# Automated Techniques for Stability Analysis of Delta-Notch Lateral Inhibition Mechanism*

Ashish Tiwari
SRI International
333 Ravenswood Ave
Menlo Park, CA, U.S.A
tiwari@csl.sri.com

Patrick Lincoln
SRI International
333 Ravenswood Ave
Menlo Park, CA, U.S.A
lincoln@csl.sri.com

## Abstract

*We describe, in detail, the modeling and automatic analysis of the Delta-Notch lateral inhibition mechanism using a hybrid systems framework. Inhibitory lateral signaling between adjacent cells is one of the central processes responsible for cell-differentiation in a cluster of identical cells. The basic modeling unit used in this paper is a hybrid automata and more complex models are built using compositions over hybrid automata. Analysis of the resulting model is done by creating a finite-state discrete abstract transition system for the hybrid model and model-checking this abstraction against the properties of interest. The abstraction technique, which is completely automated, uses suitable support from decision procedures and theorem-provers. The automated abstraction of continuous models of biological systems onto discrete transition systems enables new kinds of analysis on problems of biological interest.*

## 1. Introduction

Pattern formation is one of the most interesting phenomenon in development. In many kinds of developing tissue, adjacent cells differentiate and create a pattern of cells in contrasting states. Such pattern formation has been shown to be a consequence of intracellular gene regulatory interactions as well as lateral cell to cell signaling.

The Notch signaling mechanism controls an extraordinarily broad spectrum of cell fates and developmental processes (in organisms ranging from sea urchins to humans). Signals transmitted through the Notch receptor, in combination with other cellular factors, influence differentiation, proliferation, and apoptotic events at all stages of development [6].

### 1.1. Basics of Notch Signaling

The *Notch* gene encodes a transmembrane receptor, Notch (and its homologues). Notch-like proteins have been identified and extensively characterized in *Caenorhabditis elegans*, sea urchins, and many different vertebrates, including humans. In all animal models tested, mutation in the Notch receptor invariably resulted in developmental abnormalities and human pathologies [6].

Studies in *Drosophila*, C. elegans, and more recently in vertebrates have shown that a transmembrane protein, Delta, acts as a Notch ligand and is responsible for the transmission of the lateral signal. The basic picture emerging from many different studies suggests that the extracellular domain of the ligands (like Delta), expressed on the surface of one cell, interacts with the extracellular domain of the Notch receptor on an adjacent cell. This receptor activation futher triggers other transmitting pathways, a process that is shown to be affected by mutations in *Notch*.

A cell that expresses Delta more strongly activates Notch more strongly in its neighbours; and activation of Notch in these cells inhibits not only their differentiation, but also their expression of Delta, thereby reducing their ability to deliver lateral inhibition in return. This feedback loop tends to amplify any initial difference between neighbouring cells and results in pattern formation in a cluster of almost identical cells.

### 1.2. Modeling

In recent years, there has been an emphasis on studying simple models of some complex biological phenomenon with the goal to explain, and possibly predict or suggest, observations made by experiments. This approach is aimed at

*This research was supported in part by DARPA under the BioSpice program under contract DE-AC03-765F00098 through the Lawrence Berkeley National Laboratory, DARPA under the MoBIES program administered by AFRL under contract F33615-00-C-1700, and NASA Langley Research Center contract NAS1-00108 to Rannoch Corporation.

detecting some fundamental commonalities at some coarse-grained level [26].

A suitably rich and widely used modeling framework that has been used in diverse computer science and engineering applications is that of *hybrid systems* [4, 2, 5, 15]. A hybrid system combines discrete and continuous behaviors. Many complex biological systems can be described by breaking them down to separate the discrete behavior from the continuous one. The discrete component is usually a reflection of gene expression (being "on" or "off") or inhibitor proteins turning off the transcription of genes by RNA polymerase, while the continuous dynamics results from the protein concentration dynamics being described by differential equations (to capture exponential decay and other behaviors, for example), see [13] and [1].

### 1.3. Automated Analysis

Our emphasis here is on the application of symbolic toolsets from the formal methods community to problems of biological interest, particularly signaling and metabolic pathways. The computational tools are used not just to perform simulations on the models of biological systems, but to do more formal analysis completely automatically. The results of formal symbolic analysis with extremely efficient representations of biological networks provide insights with potential biological impact. In particular, novel hypotheses may be generated which can lead to wet-lab validation of new signaling possibilities.

However, there are certain hurdles to this, the main one being that the general reachability problem for even very simple kinds of hybrid systems is known to be undecidable [16]. Even though several decidable classes have been identified, see [3] for a survey, but all of these classes are too weak to represent any complex combination of discrete and continuous dynamics.

In this paper, we use *abstraction*, which is a technique to reduce the complexity of a system design, while preserving some of its relevant behavior, to construct a simplified system that is more accessible to analysis tools and is still sufficient for proving a certain class of properties. We use the technique resulting from a combination of data abstraction and qualitative abstraction, described in [31], to construct the abstraction. The abstract model is model-checked to automatically prove certain stability properties of the hybrid models of Delta-Notch lateral inhibition phenomenon. The methods described here provide an initial demonstration of the automated abstraction of continuous models of biological systems that provides the foundation for the application of high-powered tools, such as model checkers, to problems of biological interest.

Among the new kinds of calculation and prediction enabled by the application of formal methods techniques are the ability simulate biological systems *backwards*, answering complex queries such as "What are all the initial states that would lead to such-and-such final state?". We may also ask more complex queries about the paths in the transition graph, such as "If we reach a state that satisfies property $P$, is it true that we must eventually reach a state that satisfies property $Q$?". We may also want to restrict our attention to the subset of paths in the transition graph that satisfy some *fairness* criterion such as "If reaction $R$ is always possible then eventually reaction $R$ happens".

### 1.4. On the Abstraction Technique

An abstract model is usually obtained from a given model by partitioning the state-space of the original model into a finite set and mapping the dynamics of the original model onto this finite set. In the case of a model described using $n$ real-variables (representing, say, the concentrations of $n$ different complexes), the state-space is the $n$-dimensional real space $\mathbb{R}^n$. We partition this space into a finite number of zones using a finite set $P$ of polynomials over the $n$ variables. Each zone corresponds to a subset of $\mathbb{R}^n$ that is sign-invariant for all polynomials in the set $P$. Increasing the number of polynomials in the set $P$ results in more zones, and consequently, a finer abstraction. This basic idea, although in a much simplified form, is also at the core of qualitative reasoning techniques developed in the Artificial Intelligence community [28, 20, 29].

The process of construction of the abstract system requires logical reasoning in the theory of reals. The first-order theory of real closed fields is known to be decidable [30] and the first practical algorithm, based on cylindrical algebraic decomposition, was given in [9], which has gone through several improvements [25, 19, 17]. We use the first-order theory of reals to represent sets of continuous states and use reasoning over this theory for creating the abstract transition system.

## 2. The Model

We first formally describe the modeling paradigm used for hybrid systems. The basic modeling unit is a hybrid automata, which is described in Section 2.1. The modeling languauge of the analysis tool, SAL, is described next in Section 2.2, and the SAL model of the Delta-Notch lateral inhibition mechanism is presented in Section 2.3.

### 2.1. Hybrid Automata

A variable is *discrete* if it is interpreted over a countable domain; for example, integer and boolean variables are discrete variables. In contrast, a *continuous* variable takes values in the set $\mathbb{R}$ of real numbers. A *valuation* of a set of variables is a mapping from the variables to values from the domain of interpretation. If $Q$ and $X$ are finite sets of discrete and continuous variables, then we use $\mathbf{Q}$ and $\mathbf{X}$ to denote the set of all valuations of the variables $Q$ and $X$, respectively.

A hybrid system consists of both discrete, $Q$, and continuous, $X$, variables. Each valuation $\vec{q} \in \mathbf{Q}$ of the discrete variables specifies a *mode* of the hybrid system. In each such mode, the continuous variables of the system evolve according to some rate, which is specified by giving the time derivative of that continuous variable. If $\vec{q} \in \mathbf{Q}$ is a valuation of the discrete variables and $\vec{x} \in \mathbf{X}$ is a valuation of the continuous variables, then $(\vec{q}, \vec{x})$ is a valuation of all the variables in the system and represents a *state* of the system.

A formal model of a hybrid system is given in terms of a *hybrid automata*.

**Definition 1** *A hybrid automaton $HS$ is a tuple $(Q, X, Init, t, f)$, where $Q$ is a finite set of discrete variables, $X$ is a set of continuous variables, $Init \subseteq \mathbf{Q} \times \mathbf{X}$ is a set of initial states, $t \subset \mathbf{Q} \times \mathbf{X} \times \mathbf{Q}$ is a set of (guarded) discrete transitions, $f : \mathbf{Q} \mapsto (\mathbf{X} \mapsto T\mathbf{X})$ is a mapping from the discrete states to vector fields that specify the continuous flow in that discrete state.*

An execution of such a hybrid automaton starts in an initial state $(\vec{q_0}, \vec{x_0}) \in Init$ and goes through a sequence of states, $\langle(\vec{q_0}, \vec{x_0}), (\vec{q_1}, \vec{x_1}), (\vec{q_2}, \vec{x_2}), \ldots\rangle$, such that each new state, $(\vec{q_{i+1}}, \vec{x_{i+1}})$, is obtained from the previous state, $(\vec{q_i}, \vec{x_i})$, using one of the following transitions:

(a) continuous transition: In this case, the mode remains unchanged, i.e., $\vec{q_{i+1}} = \vec{q_i}$, but the continuous states evolve according to the rates specified by $f(\vec{q_i})$, i.e., if $F$ is a solution of the (set of) differential equations $\dot{F} = f(\vec{q_i})$, then $\vec{x_i} = F(0)$ and $\vec{x_{i+1}} = F(\tau)$ for some time instant $\tau > 0$. (The notation $\dot{F} = f(\vec{q_i})$ denotes a simultaneous set of differential equations, one for each continuous variable of the system, and the solution $F$ is a vector of functions.)

(b) discrete transition: In this case, the continuous state component remained unchanged, i.e., $\vec{x_{i+1}} = \vec{x_i}$, but the mode changes according to the requirement that $(\vec{q_i}, \vec{x_i}, \vec{q_{i+1}}) \in t$.

Thus, a hybrid automaton defines a collection of trajectories in the system state space, where each individual trajectory is given as above. We wish to point out that there are more general models of hybrid automata which allow for (i) the continuous component to change in a discrete transition (called a *reset*), and (ii) specification of invariants at each discrete mode. Furthermore, certain variables in the set of discrete and continuous variables can be marked as *input* variables. These variables are changed by the environment (continuously) and their dynamics are not specified in the hybrid automata. We can also use *parameters* (constants with unspecified value) in the specification of hybrid automata.

### 2.2. The SAL Modeling Language

Hybrid systems described in terms of interacting hybrid automata can be specified using the SAL modeling language [11, 7]. Here we shall only describe those features of the SAL modeling language that will be used in the specification of the Delta-Notch signaling mechanism.

A SAL specification consists of *module* definitions. A special kind of module, called *basemodule*, corresponds to a hybrid automaton. It consists of definitions of the *variables*, the *initial states*, and the discrete and continuous *transitions* of the hybrid automaton. The variables are classified as *input*, *output*, *local*, and *global*. Variables are typed, discrete variables are usually of type BOOLEAN and continuous variables are of type REAL.

The initial states of the hybrid automaton can be specified in one of two ways: either by giving values for some of the variables (the others are assumed to take any value from their domain), or by specifying a formula. In the latter case, the set of initial states consists of all those valuations of variables that make the formula true.

The discrete and continuous transitions are specified using a set of *guarded commands*. A guarded command is written as:

$$guard \longrightarrow x_1' = e_1; x_2' = e_2; x_3' = e_3; \ldots$$

where *guard* is a formula over the variables, $x_i$ are variables, and $e_i$ are expressions over the variables. Semantically, the above command means that if the formula in the guard is true for some state (a state is a valuation of all the variables), then the transition can be taken. In case of a discrete transition, the variables $x_i$ are updated to the new values given by expressions $e_i$. In a continuous transition, the variables $x_i$ on the left-hand sides of the assignment are special variables, whose names end with the suffix "dot". They specify the time-derivative of the corresponding continuous variable. The semantics of the continuous transition is as in hybrid automata.

### 2.3. The SAL Model of Delta-Notch Inhibition Mechanism

The hybrid model of the Delta-Notch signaling mechanism we use here is taken from [13], which is similar to the model in [21, 22] except that the continuous sigmoid switching curve for protein production and gene expression is replaced by a discrete switch. Modeling the gene expression switching by a discrete on/off switch, rather than a very steep sigmoid, is advantageous because it results in simpler models (by separating concerns), which are easy to simulate and analyze. In cases where experimental evidence suggests a more smooth sigmoid, the hybrid modeling framework can be used to model successively closer approximations by adding incremental complexity.

We first consider the model of a single cell. Since the *Notch* and *Delta* genes (and consequently the production of the corresponding proteins) can be independently switched "on" and "off", the hybrid model has four discrete states. These are defined using two BOOLEAN variables—$delta$ and $notch$. The continuous variables in the hybrid automata are: $v_D$ and $v_N$, which denote, respectively, the Delta and Notch protein concentrations in the cell, and the input variable $u_N$, which represents the concentrations of Delta in the neighboring cells. The model also consists of additional parameters—the Delta and Notch production rates, $R_D$ and $R_N$; the switching thresholds, $h_D$ and $h_N$; and the Delta and Notch decay constants, $\lambda_D$ and $\lambda_N$.

The continuous dynamics of the model are given as follows:

1. In the discrete mode $delta = false$, $notch = false$, the continuous evolution is governed by

$$\begin{aligned} \dot{v_D} &= -\lambda_D v_D \\ \dot{v_N} &= -\lambda_D v_N. \end{aligned}$$

2. In the discrete mode $delta = true$, $notch = false$, the continuous evolution is governed by

$$\begin{aligned} \dot{v_D} &= R_D - \lambda_D v_D \\ \dot{v_N} &= -\lambda_D v_N. \end{aligned}$$

3. In the discrete mode $delta = false$, $notch = true$, the continuous evolution is governed by

$$\begin{aligned} \dot{v_D} &= -\lambda_D v_D \\ \dot{v_N} &= R_N - \lambda_D v_N. \end{aligned}$$

4. In the discrete mode $delta = true$, $notch = true$, the continuous evolution is governed by

$$\begin{aligned} \dot{v_D} &= R_D - \lambda_D v_D \\ \dot{v_N} &= R_N - \lambda_D v_N. \end{aligned}$$

The discrete mode switch logic is given by the following transitions:

1. If the current mode is not $delta = false$, $notch = false$, and if $-v_N < h_D$ and $u_N < h_N$, then switch to the mode $delta = false$, $notch = false$;
2. If the current mode is not $delta = true$, $notch = false$, and if $-v_N \geq h_D$ and $u_N < h_N$, then switch to the mode $delta = true$, $notch = false$;
3. If the current mode is not $delta = false$, $notch = true$, and if $-v_N < h_D$ and $u_N \geq h_N$, then switch to the mode $delta = false$, $notch = true$;
4. If the current mode is not $delta = true$, $notch = true$, and if $-v_N \geq h_D$ and $u_N \geq h_N$, then switch to the mode $delta = true$, $notch = true$.

The SAL module corresponding to this specification is given here.

```
cell(lambdaD, lambdaN, RD,
RN, hD, hN : REAL) : MODULE =
BEGIN
   INPUT un :  REAL
   LOCAL vn :  REAL
   GLOBAL notch, delta :  BOOLEAN
   OUTPUT vd :  REAL
   LOCAL vddot, vndot :  REAL
   INITIALIZATION
       vn = 3/4; un = 0; vd = 0;
       delta = FALSE; notch = FALSE
   TRANSITION
       [ %% Mode switch transitions
       NOT(notch = FALSE AND
           delta = FALSE) AND
       -vn < hD AND un < hN -->
                 notch' = FALSE;
                 delta' = FALSE
       []
       NOT(notch = FALSE AND
           delta = TRUE) AND
       -vn >= hD AND un < hN -->
                 notch' = FALSE;
                 delta' = TRUE
       []
       NOT(notch = TRUE AND
           delta = FALSE) AND
       -vn < hD AND un >= hN -->
                 notch' = TRUE;
                 delta' = FALSE
       []
       NOT(notch = TRUE AND
           delta = TRUE) AND
       -vn >= hD AND un >= hN -->
                 notch' = TRUE;
                 delta' = TRUE
       [] %% Continuous transitions.
       notch = FALSE AND delta = FALSE
       AND -vn < hD AND un < hN -->
           vddot' = -lambdaD*vd;
```

```
            vndot' = -lambdaN*vn
        []
        notch = FALSE AND delta = TRUE
        AND -vn >= hD AND un < hN -->
            vddot' = RD - lambdaD*vd;
            vndot' = -lambdaN*vn
        []
        notch = TRUE AND delta = FALSE
        AND -vn < hD AND un >= hN -->
            vddot' = -lambdaD*vd;
            vndot' = RN - lambdaN*vn
        []
        notch = TRUE AND delta = TRUE
        AND -vn >= hD AND un >= hN -->
            vddot' = RD - lambdaD*vd;
            vndot' = RN - lambdaN*vn
        ]
END;
```

## 3. The Analysis

The SAL tool set provides interfaces that can be used to construct discrete abstractions of hybrid systems as described in [31]. The quantifier elimination decision procedure for the real closed fields is implicitly used to decide the implications over the real numbers. The tool QEPCAD [18], which is built over the symbolic algebra library SACLIB [8], is integrated to the theorem prover PVS [27] for this purpose.

The abstraction for the hybrid model can be constructed with respect to a property of interest, or independently. For this example, we specified properties related to stability of the system. The properties are defined in linear temporal logic, in short LTL. Apart from the standard logical connections, such as OR, AND, NOT, and ⇒ (implication), LTL has two additional operators—the always operator (also called the box operator), G, and the eventuality operator (also called the diamond operator), F. For instance, the formula $G(\mathit{delta} = \mathit{true}\ AND\ \mathit{notch} = \mathit{false} => G((\mathit{delta} = \mathit{true}\ AND\ \mathit{notch} = \mathit{false})))$ expresses that if the cell ever reaches the state $\mathit{delta} = \mathit{true}$ and $\mathit{notch} = \mathit{false}$, then it continues to remain in that state after that.

In SAL, such a property can be stated in the following way. Note that "cell" is the name of the module defined above.

```
stability1:  THEOREM
   cell |- G(delta AND notch = FALSE =>
             G(delta AND notch = FALSE));
```

### 3.1. Creating the Abstraction

In the model of a single cell, we shall prove the stability property mentioned above by first treating the input $u_N$ as a parameter, that is, $u_N$ is assumed to be a fixed symbolic constant. We do this by first abstracting the continuous and hybrid system behavior into a discrete, finite-state system using powerful symbolic decision procedures, and then model checking the finite-state system against the property of interest.

The abstraction technique is completely automatic and is described in [31]. The abstraction algorithm works in two phases: In the first phase, the algorithm computes a set of polynomials (over the continuous variables), which is used to partition the continuous state-space. This is achieved by starting with a finite set of polynomials which appear in the conditions of discrete transitions, initialization expressions, and property to be proved. This set is saturated under the derivative operator, that is, the time derivative of a polynomial in this set is also added to this set. Note that computation of the derivative of a polynomial can be symbolically performed using the derivatives of individual variables. This saturation process may not terminate, but can be stopped at any time for our purposes. The termination condition is specified by giving the bound on the order of the derivatives that is computed.

In the second phase, the discrete and continuous transitions are abstracted. The abstract state-space is given as a cross-product of the discrete-state space and the regions of the original continuous state-space where the polynomials computed in the saturation phase are sign-invariant. This requires the use of decision procedures for the real-closed fields, intermingled with theorem-proving strategies.

For scalability of the abstraction technique, it is necessary that the formulas that arise in proof obligations while constructing an abstraction and when checking for feasibility of abstract states, do not grow in size. We have achieved this by partitioning the set of variables into classes such that variables that interact with each other (occur in the same polynomial) are put into the same class. Furthermore, when proving a proof obligation, atomic formulas that are not "relevant" to the succedent (that is, atomic formulas that do not contain any variable that occur in the polynomials in the succedent) are removed from the formula. This optimization simplifies the formulas to be proved. Furthermore, it also reduces the number of proof obligations that arise in the construction of the feasible abstract state space.

A unique feature of our abstraction algorithm (and its implementation) is that it allows to abstract parameterized systems, without instantiating the parameters. In the Delta-Notch example, the abstract system is constructed using the following assumptions on the parameters:

$$\begin{array}{lll} lambdaD > 0, & lambdaN > 0, & RD > 0, \\ RN > 0, & hD < N, & hN > 0. \end{array}$$

Note that additional assumptions on the parameters can be made, for instance,

$$RD/lambdaD > hN, \quad RN/lambdaN > -hD.$$

The use of these two additional assumptions did not effect the validity of the properties with respect to the constructed abstraction. The discrete transition system for the one cell model, which we constructed for our experiments, consists of two boolean variables, *delta* and *notch*, and seven variables interpreted over the three-valued domain $SIGN = \{pos, zero, neg\}$. Thus, it has $4 * 3^7$ possible states. However, most of these states are "infeasible", that is, they do not correspond to a nonempty set of concrete states. In fact, the number of feasible abstract states is $4 * 484$. The mapping of the seven abstract variables, $g0, g1, \ldots, g6$, to polynomials over the original hybrid system is as follows:

$$\begin{array}{lll lll} g0 & : & RN + (-1) * vn * lambdaN & g3 & : & vn \\ g1 & : & RD + (-1) * vd * lambdaD & g4 & : & un - hN \\ g2 & : & vd & g5 & : & vd - hN \\ g6 & : & -vn - hD & & & \end{array}$$

The transitions in the abstract model arise from the discrete mode change transitions of the original model, or from the continuous dynamics. For instance, a mode switch to the mode $delta = true$ and $notch = false$, is abstracted to the following transition (we denote the boolean formula `delta = TRUE` by simply `delta`, and similarly for `notch`, in the subsequent text):

```
g4 = neg AND (g6 = pos OR g6 = zero)
AND NOT(delta AND notch = FALSE) -->
                notch' = FALSE;
                delta' = TRUE
```

Note that the condition $-vn \geq hD \;\wedge\; un < hN$ is abstracted to $(g6 = pos \;\vee\; g6 = zero) \;\wedge\; g4 = neg$—the first conjunct $g6 = pos \;\vee\; g6 = zero$ abstracting the condition $-vn \geq hD$ and the second conjunct $g4 = neg$ abstracting the condition $un < hN$. Continuous transitions are abstracted using qualitative techniques as described in [31]. For instance, the continuous evolution in the mode $delta = false \;\wedge\; notch = false$ is abstracted to the SAL transition shown in Figure 1.

Corresponding to the three other modes, there are three corresponding abstract transitions similar to the one in Figure 1. Note that the new value of a variable, say $g0$, is denoted by priming it, $g0'$. The binary "IN" operator denotes that the (new) value of the left-hand side variable is non-deterministically chosen from the set of values specified on the right-hand side. (In contrast, the "=" operator sets the value of the left-hand side variable to a deterministic value.) The pattern of the "assignments" to variables $g6$, $g1$, and $g0$ is identical, and similarly, the pattern of assignments to $g5$, $g3$, and $g2$ is the same. The reason is that the derivatives of the polynomials corresponding to $g6$, $g1$, and $g0$ are equal, upto a *positive* constant multiple, to the polynomial corresponding to $g3$, $g1$, and $g0$, respectively. Similarly, the derivative of the polynomial corresponding to $g5$, $g3$, and $g2$ are, upto a *negative* multiplicative constant, the polynomials corresponding to $g2$, $g3$, and $g2$, respectively. The derivatives are computed according to the dynamics in the mode given by $delta = false$ and $notch = false$. For example, in this mode, the derivative of the polynomial $R_N - \lambda_N v_N$, corresponding to $g_0$, is $\lambda_N^2 v_N$, which is a positive constant multiple of $v_N$, the polynomial corresponding to $g_3$.

Finally, we have treated the input variable $uN$ as a parameter, that is, the value of $uN$ is assumed to be a fixed symbolic constant. In the initialization, we can either fix $uN < hN$, or $uN \geq hN$. Thus, the abstract system captures the behavior of the concentrations of the delta and notch proteins in a single cell, assuming that the environment of the cell remains unchanged (or, changes slowly compared to the rate of the dynamics of the Delta-Notch concentrations.)

The condition in the guard involving "INV" checks that the new state computed by the transition (denoted by the primed variables in the argument of "INV") is feasible, that is, it corresponds to an nonempty set of concrete states. For example, the state $g2 = zero \;\wedge\; g5 = zero$ is infeasible, as we cannot have polynomials $vd$ and $vd - hN$ equal zero simultaneously. The "INV" condition guarantees that such states are not reached. (Note that SAL allows the presence of primed next-state variables in the guards.)

### 3.2. Proving Properties using Model-Checking

The abstract discrete state transition system constructed above is a finite-state system, and hence, it can be model-checked. The SAL explicit state model-checker explores a SAL specification and checks it against a *linear temporal logic*, or LTL, formula.

We successfully proved the following properties of the abstract single cell model:

```
stability1:  THEOREM
   cell |- G(delta AND notch = FALSE =>
             G(delta AND notch = FALSE));
stability2:  THEOREM
   cell |- G( delta = FALSE AND notch =>
             G(delta = FALSE AND notch));
```

```
g4 = neg AND g6 = neg AND delta = FALSE AND notch = FALSE AND
INV(g0', g1', g2', g3', g4', g5', g6') -->
         g6' IN IF g6 = pos THEN IF g3 = pos OR g3 = zero THEN {pos}
                                 ELSE {pos, zero} ENDIF
                ELSIF g6 = neg THEN IF g3 = neg OR g3 = zero THEN {neg}
                                    ELSE {neg, zero} ENDIF
                ELSE IF g3 = pos THEN {pos}
                     ELSIF g3 = neg THEN {neg} ELSE {zero} ENDIF ENDIF;
         g5' IN IF g5 = pos THEN IF g2 = neg OR g2 = zero THEN {pos}
                                 ELSE {pos, zero} ENDIF
                ELSIF g5 = neg THEN IF g2 = pos OR g2 = zero THEN {neg}
                                    ELSE {neg, zero} ENDIF
                ELSE IF g2 = neg THEN {pos}
                     ELSIF g2 = pos THEN {neg} ELSE {zero} ENDIF ENDIF;
         g4' IN IF g4 = pos THEN {pos}
                ELSIF g4 = zero THEN {zero} ELSE {neg} ENDIF;
         g3' IN IF g3 = pos THEN IF g3 = neg OR g3 = zero THEN {pos}
                                 ELSE {pos, zero} ENDIF
                ELSIF g3 = neg THEN IF g3 = pos OR g3 = zero THEN {neg}
                                    ELSE {neg, zero} ENDIF
                ELSE IF g3 = neg THEN {pos}
                     ELSIF g3 = pos THEN {neg} ELSE {zero} ENDIF ENDIF;
         g2' IN IF g2 = pos THEN IF g2 = neg OR g2 = zero THEN {pos}
                                 ELSE {pos, zero} ENDIF
                ELSIF g2 = neg THEN IF g2 = pos OR g2 = zero THEN {neg}
                                    ELSE {neg, zero} ENDIF
                ELSE IF g2 = neg THEN {pos}
                     ELSIF g2 = pos THEN {neg} ELSE {zero} ENDIF ENDIF;
         g1' IN IF g1 = pos THEN IF g2 = pos OR g2 = zero THEN {pos}
                                 ELSE {pos, zero} ENDIF
                ELSIF g1 = neg THEN IF g2 = neg OR g2 = zero THEN {neg}
                                    ELSE {neg, zero} ENDIF
                ELSE IF g2 = pos THEN {pos}
                     ELSIF g2 = neg THEN {neg} ELSE {zero} ENDIF ENDIF;
         g0' IN IF g0 = pos THEN IF g3 = pos OR g3 = zero THEN {pos}
                                 ELSE {pos, zero} ENDIF
                ELSIF g0 = neg THEN IF g3 = neg OR g3 = zero THEN {neg}
                                    ELSE {neg, zero} ENDIF
                ELSE IF g3 = pos THEN {pos}
                     ELSIF g3 = neg THEN {neg} ELSE {zero} ENDIF ENDIF
```

**Figure 1. Abstract SAL transition corresponding to the continuous dynamics in the mode $delta = false$, $notch = false$ of the Delta-Notch one cell model.**

These two properties state that if the system ever reaches a state where $delta = true$ and $notch = false$ (or, $delta = false$ and $notch = true$), then the system continues to remain in that state subsequently forever. These two properties can be proved with all possible initializations for the abstract variable $g4$, indicating that the properties are independent of how the environment sets the variable $uN$. In contrast, when the property `G(delta = FALSE AND notch = FALSE => G(delta = FALSE AND notch = FALSE))` is model-checked against the abstract system, we get a counter-example.

The above two properties show that certain states correspond to equilibrium points. To complete the proof of stability, we also need to establish that the system always eventually *reaches* one of these equilibrium states.

```
stability3:  THEOREM
  cell |- G(F((delta = FALSE AND notch)
              OR
              (delta AND notch = FALSE)));
```

However, the abstract model fails to verify this property, because the abstract system contains several self-loops (which prevent every execution from reaching the desired states). This is circumvented by adding certain "fairness" assumptions. In the Delta-Notch example, we added additional conditions in the guards of the abstract transitions corresponding to the continuous dynamics. In particular, we added the following conjunct in the each of the four guards: `PRO(g0, g1, g2, g3, g4, g5, g6, g0', g1', g2', g3', g4', g5', g6')`, where the progress function, `PRO`, is defined by,

```
PRO(x1,x2,x3,x4,x5,x6,x7,x8,x9,x10,
   x11,x12,x13,x14:  SIGN): BOOLEAN =
   NOT(x1 = x8 AND x2 = x9 AND
       x3 = x10 AND x4 = x11 AND
       x5 = x12 AND x6 = x13 AND
       x7 = x14);
```

The progress function states that the new state is different from the old state. With this modification, we were able to model-check the eventuality properties for the abstract model as well. It should be pointed out here that, in general, restricting the abstract model using such additional constraints could lead to unsound abstractions. We are further investigating this aspect to determine the kinds of properties that are preserved by imposing such restrictions.

### 3.3. Cluster of Cells

A two cell complex can be created by composing two single cells. Since the same variable names occur in the two instances of the one cell model, certain variables (which are not "local" to the module) need to be renamed to avoid conflicts. Similarly, communication between the two cells is captured by renaming variables to the same name.

```
twocells:  MODULE =
LOCAL vd1, vd2 IN
((RENAME g4 TO vd2, g5 TO vd1 IN cell)
    []
 (OUTPUT delta2, notch2 IN
    (RENAME g4 TO vd1, g5 TO vd2,
            delta TO delta2,
            notch TO notch2 IN cell)));
```

The composition operator, [], is used to construct the module "twocells" using two instances of the module "cell". We have connected the output variable $g5$ of one cell to the input variable $g4$ of another cell by renaming them to the same name. The variables $delta2$ and $notch2$ describe the mode of the second cell.

The module "twocells" can be model-checked against stability properties as before. In particular, we can verify the following properties of this two cell system.

```
stability4:  THEOREM
   twocells |-
      G((delta AND notch = FALSE AND
         delta2 = FALSE AND notch2)=>
        G(delta AND notch = FALSE AND
         delta2 = FALSE AND notch2));
stability5:  THEOREM
   twocells |-
      G((delta = FALSE AND notch AND
         delta2 AND notch2 = FALSE)=>
        G(delta = FALSE AND notch AND
         delta2 AND notch2 = FALSE));
```

This shows that the states where one cell has high Delta and low Notch concentration, while the other has low Delta and high Notch concentration, are stable states for a two cell complex. The other states are shown to be unstable, as model-checking the corresponding properties yields a counter-example. However, the property that the two cell system eventually reaches one of the equilibrium states does not hold true, and the model-checker produces a counter-example.

```
stability6:  THEOREM
   twocells |-
      F((delta = FALSE AND notch AND
         delta2 AND notch2 = FALSE) OR
        (delta AND notch = FALSE AND
         delta2 = FALSE AND notch2));
```

A closer look at the counter-example reveals that the property falsifying trajectory corresponds to an oscillatory behavior of the system. The abstract transition system can be interpreted as exhibiting the sum total of all behaviors of specific concrete realizations of the original system (using specific initialization values and parameter values consistent with the assumptions made during construction of the abstraction). Since certain choices of the parameters can lead to oscillatory behavior, the abstract system exhibits the same too. In the real world, however, minor perturbations and asymmetry can break the oscillation and force the system into one of the two stable states.

The stability of a cluster of cells can be analyzed based on the results above for a single cell. If we ignore the possibility of oscillation, we can construct a simple abstraction of a cluster of cells and automatically establish properties of such systems. For the Delta-Notch example, an analysis of a cluster of cells at a very coarse level of abstraction appears in [10].

## 4. Conclusion

The results here provide a step toward the new paradigm of "Symbolic Systems Biology" [12]. By utilizing high-powered computational methods at the symbolic level, Symbolic Systems Biology enables new kinds of analysis of the network effects in complex biochemical pathways. Automated abstraction linking models of biological systems at various levels of detail is a key component of any successful approach to systems biology.

Hybrid systems have been used to model, simulate, and analyze biological phenomena. Apart from the Delta-Notch signaling mechanism, hybrid representations have also been used to model genetic switch mechanism of $\lambda$ phage [24] and quorum sensing in *V. fischeri* [1]. In this paper, we have illustrated the use of automatic analysis tools to establish properties of a hybrid model of the Delta-Notch lateral inhibition mechanism. We have obtained results for the hybrid model using the SAL modeling and analysis framework. The computational time spent on creating the abstract system for the hybrid model described in this paper was of the order of a few seconds (real clock time). The time to model-check the abstract model against the various properties was of the same order.

We are working towards modeling certain other biological mechanisms in the SAL framework. The unique combination of decision procedures and theorem proving in the SAL toolset provides powerful support to develop more powerful analysis tools. The abstraction based methodology is well-suited to handle larger and much bigger examples, as it breaks up the automated reasoning obligations to large number of *small* steps.

**Acknowledgements.** We thank Claire Tomlin for pointing us to the Delta-Notch lateral signaling example. Most of the SAL development has been done by the members of the CSL group at SRI, and Leonardo Demoura, in particular, has developed the SAL model-checking environment that was used to carry out the experiments described in this paper.

## References

[1] R. Alur, C. Belta, V. Kumar, M. Mintz, G. J. Pappas, H. Rubin, and J. Schug. Modeling and analyzing biomolecular networks. 2002.

[2] R. Alur, T. Henzinger, and E. D. Sontag (eds.). *Hybrid Systems III*, volume 1066 of *LNCS*. Springer-Verlag, Berlin, 1996.

[3] R. Alur, T. A. Henzinger, G. Lafferriere, and G. J. Pappas. Discrete abstractions of hybrid systems. *Proceedings of the IEEE*, 88(2):971–984, July 2000.

[4] P. Antsaklis, W. Kohn, A. Nerode, and S. Sastry (eds.). volume 999 of *LNCS*. Springer-Verlag, Berlin, 1995.

[5] P. Antsaklis, W. Kohn, A. Nerode, and S. Sastry (eds.). *Hybrid Systems IV*, volume 1273 of *LNCS*. Springer-Verlag, Berlin, 1997.

[6] S. Artavanis-Tsakonas, M. D. Rand, and R. J. Lake. Notch signaling: Cell fate control and signal integration in development. *Science*, 284:770–776, Apr. 1999.

[7] S. Bensalem, V. Ganesh, Y. Lakhnech, C. Muñoz, S. Owre, H. Rueß, J. Rushby, V. Rusu, H. Saïdi, N. Shankar, E. Singerman, and A. Tiwari. An overview of SAL. In C. M. Holloway, editor, *LFM 2000: Fifth NASA Langley Formal Methods Workshop*, pages 187–196, Hampton, VA, jun 2000. NASA Langley Research Center.

[8] B. Buchberger, G. E. Collins, M. J. Encarnacion, H. Hong, J. R. Johnson, W. Krandick, R. Loos, A. M. Mandache, A. Neubacher, and H. Vielhaber. SACLIB 1.1 user's guide. In *RISC-Linz Report Series, Tech Report No 93-19*. Kurt Gödel Institute, 1993. `www.eecis.udel.edu/~saclib/`.

[9] G. E. Collins. Quantifier elimination for the elementary theory of real closed fields by cylindrical algebraic decomposition. In *Proc. Second GI Conf. Automata Theory and Formal Languages*, volume 33 of *LNCS*, pages 134–183, 1975.

[10] M. A. Colon and H. B. Sipma. Synthesis of linear ranking functions. In *Proc. of the 7th Intl. Conference on Tools and Algorithms for the Construction and Analysis of Systems, TACAS 2001*, volume 2031 of *LNCS*, pages 67–81. Springer, 2001.

[11] Computer Science Laboratory. SAL: Symbolic Analysis Laboratory. `http://www.csl.sri.com/projects/sal/`.

[12] S. Eker, M. Knapp, K. Laderoute, P. Lincoln, J. Meseguer, and K. Sonmez. Pathway logic: Symbolic analysis of biological signaling. In *Proceedings of the Pacific Symposium on Biocomputing*, pages 400–412, January 2002. To appear.

[13] R. Ghosh and C. J. Tomlin. Lateral inhibition through Delta-Notch signaling: A piecewise affine hybrid model. In M. D. D. Benedetto and A. Sangiovanni-Vincentelli, editors, *Hybrid Systems: Computation and Control HSCC*, volume 2034 of *LNCS*, pages 232–246. Springer, 2001.

[14] C. Haddon, Y.-J. Jiang, L. Smithers, and J. Lewis. Delta-Notch signaling and the patterning of sensory cell differentiation in the zebrafish ear: evidence from the mind bomb mutant. *Development*, 125:4637–4644, 1998.

[15] T. Henzinger and S. Sastry (eds.). volume 1386 of *LNCS*. Springer-Verlag, Berlin, 1998.

[16] T. A. Henzinger, P. W. Kopke, A. Puri, and P. Varaiya. What's decidable about hybrid automata? *Journal of Computer and System Sciences*, 57:94–124, 1998.

[17] H. Hong. An improvement of the projection operator in cylindrical algebraic decomposition. In *Proc. ISAAC 90*, pages 261–264, 1990.

[18] H. Hong. Quantifier elimination in elementary algebra and geometry by partial cylindrical algebraic decomposition version 13. In *The world wide web*, 1995. `www.eecis.udel.edu/~saclib/`.

[19] D. Lazard. *An improved projection for cylindrical algebraic decomposition*. Technical Report, Informatique, Universite Paris IV, F-75252 Paris Cedex 05, France, 1990.

[20] T. Loeser, Y. Iwasaki, and R. Fikes. Safety verification proofs for physical systems. In *Proc. of the 12th Intl. Workshop on Qualitative Reasoning*, pages 88–95. AAAI Press, 1998.

[21] G. Marnellos, G. A. Deblandre, E. Mjolsness, and C. Kintner. Delta-Notch lateral inhibitory patterning in the emergence of ciliated cells in Xenopus: Experimental observations and a gene network model. In *Pacific Symposium on Biocomputing*, pages 326–337, 2000.

[22] G. Marnellos and E. Mjolsness. A gene network approach to modeling early neurogenesis in *drosophila*. In *Pacific Symposium on Biocomputing*, pages 30–41, 1998.

[23] G. Marnellos and E. Mjolsness. Probing the dynamics of cell differentiation in a model of *drosophila* neurogenesis. In C. Adami, R. K. Belew, H. Kitano, and C. E. Taylor, editors, *Proc of the 6th Intl. Conf. on Artificial Life*, 1998.

[24] H. Matsuno, A. Doi, M. Nagasaki, and S. Miyano. Hybrid petri net representation of gene regulatory network. In *Pacific Symposium on Biocomputing*, 2000.

[25] S. McCallum. An improved projection operator for cylindrical algebraic decomposition of three dimensional space. *J. Symbolic Computation*, 5:141–161, 1988.

[26] I. Salazar-Ciudad, J. Garcia-Fernandez, and R. V. Sole. Gene networks capable of pattern formation: From induction to reaction-diffusion. *J. Theo. Biol.*, 205:??–??, 2000.

[27] N. Shankar, S. Owre, and J. M. Rushby. *The PVS Proof Checker: A Reference Manual*. Computer Science Lab, SRI International, 1993.

[28] B. Shults and B. J. Kuipers. Proving properties of continuous systems: qualitative simulation and temporal logic. *AI Journal*, 92:91–129, 1997.

[29] O. Sokolsky and H. S. Hong. Qualitative modeling of hybrid systems. In *Proc. of the Montreal Workshop*, 2001. Available from `http://www.cis.upenn.edu/~rtg/rtg_papers.htm`.

[30] A. Tarski. *A Decision Method for Elementary Algebra and Geometry*. University of California Press, 1948. Second edition.

[31] A. Tiwari and G. Khanna. Series of abstractions for hybrid automata. In C. J. Tomlin and M. R. Greenstreet, editors, *Hybrid Systems: Computation and Control HSCC*, volume 2289 of *LNCS*, pages 465–478. Springer, Mar. 2002.