CIMS-TR 2004-853

Naturally Speaking: A Systems Biology Tool with Natural Language Interfaces

Marco Antoniotti[2] **Ian T. Lau**[1,2] **Bud Mishra**[2,3]

[1] Biology Department
New York University
New York, NY, U.S.A.

[2] Bioinformatics Group
Courant Institute of
Mathematical Sciences
New York University
New York, NY, U.S.A.

[3] Cold Spring Harbor
Laboratory,
Cold Spring Harbor, NY,
U.S.A.

marcoxa@cs.nyu.edu itl204@nyu.edu mishra@nyu.edu

Keywords
Modeling, System Biology, Temporal Logic, User Interfaces, Natural language

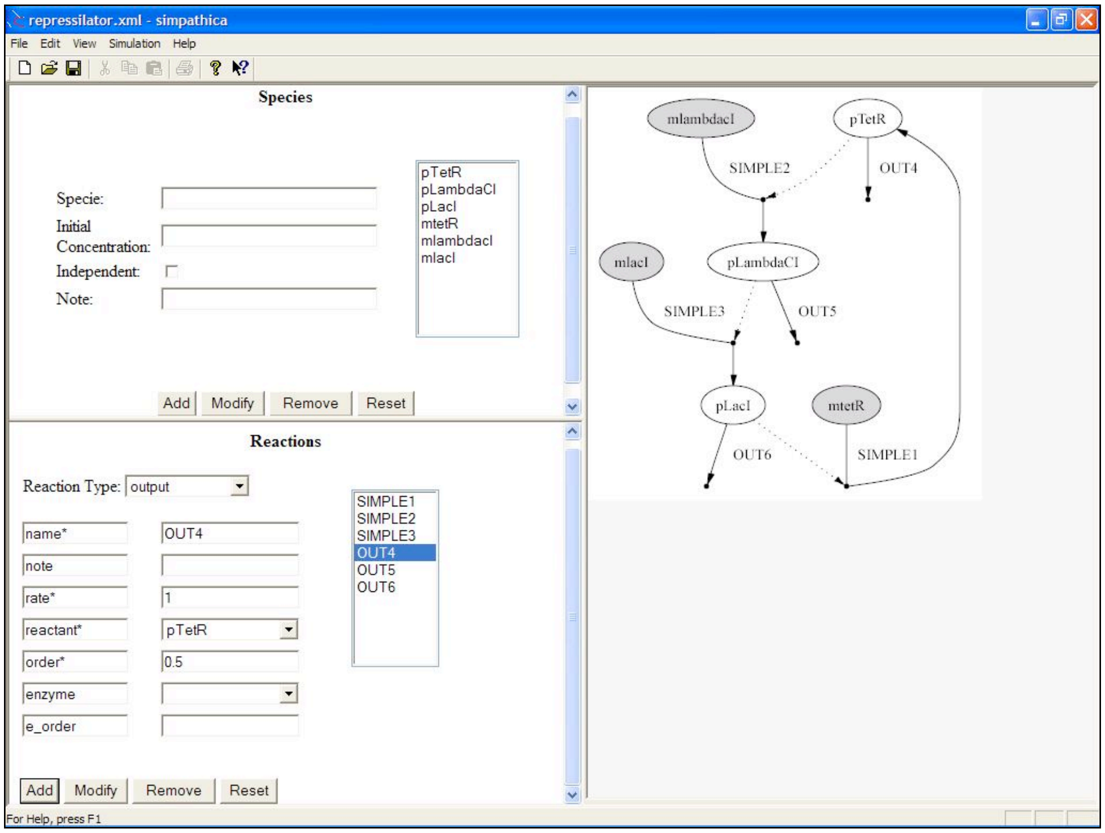
Abstract

This short paper describes a systems biology software tool that can engage in a dialogue with a biologist by responding to questions posed to it in English (or another natural language) regarding the behavior of a complex biological system, and by suggesting a set of "facts" about the biological system based on a time-tested "generate and test" approach. Thus, this bioinformatics system improves the quality of the interaction that a biologist can have with a system built on rigorous mathematical modeling, but without being aware of the underlying mathematically sophisticated concepts or notations. Given the nature of the mathematical semantics of our Simpathica/XSSYS tool, it was possible to construct a well-founded natural language interface on top of the computational kernel. We discuss our tool and illustrate its use with a few examples. The natural language subsystem is available as an integrated subsystem of the Simpathica/XSSYS tool and through a simple Web-based interface; we describe both systems in the paper. More details about the system can be found at: http://bioinformatics.nyu.edu, and its sub-pages.

Introduction

Many biologists face the hurdle of interacting with bioinformatics analysis tools that require mathematical sophistication and training. For example, drawing qualitative conclusions from time-course experimental data and simulated traces of mathematical models involves manually examining the data plots – possibly generated from differential or stochastic models – which are often fitted to actual experimental observations by means of involved statistical filtering procedures. As the number of traces and the amount of quantitative data increase, and their relationships become more intricate, this process not only becomes exceedingly time-consuming, but also bewilderingly complex. In addition, the process is further complicated by the care needed to avoid false inferences (either positive, negative, or both) when interpreting experimental data that is corrupted by highly correlated stochastic noise processes—a problem that worsens with dimension. Unfortunately, this is true of all currently available experimental datasets dealing with biological phenomena, e.g., microarray time-course experiments and models of complex biological systems, as they usually involve a large number of experimental conditions that are inter-related with one another. To address these problems, we devised the Simpathica/XSSYS Trace Analysis Tool, a bioinformatics system that enables users to query these datasets qualitatively using a propositional temporal logic.

Alas, the nature of our solution to the problem of complex data analysis introduces one more layer requiring a specialized training in the form of formulating hypotheses in temporal logic. Therefore, to make the system accessible to biologists, we have now integrated a *natural language query* subsystem within the Simpathica/XSSYS Trace Analysis Tool. In the following we describe our approach and give a few examples of its use. Finally, as an interesting avenue of exploration we also describe a prototype implementation of a "story generation" system based on a restricted exploration of the satisfiability of temporal logic sentences over a set of (simulated) traces of a biological system.



pTetR
pLambdaCl
pLacl
mtetR
mlambdacl
mlacl

Reaction Type: output

name*	OUT4
note	
rate*	1
reactant*	pTetR
order*	0.5
enzyme	
e_order	

SIMPLE1
SIMPLE2
SIMPLE3
OUT4
OUT5
OUT6

Figure 1. The Simpathica Main Window. The system being analyzed is the "repressilator" circuit (EL00). The top left frame contains a list of the reactants. The bottom left frame is used to insert different kinds of reactions selected from a list of known reactions. Finally the right frame contains a depiction of the reactions' network.

Description

The Simpathica/XSSYS Trace Analysis Tool (APP+03) uses a branching-time propositional temporal logic (E90) to formulate queries about the evolution of a biological system. Temporal logic (TL), also called tense logic, is a modal logic that incorporates special operators, or modes, that have a "temporal" interpretation. More concretely, it analyzes time course data sets for each observable variable using a concise and semantically well-founded temporal logic language. The Simpathica/XSSYS system can utilize data from a variety of sources, e.g. the NYUMAD and NYUSIM databases (RAC+01), various BioSpice modules (B03), PLAS files (V00), and simple CSV text files.

Temporal Logic has been studied in depth in the context of systems whose behavior changes in time, for instance, computer hardware, network protocols and engineering systems. We omit a detailed introduction to any or all of many specific Temporal Logics that have been introduced in the past. Instead we concentrate on the main ideas at the core of these logics in order to provide the intuition about how it can be used in the analysis of biochemical systems.

Fundamental to a temporal logic is the notion that time-dependent terms from natural language, such as "*sometimes*", "*eventually*" and "*always*," can be given a precise meaning (semantics) in terms of the abstract behavior of a system under discourse. As an example, consider the following sentence:

The concentration of guanosin triphosphate (GTP) is equal to k.

Such a sentence is *true* only in certain circumstances. Given a biological system in equilibrium the above sentence may or may not be true at any or all instants of time. In particular, we can easily construct sentences (in a suitable natural language) that express the fact that, *given a certain set of initial conditions* the above sentence *will **eventually** hold true*. Temporal Logic precisely formalizes the meaning of the adverb *eventually* (and other such "modes": *always*, *infinitely often* and *almost always*) and the resulting semantics lead to a precise model-checking algorithm for determining the validity of TL sentences in the context of an automaton.

This particular attribute of TL is very important as it concisely captures the notion of a logical property like "steady-state" and formalizes this notion in a simple consistent way that is directly handled by the model-checking algorithm.

Consider a system *M* and a (simulation) trace **trace***(M)*. If we consider a state *s* in **trace***(M)***,** we can simply check if all the first derivatives in *s* are 0. Suppose we have a procedure that answers *yes* (or *no*) when this is the case. Let us call this predicate, **zero_derivative**. Suppose that there actually is a state *s'* in **trace***(M)* where **zero_derivative** yields *yes*. Now, by the rules of Temporal Logic the following statement would be *true*

```
Eventually(zero_derivative)
```

for each instant from the start, at least up until the instant characterized as state *s'*.

Now we can expand the language of Temporal Logic and introduce a new predicate "steady state" to be a synonym of the following concept: there exists an instant (a state s' in **trace***(M)*) after which **zero_derivative** will always be true. More formally,

```
    steady_state(M)
```

is defined to be logically equivalent to the following:

```
    Eventually(Always(zero_derivative))
```

meaning that, when we consider the simulation (or *in vivo*) trace of the system there will be a time where all the rates of change of the system's variables reach 0 and remain at that value.

Alternatively, we could be more selective and ask whether some specific variable reaches the steady state. We can determine the answer as a result of the Definition 4.

```
    steady_state(M, GTP).
```

Another set of properties that we may want to express (and subsequently check) is the one involving "persistence." In other words, properties of the form: *something is* ***always*** *true* (or *false*). For instance, we could ask whether in a given system

```
    Always(GTP > k).
```

Thus, we query whether the `GTP` level always remains greater than k, independent of other changes occurring during the evolution of the system.

The previous discussion illustrates the main ideas needed to translate an English sentence involving temporal claims into a query in temporal logic. The translation from English to TL is rather straightforward. Simple conjunctions ("and"s), disjunctions ("or"s) and negations ("not"s) can be expressed directly

Suppose we wish to determine if (1) our system reaches a steady state *and* (2) the level of `GTP` is less than `k` after a certain instant. This statement is simply expressed in TL as

```
  steady_state and Eventually(Always(GTP < k)).          (a)
```

Note that the validity of the above statement is completely determined by the two constituent sub-expressions. Furthermore, the truth property of the statement requires examining the entire system trace, since **steady_state** is a "global" property, and the second conjunct has the same form. To appreciate the subtleties of TL, consider the following expression: eventually the system will be in steady state and the level of `GTP` will be less than `k`.

```
  Eventually(steady_state and Always(GTP < k)).          (b)
```

Given the properties of TL, the above expression (if true) will actually guarantee that when the system attains the steady state, it *also* has a `GTP` level less than `k`. This is a different statement than (a), and it shows how flexible and yet precise a TL statement can be, without sacrificing a high degree of expressive power.

There are other built in operators like conditionals that describe the system or the variable in a qualitative way. For example, the statement

```
    Always(CDK1 > 3 * CDC25)
        implies Eventually(steady_state()).
```

returns true if it is the case that if CDK1 is always more than 3 times CDC25, the system eventually reaches steady state, that is, there being no net change in the values of the quantitations. Nested queries such as

```
Always(PRPP = 1.7 * PRPP1)
     implies
          steady_state()
          and Eventually (Always(IMP < 2 * IMP1))
          and Eventually (Always(hx_pool < 10 * hx_pool))).
```

are just as simple for our tool to evaluate, though difficult for a human to understand at first glance (the variables PRPP, PRPP1, IMP, IMP1, hx_pool, and hx_pool1 appear in the analysis of the purine metabolism pathway described in (APUM03).)

In (APP+03) we discuss some of the mathematical and computational problems associated with this approach, e.g. the dependency of the analysis on the density of time points. The Simpathica/XSSYS system essentially implements a *model-checking* algorithm (CGP99) based on a "labeling" of each state, i.e., of each time-indexed time point. The labeling of states enables the Simpathica/XSSYS Trace Analysis Tool to use temporal logic to query complex logical dependencies of the variables on one another, using also some specialized "verbs" whose meaning should be more intuitive for a biologist.

For example, the query

```
Eventually(growing(CDK1)) and Always(CYCB > CDC25)).
```

would evaluate to true if within the data set, CDK1 eventually starts increasing and CYCB concentration always remains greater than that of CDC25. If the query is false over the trace, the system would indicate the time at which it first violates the condition.

Query Maker – A Natural Language interface

Although the Simpathica/XSSYS system is very powerful and effective, it is not very accessible to users without experience with the temporal logic, an admittedly complex and esoteric mathematical tool for the layperson. Therefore, we decided to wrap the Temporal Logic system with a natural language interface to make the system more accessible. Of course, several other systems have approached similar problems by providing a natural language interface to a computational tool. E.g., pioneering work at Edinburgh University in natural language in the context of model checking for hardware verification showed that a subset of English is sufficient to express temporal logic queries (HK99). We adapted the approach to our biological setting by building a specialized set of "verbs," immediately recognized by a biologist (e.g. "*growing*", "*steady state*", "*flat*",) and then tied it to our specialized data analysis tool. All in all, we assumed that "if a question cannot be asked in English, it will not be asked by a biologist." The Query Maker natural language interface is designed with this principle in mind.

The interface is built on top of a simple, context-free semantic parser (N92). Figure 2 shows a screenshot of the systems. The questions are first parsed, and have their semantics interpreted following a set of grammar rules. Then the questions are translated into temporal logic queries, which are then fed into the temporal logic system. Finally, the Temporal Logic queries are partially compiled with a "Just-In-Time" compiler that produces machine code for them. The system runs under Windows, Mac OSX and Linux, and it also has a Web-based interface at the address `http://bioinformatics.nyu.edu:3000/home/lasp`.

For example, if a biologist asks

"Is it eventually the case that if var1 is always between var2 and var3 and var4 is always constant, then v5 will always be bounded by v3?"

the question will be translated to

```
Eventually(Always(var1 > var2 and var1 < var3)
              and Always(flat(v4)))
                   implies Always(v3 < v5).
```

Even though Query Maker has many limitations, because of its small vocabulary and the fact that not all temporal logic queries can be expressed clearly in plain English, we can see that it is already able to formulate and manipulate relatively complex queries. Our hope is that after repeated usage, biologists would be able to formulate their own temporal logic queries with desired complexity.

Example: the Yeast Cell Cycle

The cell cycle is the sequence of repeating events through which a cell grows, replicates its genetic material, and finally, physically separates into two daughter cells. It is a tightly controlled process divided into the *G1*, *S*, *G2*, and *M* phases, corresponding to growth, duplication of genetic material, and finally mitosis. The control mechanisms of the budding yeast cell cycle can be accurately modeled, as in Novak and Tyson (NT97, NT01). We will inquire the traces of the wild-type model as well as a mutant that lacks a particular control mechanism (SK-knockout mutant).

It is known from various published analysis – e.g. (NT97, NT01) – that elimination of the SK control in the G1 phase causes CKIt (Cyclin-dependent Kinase Inhibitor) levels to remain high, disrupting the cycling of the events. As a result, the mutant system reaches a premature steady state, while the wild-type continues oscillating through the cell-cycle. In other words, the question

"Will the system eventually reach steady state?"

will yield a *true* answer for the mutant case, and yield a *false* answer for the wild-type.

It is also known that in wild type yeast, when CKIt level drops below CycBt, active Cyclin B begins to form and activates a cascade of events that propels the cell to divide. In the mutant, since CKIt levels do not drop due to the absence of SK, Cyclin B level remains low. Therefore, the question

"After 0.1 minutes, when CKIt is less than or equal to CycBt, does CycBt increase?"

will yield a *true* answer for the mutant case, and yield a *false* answer for the wild-type. In the mutant case the system answers with

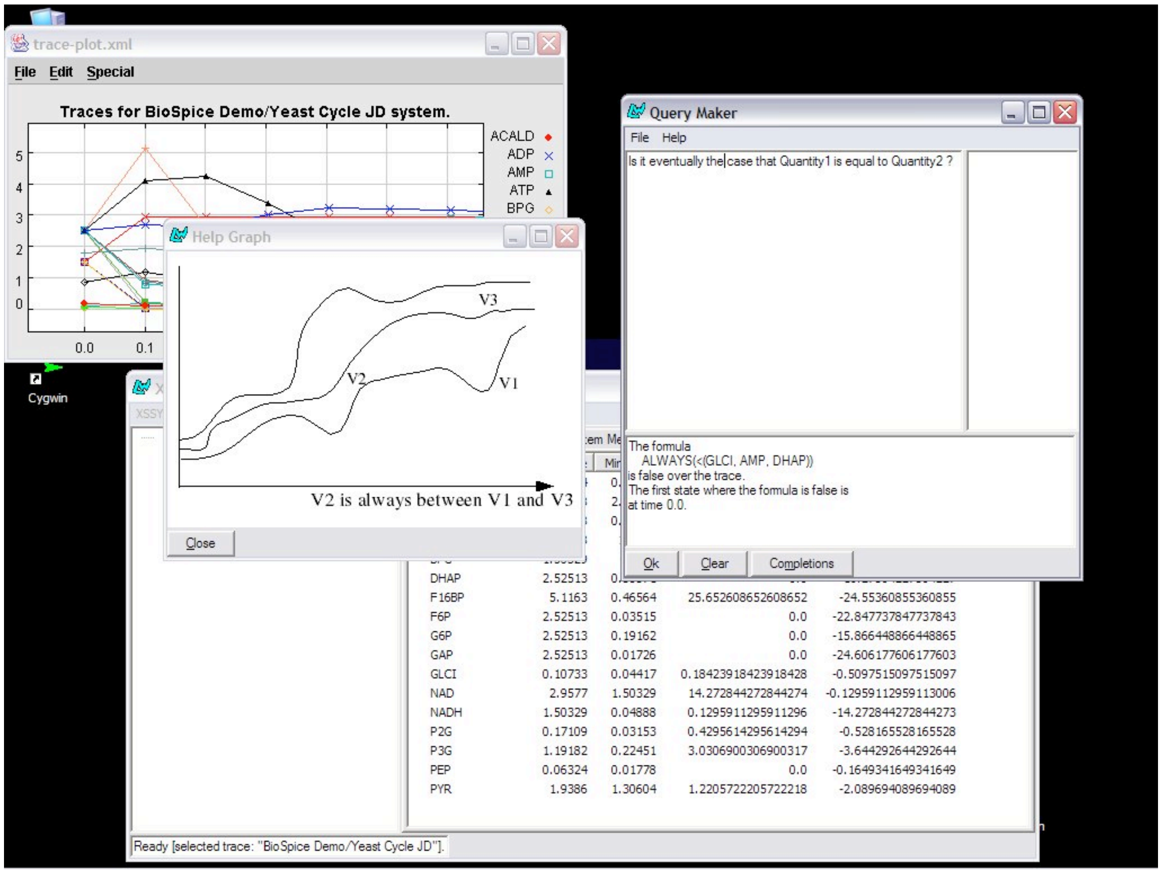
```
The formula
      Eventually((TICK > 0.1)
                   and AU(not(CKIT <= CYCBT)
                          and not(GROWING(CYCBT))
                          UNTIL
                          (CKIT <= CYCBT
                           and GROWING(CYCBT))))
is false over the trace.
```

I.e. the formula is false in the mutant case. Note the "internal" variable `TICK`, which represents time.

Integrated and Web-based User Interfaces

We have built two user interfaces for the Query Maker subsystem of XSSYS: an integrated one for the stand-alone application, shown in Figure 2, and a Web based one.

The integrated interface allows a user to formulate questions and check answers while being able to access all the functionalities of the XSSYS system. We also provide a simple "Help" facility that explains in a graphical way the meaning of each temporal logic operator, and that explains how to formulate questions that make them.



```
Is it eventually the case that Quantity1 is equal to Quantity2 ?
```

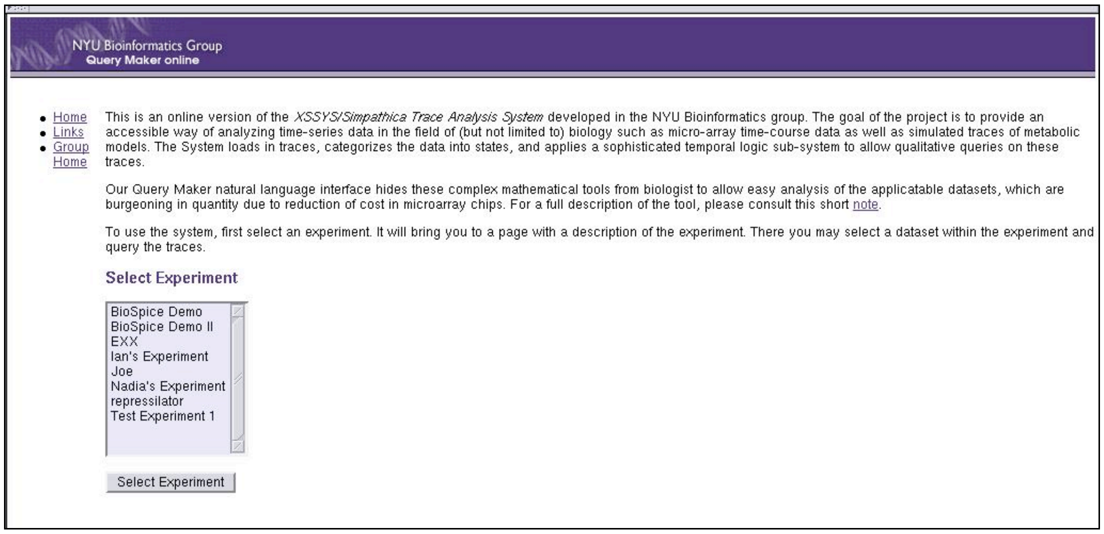
```
The formula
   ALWAYS(<(GLCI, AMP, DHAP))
is false over the trace.
The first state where the formula is false is
at time 0.0.
```

DHAP	2.52513	[illegible]	[illegible]	[illegible]
F16BP	5.1163	0.46564	25.652608652608652	-24.55360855360855
F6P	2.52513	0.03515	0.0	-22.847737847737843
G6P	2.52513	0.19162	0.0	-15.866448866448865
GAP	2.52513	0.01726	0.0	-24.606177606177603
GLCI	0.10733	0.04417	0.18423918423918428	-0.5097515097515097
NAD	2.9577	1.50329	14.272844272844274	-0.12959112959113006
NADH	1.50329	0.04888	0.1295911295911296	-14.272844272844273
P2G	0.17109	0.03153	0.4295614295614294	-0.528165528165528
P3G	1.19182	0.22451	3.0306900306900317	-3.644292644292644
PEP	0.06324	0.01778	0.0	-0.1649341649341649
PYR	1.9386	1.30604	1.2205722205722218	-2.089694089694089

Ready [selected trace: "BioSpice Demo/Yeast Cycle JD"].

Figure 2. A screenshot of the Simpathica/XSSYS Natural Language Interface. The "Query Maker" window is used to type in English queries. A Help System showing the intuitive meaning of typical queries can be also consulted to facilitate the expression of the Temporal Logic queries.

The Web based interface maps some of the simpler functionalities of the XSSYS application. The Web based interface is organized in three pages: (a) "dataset selection" page, (b) a "query" page, and (c) a "results" page. The three pages are shown in Figure 3, Figure 4, and Figure 5. The dataset chooser connects to our NYUSIM database, which is a repository of simulation traces. Simpathica and XSSYS write and read this database thus making it possible to keep a well ordered list of datasets along with their necessary meta-data for identification and explanation.



This is an online version of the *XSSYS/Simpathica Trace Analysis System* developed in the NYU Bioinformatics group. The goal of the project is to provide an accessible way of analyzing time-series data in the field of (but not limited to) biology such as micro-array time-course data as well as simulated traces of metabolic models. The System loads in traces, categorizes the data into states, and applies a sophisticated temporal logic sub-system to allow qualitative queries on these traces.

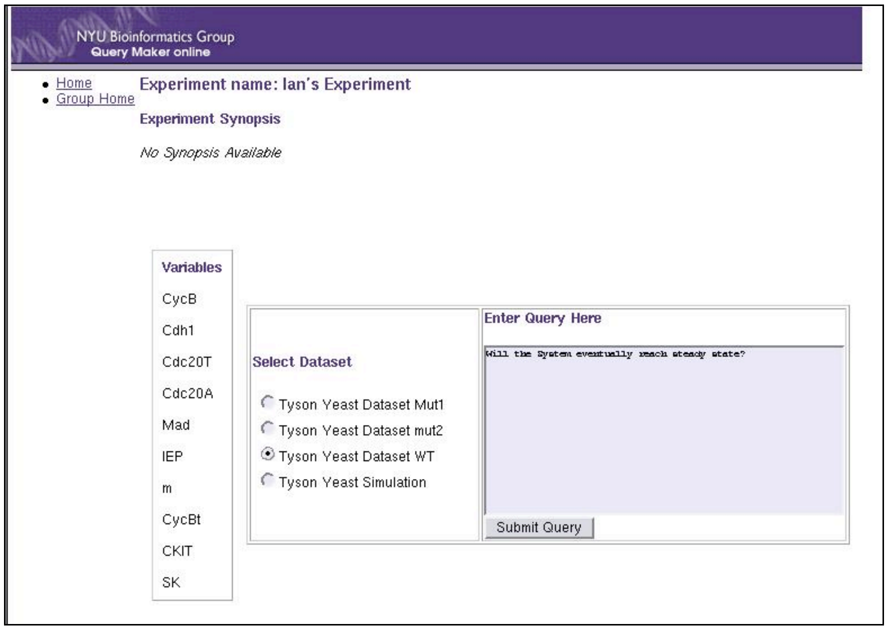
Our Query Maker natural language interface hides these complex mathematical tools from biologist to allow easy analysis of the applicatable datasets, which are burgeoning in quantity due to reduction of cost in microarray chips. For a full description of the tool, please consult this short note.

To use the system, first select an experiment. It will bring you to a page with a description of the experiment. There you may select a dataset within the experiment and query the traces.

Select Experiment

BioSpice Demo
BioSpice Demo II
EXX
Ian's Experiment
Joe
Nadia's Experiment
repressilator
Test Experiment 1

Figure 3. This is the opening page of the "Query Maker Online" (QMO) system viewable at http://bioinformatics.nyu.edu:3001/Projects/qmo/lasp/home. The system shows a list of the "experiments" for which the NYUSIM database has datasets visible to the general public.



Experiment name: Ian's Experiment

Experiment Synopsis

No Synopsis Available

Variables

CycB
Cdh1
Cdc20T
Cdc20A
Mad
IEP
m
CycBt
CKIT
SK

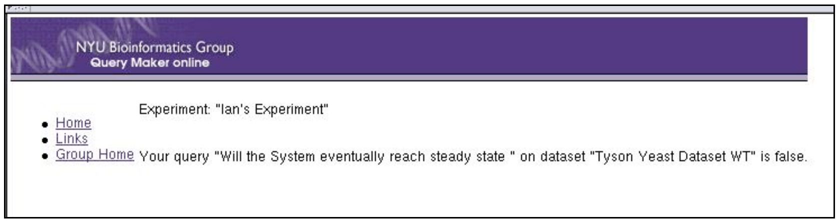
Select Dataset

( ) Tyson Yeast Dataset Mut1
( ) Tyson Yeast Dataset mut2
(●) Tyson Yeast Dataset WT
( ) Tyson Yeast Simulation

Enter Query Here

Will the System eventually reach steady state?

Figure 4. Once an experiment has been selected, QMO shows a page with a list of the "variables" appearing in each of the datasets form the experiment. The user can enter a query involving those variables in the text area on the right.



Experiment: "Ian's Experiment"

Your query "Will the System eventually reach steady state " on dataset "Tyson Yeast Dataset WT" is false.

Figure 5. After accessing the NYUSIM database, loading the data on the QMO server and performing the query analysis, the final page shows the result.

Sentence Generation of "Biologically Interesting Factoids"

At its core the XSSYS system manipulates a set of CTL temporal logic formulae. Each formula is easily translated into a natural language (English). Given this features, we explored the possibility to automatically generate several temporal logic formulae in increasing order of complexity (i.e. formula length,) with the intent of discovering new facts about a dataset, and to produce a " *biologically interesting factoid*" story for the consumption of a biologically savvy reader. I.e. we have set up a traditional *generate-and-test* framework. In the following we describe the generation algorithm and discuss some of its features.

The generation algorithm must use several heuristics to constrain the size of the set F of formulae to be analyzed, as a simple counting argument on the structure of the concrete syntax of CTL formulae, reveals that the number of formulae of "syntactic depth" d is $\square\left(2^{2^d}\right)$: obviously too large a number to consider, even for the simple case of $d = 3$. Given a number of relatively straightforward heuristics, the formula generation and testing procedure can be kept under control, although the worst case scenario still applies. The heuristics involved are based on a (arbitrary) lexicographic ordering of the variables, and on an accounting of the symmetries in the binary operators of the underlying temporal logic language. Also, user supplied ranges for the values of the variables involved are taken into account. In essence the procedure performs the following steps:

Procedure *Formula Generation*:

1. **Input**: a set of variable V_S from an experiment; the element of V_S are the "story variables."
2. **For each** formula template from the set:
    a. **`Represses(P1, P2).`**
    b. **`Activates(P1, P2).`**
    c. **`Steady_state().`**
    d. **`Constant(P, t1, t2).`**
    e. Formulae representing the response of the system to a particular input at time t_i (e.g. an *impulse* or a *sustained* input.)

    generate the set of all possible combinations of instantiated formulae using only the elements of V_S.

Because of the set of heuristics used, the resulting set of formulae has limited size. Once the set of formulae F has been generated, then we can check each of its members against the datasets comprised in an experiment. Figure 6 shows the overall architecture of the "story generation" system. The result will be a set of valuations for each $f \in F$ with respect to each dataset; e.g. a dataset corresponding to a wild-type and one corresponding to a mutant, as in the Yeast Cell Cycle example given before.

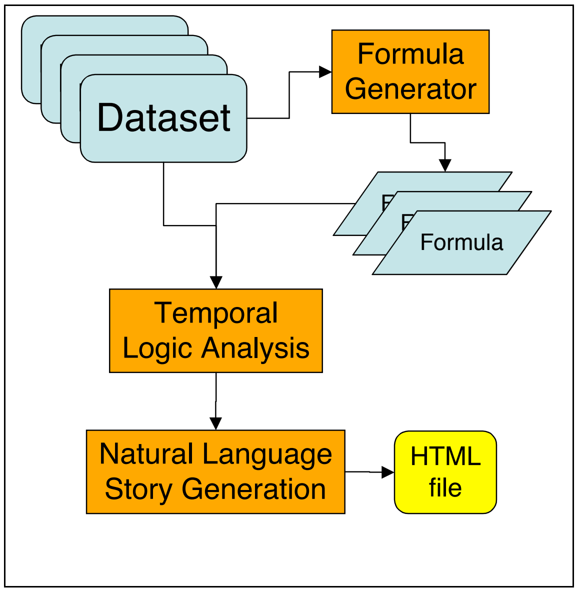


Figure 6. The architecture of the "biologically interesting factoid" generation system. Given a number of datasets (logically belonging to a given "experiment", the system generates a set of CTL formulae using a number of carefully chosen heuristics (to constrain the number of formulae being generated). Each formula is fed to the temporal logic analyzer XSSYS, which is essentially a restricted model-checker, and the results of such analysis is then fed to the Natural Language Generation system which finally produces an HTML formatted file.

Given a number of datasets and a set of "interesting" values for the variables in V_S, the "factoid generation" system produces an HTML formatted output. Table 1 shows an excerpt from the output produced by analyzing three datasets obtained by simulation of the Yeast Cell Cycle models described in (NT97).

Report on "Test Experiment Tyson WT, 1 Mutant, 2 Mutants.".

RESULTS

The results refer to the following datasets:

The first dataset is named "Ian's Experiment/Tyson Yeast Dataset WT".

The second dataset is named "Ian's Experiment/Tyson Yeast Dataset Mut1".

The third dataset is named "Ian's Experiment/Tyson Yeast Dataset mut2".

…

84. CDH1 less than or equal to 1.0071783 will always hold until CDH1 activates CYCB, is true in the first dataset, is true in the second dataset, and is false in the third dataset.
85. CDH1 represses CYCB implies CYCB is greater than or equal to 0.65, is false in the first dataset, is true in the second dataset, and is true in the third dataset.
86. CDH1 greater than or equal to 1.0071783 will always hold until CDH1 activates CYCB, is false in the first dataset, is true in the second dataset, and is true in the third dataset.
87. eventually, CDH1 is less than or equal to CYCB, is false in the first dataset, is true in the second dataset, and is true in the third dataset

Table 1. A fragment of the "biologically interesting factoid" story produced by the generation system. The system actually produced 234 such sentences involving the species CDH1 and CYCB and a number of "interesting values" they can assume. These sentences can be more readily looked up by a biologist and possibly indexed for better retrieval.

Concluding Remarks

We have presented a simple natural language interface for a time course data analysis tool that tackles the problem of making a mathematically sophisticated system more accessible to a biologist with little mathematical training. We have also presented an initial system that is capable of generating an English rendition of a long list of simple facts about a given biological system for which we have a simulatable model or an experimental "trace". While our systems rely on a large body of literature and experience, it also represents a novel integration of a wide array of techniques to solve a general problem facing bioinformaticists and biologists. Our implementations can obviously be improved in several ways. For instance, we are working closely with biologists to expand the set of predefined "verbs" and the grammar rules to account for more elaborate questions. Moreover, we are also taking into account more sophisticated temporal logic formulations that will lead to the manipulation of more expressive questions.

References

(APP+03) M. Antoniotti, F. Park, A. Policriti, N. Ugel, and B. Mishra, "Foundations of a Query and Simulation System for the Modeling of Biochemical and Biological Processes." The Pacific Symposium on Biocomputing: **PSB 2003**, (Eds. R.B. Altman, A.K. Dunker, L. Hunter, T.A. Jung & T.E.Klein), pp 116-127, World Scientific, January, 2003.

(APUM03) M. Antoniotti, A. Policriti, N. Ugel, B. Mishra, *Model Building and Model Checking for Biochemical Processes*, Cell Biochemistry and Biophysics, 38:271-286, 2003.

(B03) DARPA BioSpice project, Web site at http://www.biospice.org.

(CGP99) E. M. Clarke, O. Grunberg, and D. A. Peled, "Model Checking", MIT Press, 1999.

(EL00) M. Elowitz and S. Leibler (2000). A synthetic oscillatory network of transcriptional regulators, *Nature* **403**, 335-338.

(E90) E. A. Emerson, “Temporal and Modal Logic”, in *Handbook of Theoretical Computer Science*, Volume B, J. van Leeuwen (ed.), pp 996-1072, MIT Press, Cambridge, MA, U.S.A., 1990.

(HK99) A. Holt, and E. Klein, “A semantically-derived subset of English for hardware verification.” P*roceedings of the 37th Annual Meeting of the Association for Computational Linguistics*, University of Maryland, College Park, Maryland, USA, June 1999, pages 451-456 (Association for Computational Linguistics, 1999)(N92) P. Norvig, “Paradigms of Artificial Intelligence Programming: Case Studies in Common Lisp.” Morgan Kaufmann, 1992.

(NT97) B. Novak, and J. J. Tyson, “Modeling the control of DNA replication in fission yeast,” *Proc. Natl. Acad. Sci.* **94,** 9147-9152, 1997.

(NT01) B. Novak, and J. J. Tyson, “Regulation of Eukaryotic Cell Cycle: molecular Antagonism, Hysteresis and Irreversible Transitions,” *J. Theor. Biol.* **201,** 249-263, 2001.

(RAC+01) M. Rejali, M. Antoniotti, V. Cherepisky, C. Leventhal, S. Paxia, A. Rudra, J. West, B. Mishra, “Goals, Design and Implementation of a Versatile MicroArray Data Base”, *CGC 2001*, Baltimore, U.S.A., November 2001.

(V00) E. O. Voit, “Computational Analysis of Biochemical Systems,” Cambridge University Press, 2000.

Acknowledgements

The work reported in this paper was supported by grants from DARPA's BioCOMP project (Title: “Algorithmic Tools and Computational Frameworks for Cell Informatics,” http://www.biospice.org) and AFRL contract (contract #: F30602-01-2-0556). Additional support was provided by NSF's Qubic program, HHMI biomedical support research grant, Howard Hughes Honors Summer Research Institute at NYU, the US department of Energy, the US air force, National Institutes of Health and New York State Office of Science, Technology & Academic Research. We are also grateful to our colleagues at NYU and CSHL for their inputs and continuous support.