c->dimer;
   c->a_mu[repressB1]  = C_mu[repressB1]*c->B1*dimer/
                     (K[6]+pow((double)dimer,rephill));
}

void
UpdateUnRepressB1(CELL *c)                 /* 22 */
```

```
{
   c->a_mu[unRepressB1] = C_mu[unRepressB1]*(c->repB1);
}

void
UpdateAutoActivateB1(CELL *c)              /* 23 */
{
   int b1plex = c->b1plex;
   int id = (c->id == R4);
   c->a_mu[autoActivateB1]  = id*C_mu[autoActivateB1]*
                                        pow((double)b1plex,b1auto)/
                                        (K[4]+pow((double)b1plex,b1auto))*
                                        b1plex*(c->B1);
}

void
UpdateUnAutoActivateB1(CELL *c)          /* 24 */
{
   c->a_mu[unAutoActivateB1] = C_mu[unAutoActivateB1]*(c->autoB1);
}

void
UpdateTranscribeAutoB1(CELL *c) /* 25 */
{
   c->a_mu[transcribeAutoB1] = C_mu[transcribeAutoB1]*(c->autoB1);
}
void
UpdateDecayb1(CELL *c)                     /* 26 */
{
   c->a_mu[decayb1]  = C_mu[decayb1]*(c->b1);
}

void
UpdateActivateB2(CELL *c)          /* 27 */
{
   int act;
   int r3 = (c->id == R3);
   int r4 = (c->id == R4);
   int r5 = (c->id == R5);
         if(r4) act = c->b1plex;
         else if(r5) act = c->krox;
         else act = c->krox;
   c->a_mu[activateB2]  = !r3*C_mu[activateB2]*pow((double)act,b2hill)/
                     (K[5]+pow((double)act,b2hill))*act*(c->B2);
}
```

```
void
UpdateUnActivatedB2(CELL *c)                /* 28 */
{
   c->a_mu[unActivateB2] = C_mu[unActivateB2]*(c->actB2);
}

void
UpdateTranscribeB2(CELL *c)                 /* 29 */
{
   c->a_mu[transcribeB2] = C_mu[transcribeB2]*(c->actB2);
}

void
UpdateDecaymB2(CELL *c)                     /* 30 */
{
        c->a_mu[decaymB2] = C_mu[decaymB2]*(c->mB2);
}

void
UpdateTranslateB2(CELL *c)                  /* 31 */
{
   c->a_mu[translateB2] = C_mu[translateB2]*(c->mB2);
}

void
UpdateDecayb2(CELL *c)                      /* 32 */
{
   c->a_mu[decayb2]  = C_mu[decayb2]*(c->b2);
}

void
UpdateDivide(CELL *c)                       /* 33 */
{
        /* no changes needed */
}

void
UpdateActivateKrox(CELL *c)                         /* 34 */
{
   int r3 = (c->id == R3);
   c->a_mu[activateKrox]  = !r3*C_mu[activateKrox]*(c->Krox);
}

void
UpdateUnActivateKrox(CELL *c) /* a_mu[35] */
{
```

```
   c->a_mu[unActivateKrox] = C_mu[unActivateKrox]*(c->actKrox);
}

void
UpdateTranscribeKrox(CELL *c) /* a_mu[36] */
{
   c->a_mu[transcribeKrox] = C_mu[transcribeKrox]*(c->actKrox);
}

void
UpdateDecaymKrox(CELL *c)     /* a_mu[37] */
{
        c->a_mu[decaymKrox] = C_mu[decaymKrox]*(c->mKrox);
}

void
UpdateRepressKrox(CELL *c)    /* a_mu[38] */
{

   int b1 = c->b1;
   int a1 = c->a1;
   int max = (a1 > b1) ? a1 : b1;

   c->a_mu[repressKrox]  = C_mu[repressKrox]*c->Krox*(max)/
                  (K[6]+pow((double)(max),rephill));

}

void
UpdateUnRepressKrox(CELL *c)  /* a_mu[39] */
{
   int r3 = (c->id == R3);
   c->a_mu[unRepressKrox] = !r3*C_mu[unRepressKrox]*(c->repKrox);
}

void
UpdateTranslateKrox(CELL *c)  /* a_mu[40] */
{
   c->a_mu[translateKrox] = C_mu[translateKrox]*(c->mKrox);
}

void
UpdateDecaykrox(CELL *c)      /* a_mu[41] */
{
   c->a_mu[decaykrox]  = C_mu[decaykrox]*(c->krox);
}
```

```
void
UpdateBindRXR(CELL *c)              /* a_mu[42] */
{
        c->a_mu[bindRXR] = C_mu[bindRXR]*(c->ra)*(c->rxr);
}

void
UpdateMakeRXR(CELL *c)  /* a_mu[43] */
{

}

void
UpdateDecayRXR(CELL *c)             /* a_mu[44] */
{
   c->a_mu[decayRXR]  = C_mu[decayRXR]*(c->rxr);
}

void
UpdateUnbindBRXR(CELL *c)       /* a_mu[45] */
{
        c->a_mu[unbindBRXR]  = C_mu[unbindBRXR]*(c->brxr);
}

void
UpdateDecayBRXR(CELL *c)        /* a_mu[46] */
{
        c->a_mu[decayBRXR]  = C_mu[decayBRXR]*(c->brxr);
}

void
UpdateDimerize(CELL *c)             /* a_mu[47] */
{
        c->a_mu[dimerize] = C_mu[dimerize]*(c->brar)*(c->brxr);

}

void
UpdateUnDimerize(CELL *c)   /* a_mu[48] */
{
        c->a_mu[unDimerize] = C_mu[unDimerize]*(c->dimer);
}

void
UpdateDecayDimer(CELL *c)       /* a_mu[49] */
```

```
{
	c->a_mu[decayDimer] = C_mu[decayDimer]*(c->dimer);
}

void
UpdateComplexa1(CELL *c)
{
	c->a_mu[complexa1] = C_mu[complexa1]*(c->a1)*(c->plex);
}

void
UpdateUnComplexa1(CELL *c)
{
	c->a_mu[unComplexa1] = C_mu[unComplexa1]*(c->a1plex);
}

void
UpdateDecaya1Complex(CELL *c)
{
	c->a_mu[decaya1Complex] = C_mu[decaya1Complex]*(c->a1plex);
}

void
UpdateComplexb1(CELL *c)
{
	c->a_mu[complexb1] = C_mu[complexb1]*(c->b1)*(c->plex);
}

void
UpdateUnComplexb1(CELL *c)
{
	c->a_mu[unComplexb1] = C_mu[unComplexb1]*(c->b1plex);
}

void
UpdateDecayb1Complex(CELL *c)
{
	c->a_mu[decayb1Complex] = C_mu[decayb1Complex]*(c->b1plex);
}

void
UpdateMakeComplex(CELL *c)
{

}
```

```
void
UpdateDecayComplex(CELL *c)
{
        c->a_mu[decayComplex] = C_mu[decayComplex]*(c->plex);
}
```

## ranlib

The ranlib routines used in this program are in the public domain and can be found at http://www.netlib.org/random/ and are fully described in the literature (L'Ecuyer et al., 1991).

## Makefile

```
CC = gcc

CFLAGS = -O2 -Wall
TARGET = a.out

LIBS = -lm



SRCS = Hox.c main.c write_gnu_data_file.c inputs.c UpdateAmu.c ll.c ranlib.c com.c
OBJS = Hox.o main.o write_gnu_data_file.o inputs.o UpdateAmu.o ll.o ranlib.o com.o

$(TARGET):  $(OBJS)
            $(CC) -o $(TARGET) $(CFLAGS) $(LFLAGS) $(OBJS) $(LIBS)
            chmod 755 $(TARGET)

main.o:     main.c Hox.h header.h
            $(CC) -c $(CFLAGS) $(LFLAGS) main.c

Hox.o:      Hox.c Hox.h header.h
            $(CC) -c $(CFLAGS) $(LFLAGS) Hox.c

UpdateAmu.o:UpdateAmu.c UpdateAmu.h header.h
            $(CC) -c $(CFLAGS) $(LFLAGS) UpdateAmu.c

ranlib.o:   ranlib.c ranlib.h
            $(CC) -c $(CFLAGS) $(LFLAGS) ranlib.c

com.o:      com.c ranlib.h
```

```
		$(CC) -c $(CFLAGS) $(LFLAGS) com.c

inputs.o:	inputs.c Hox.h header.h
		$(CC) -c $(CFLAGS) $(LFLAGS) inputs.c

ll.o:		ll.c Hox.h header.h
		$(CC) -c $(CFLAGS) $(LFLAGS) ll.c

write_gnu_data_file.o:		write_gnu_data_file.c
		$(CC) -c $(CFLAGS) $(LFLAGS) write_gnu_data_file.c

clean:
		rm -f $(TARGET) $(OBJS) count
```

# Appendix D: Mathematica Source Code

The *Mathematica* package that was used for making the movies is included below. The notebook can be found on the CD-ROM as well.

## *Basic Enzyme Reaction*

The following source code was used to generate the data used for Figures 2.1 and 2.2.

```
MM[inputsub_, inputenz_, end_,k_List] :=
	Module[{kf = k[[1]], kb = k[[2]], k2 = k[[3]], enz = inputenz,
			sub = inputsub, com = 0, pro = 0, t = 0.0, tt = {},  dat = {}, s},
		While[t < end && (sub > 0 || com > 0),
			amu = {kf*sub*enz, kb*com, k2*com};
			a0 = Plus @@ amu;
			r1 = Random[ ];
			t += - Log[r1]/a0;
			r2 = Random[ ];
			picker = r2*a0;
			s = Drop[FoldList[Plus, 0, amu], 1];
			Which[picker < s[[1]],
					sub -= 1; enz -= 1; com += 1,
				picker < s[[2]],
					sub += 1; enz += 1; com -= 1,
				picker < s[[3]],
```

```
                                    enz += 1; com -= 1; pro += 1
                ];
                AppendTo[tt, t];
                AppendTo[dat, {sub, enz, com, pro}]
            ];
            {tt, dat}
        ]
```

## *Data Display Routines*

### Response Curves

For displaying how the levels in a particular cell of a certain specie changes over time.

```
Needs["Graphics`MultipleListPlot`"];

Response[files:{___String},rhom_] := Module[{data = {},name,i,j,m,l},
   clrs = {RGBColor[1,0,0],RGBColor[0,1,0],
      RGBColor[0,0,1],RGBColor[0,1,1],RGBColor[1,0,1],RGBColor[0,0,0],
      RGBColor[1,.5,0],
      RGBColor[0,.5,.5],RGBColor[.5,1,.5],RGBColor[1,.5,.5],
      RGBColor[.5,.5,.5]
      };
   l = Length[files];
   For[j = 1, j ≤ l, j++,
    d = ReadList[files[[j]],Real,RecordLists->True];
    If[rhom == 5,
        d = Map[Part[#,2]&,d],
            d = Map[Part[#,4]&,d]
     ];
    AppendTo[data,d];
     ];
   m = Max[data];
   For[i=1, i<=l, i++,mx = Length[data[[1,i]]];
     a =
      Table[Text[
        StyleForm[files[[i]],FontColor->clrs[[i]]],{8,12-2*i}],{i,1,
        l}]
    ];
   MultipleListPlot[Apply[Sequence,data], SymbolStyle->clrs ,
    Prolog->a]
  ]
```

## Excess Variance

The routine that generates the Figures 3.10 and the like.

```
Needs["Graphics`MultipleListPlot`"];

ExcessVar[files:{___String}] := Module[{data = {},name,i,j,m,l},
   clrs = {RGBColor[1,0,0],RGBColor[0,1,0],
      RGBColor[0,0,1],RGBColor[0,1,1],RGBColor[1,0,1],RGBColor[0,0,0],
      RGBColor[1,.5,0],
      RGBColor[0,.5,.5],RGBColor[.5,1,.5],RGBColor[1,.5,.5],
      RGBColor[.5,.5,.5]
      };
   l = Length[files];
   For[j = 1, j ≤ l, j++,
    d = ReadList[files[[j]],Real,RecordLists->True];
        d = Map[Part[#,2]&,d];
    AppendTo[data,d];
    ];
   m = Max[data];
   For[i=1, i<=l, i++,mx = Length[data[[1,i]]];
     a =
      Table[Text[
         StyleForm[files[[i]],FontColor->clrs[[i]]],{8,12-2*i}],{i,1,
         l}]
   ];
   MultipleListPlot[Apply[Sequence,data], SymbolStyle->clrs , Prolog->a]
   ]
```

## Level Initilaztion

The initial incarnation of the data display routines, these are still in use by J. Solomon

(personal communication).

```
Needs["Graphics`MultipleListPlot`"];

Movie[dir_,files:{___String}, num_Integer,opts___Rule] := Movie[dir,files,0,num,opts];

Options[Movie] = {Step->1};

Movie[dir_,files:{___String}, start_Integer,num_Integer,opts___Rule] :=
  Module[{data = {},name,numbers={},i,j,m,l},
    clrs = {RGBColor[1,0,0],RGBColor[0,1,0],
       RGBColor[0,0,1],RGBColor[0,1,1],RGBColor[1,0,1],RGBColor[0,0,0],
```

```
    RGBColor[1,.5,0],
    RGBColor[0,.5,.5],RGBColor[.5,1,.5],RGBColor[1,.5,.5],
    RGBColor[.5,.5,.5]
    };
  mystep=Step/.{opts}/.Options[Movie];
  l = Length[files];
  For[j = 1, j ≤ l, j++,
   data = {};
     For[i = start, i ≤ num, i+=mystep,
    name =dir<>files[[j]]<>"."<>ToString[i]<>".dat";
    d = ReadList[name,Real,RecordLists->True];
    d = Map[Last,d];
    AppendTo[data,d];
     ];
   AppendTo[numbers,data];
   ];
  m = Max[numbers];
  Which[m < 2500, scale = 100,
            m < 6000, scale = 300,
           m < 10000, scale = 500,
           True, scale = 1000];
  For[i=1, i≤(num-start)/mystep, i++,mx = Length[numbers[[1,i]]];
    a =
    Table[Text[
       StyleForm[files[[i]],FontColor->clrs[[i]]],{mx-2,
         m-i* scale}],{i,1,l}];
   MultipleListPlot[Map[Part[#,i]&,numbers],PlotRange->{-200,m},
    SymbolStyle->clrs ,
    Prolog->a]]
  ]
```

## Stain Initilaztion

These routines produce the virtual dynamic *in situ* as in Figure 3.5.

```
Stain[direc_,files:{___String}, num_Integer,opts___Rule] :=
Stain[direc,files,0,num,opts];

Options[Stain] = {Step->1,FrameScale->False,Tiffs->False};

TimeSlice[data_,time_] := Module[{},
  Map[#[[time]]&,data]
];
```

```
SpecieSlice[data_,specie_] := Module[{},
   Map[#[[specie]]&,data]
];

MakeBar[specie_,num_,name_,gmx_,border_,ndir_] :=
 Module[{bar={},i,mol,scale,color,tc,x,y,gmaxx},
   mx = 1+Max[specie];
   gmaxx = gmx  + 1;
   xoff = 6;
   s = specie/(mx+1);
   s = specie;
   numdirs = Length[specie];
   Which[num == 5,
    tc = CMYKColor[0,1,1,0],
    num== 4,
    tc = CMYKColor[1,0,1,0],
    num == 3,
    tc = CMYKColor[1,1,0,0],
    num == 2,
    tc = CMYKColor[0,1,0,0],
    num ==1,
    tc = CMYKColor[1,0,0,0],
    num ==6,
    tc = CMYKColor[0,0,1,0]
   ];
   bar = {bar,CMYKColor[0,0,0,0],
      Rectangle[{xoff,num+(y-1)/numdirs},{xoff+45,num+y/numdirs}]};
   For[y = 1, y ≤ numdirs,y++,
    numpoints = Length[specie[[y]]];
    For[x = 1, x≤ numpoints,x++,
     dat = s[[y,x]]/gmaxx[[y]];
     Which[num == 5,
      color =CMYKColor[0,0+dat,0+dat,0],
      num== 4,
      color =CMYKColor[0+dat,0,0+dat,0],
      num == 3,
      color = CMYKColor[0+dat,0+dat,0,0],
      num == 2,
      color =CMYKColor[0,0+dat,0,0],
      num == 1,
      color =CMYKColor[0+dat,0,0,0],
      num == 6,
      color =CMYKColor[0,0,0+dat,0]
      ];

     bar = { bar,
```

```
          {color,
           Rectangle[{x+xoff,num+(y-1)/numdirs},{x+1+xoff,
              num+y/numdirs}]} };
       ];
      l =
       Line[{{border[[y]]+2+xoff,num+(y-1)/numdirs},{border[[y]]+2+xoff,
            num+y/numdirs}}];
      bar = {bar,{CMYKColor[0,0,0,1],l}};
      ];
    (*bar = {bar,CMYKColor[0,0,0,1]} Text[
            "r5",{border[[1]]/2+xoff,ndir+2.5}],
       Text["r4",{border[[1]]+xoff+border[[1]]/2,ndir+2.5}]};
      bar = {bar,{tc,Text[name,{-4+xoff,num+1/numdirs}]}};*)

    bar = Flatten[bar];
    bar
    ]
MakeFrame[tslice_,name_,num_,border_,ndir_,mx_,tiffs_] :=
    Module[{i,l,b = {},counter,ss,x,y,dim,g,filename},
      l = Length[tslice[[1]]];
      xoff = 6;
      For[i = 1, i ≤ l,i++,
        b = {b,
           MakeBar[SpecieSlice[tslice,i],i,name[[i]],Transpose[mx][[i]],
             border,ndir]}
        ];
      time = 7.75+Floor[num/72]*.05;
      counter = ToString[time]<>" dpc";
      x =border[[1]];
      b = {b,Text[counter,{x+xoff+3,ndir+2.5}]};
      g = Graphics[b];
      Show[g,PlotRange->All];
      If[tiffs,
        filename = ToString[num]<>".tiff";
        Display["TIFF/"<>filename,g,"TIFF",ImageResolution->300];
      ]
];

Stain[dirs:{___String},files:{___String}, start_Integer,num_Integer,
    opts___Rule] :=
  Module[{data ,name,numbers,rundat,i,j,m,l,mx={},t,mystep,sf,f = files,
      border,ndir = Length[dirs]},
    mystep=Step/.{opts}/.Options[Stain];
    sf=FrameScale/.{opts}/.Options[Stain];
    tiffs = Tiffs/.{opts}/.Options[Stain];
    l = Length[f];
```

```
data  = {};
border = {};
mx = {};
For[runs = 1, runs ≤ Length[dirs], runs++,
  AppendTo[border,ReadList[dirs[[runs]]<>"border",Real]];
  numbers = {};
   For[i = start, i ≤ num, i+=mystep,
    rundat = {};
      For[j = 1, j ≤ l, j++,
        name =dirs[[runs]]<>f[[j]]<>"."<>ToString[i]<>".dat";
        d = ReadList[name,Real,RecordLists->True];
        d = Map[Last,d];
        AppendTo[rundat,d];
        AppendTo[mx,Max[d]];
     ];
    AppendTo[numbers,rundat];
  ];
 AppendTo[data,numbers];
];
mx = Partition[mx,Length[mx]/Length[dirs]];
mx = Map[Partition[#,5]&,mx];
maximums = {};
For[i = 1, i≤ Length[mx], i++,
  AppendTo[maximums,Map[Max,Transpose[mx[[i]]]]];
  ];
For[i = 1, i <=Length[numbers],i++,
  tslice = TimeSlice[data,i];
  MakeFrame[tslice,f,i,TimeSlice[border,i],ndir,maximums,tiffs];
  ]
]
```

## Plots

The plots are then invoked with the following typical commands

```
Response[{"mhoxa1","mhoxa1.100","mhoxa1.20","mhoxa1.2000","mhoxa1.7500"},5]


Movie["output.13/",{"ra","hoxa1","hoxb1","hoxb2","rar","rxr","dimer","krox20",
  "brar","brxr"},898,Step->2]

Stain[{"wt/output.13/","wt/output.17/","wt/output.19/",
  "wt/output.23/"},{"mkrox20","mhoxb2","mhoxb1","mhoxa1","ra"},1081,
 Step->1,Tiffs->True]
```

# References for Appendices

Barrow, J. R., Stadler, H. S., and Capecchi, M. R. (2000). Roles of Hoxa1 and Hoxa2 in patterning the early hindbrain of the mouse. *Development* **127,** 933-44.

Gavalas, A., Studer, M., Lumsden, A., Rijli, F. M., Krumlauf, R., and Chambon, P. (1998). Hoxa1 and Hoxb1 synergize in patterning the hindbrain, cranial nerves and second pharyngeal arch. *Development* **125,** 1123-36.

Gavalas, A., Trainor, P., Ariza-McNaughton, L., and Krumlauf, R. (2001). Synergy between *Hoxa1* and *Hoxb1*: the relationship between arch patterning and the generation of cranial neural crest. *Development* **128,** 3017-3027.

Hamburger, V., and Hamilton, H. (1951). A series of normal stages in the development of the chick embryo. *J. Morph.* **88,** 49-92.

Han, K., and Manley, J. (1993). Functional Domains of the Drosophila Engrailed Protein. *Embo. J.* **12,** 2723-2733.

Itasaki, N., Bel-Vialar, S., and Krumlauf, R. (1999). 'Shocking' developments in chick embryology: electroporation and in ovo gene expression. *Nat. Cell. Biol.* **1,** E203-7.

L'Ecuyer, P., Cote, S., Brown, B. W., Lovato, J., and Russell, K. (1991). Implementing a random number package with splitting facilities. *ACM TOMS* **17,** 98-111.

LaBonne, C., and Bronner-Fraser, M. (2000). Snail-Related Transcriptional Repressors are Required in Xenopus for both the Induction of the Neural Crest and Its Subsequent Migration. *Dev. Biol.* **221,** 195-205.

Qiagen. (2000). SuperFect Transfection Reagent Handbook.

Studer, M., Gavalas, A., Marshall, H., Ariza-McNaughton, L., Rijli, F. M., Chambon, P., and Krumlauf, R. (1998). Genetic interactions between Hoxa1 and Hoxb1 reveal new roles in regulation of early hindbrain patterning. *Development* **125,** 1025-36.

Swartz, M., Eberhart, J., Mastick, G. S., and Krull, C. E. (2001). Sparking New Frontiers: Using in Vivo Electroporation for Genetic Manipulations. *Dev Biol* **233,** 13-21.

Tang, M., Redemann, C., and Szoka, F. (1996). *In vitro* gene delivery by degraded polyamidoamine denrimers. *Bioconjugate Chemistry***,** 703.

Vignali, R., Poggi, L., Madeddu, F., and Barsacchi, G. (2000). HNF1 beta is required for mesoderm induction in the Xenopus embryo. *Development* **127,** 1455-1465.

Wilkinson, D. G. (1992). Whole mount *in situ* hybridization of vertebrate embryos. *In* "In situ hybridization: A Practical Approach" (D. G. Wilkinson, Ed.), pp. 75-83. IRL Press, Oxford.