Pathway Logic

Carolyn Talcott*

SRI International
333 Ravenswood Avenue, Menlo Park, CA 94025, USA
clt@csl.sri.com

Abstract. Pathway Logic (PL) is an approach to modeling and analysis of biological processes based on rewriting logic. This tutorial describes the use of PL to model signal transduction processes. It begins with a general discussion of Symbolic Systems Biology, followed by some background on rewriting logic and signal transduction. The representation and analysis of a small model of Ras and Raf activation is presented in some detail. This is followed by discussion of a curated model of early signaling events in response to Epidermal Growth Factor stimulation.

Key words: Symbolic systems biology, rewriting logic, signal transduction, Pathway Logic, Epidermal Growth Factor signaling

1 Symbolic Modeling of Cellular Processes

Biological processes are complex. They exhibit dynamics with a huge range of time scales: microseconds to years. The spatial scales cover 12 orders of magnitude: metabolite to single protein to cell to organ to whole organism. Just considering the cellular level, cells interact with their environment, both sensing and affecting. They have many behaviors: they can grow, proliferate, migrate, differentiate, or die. Underlying these behaviors are a variety of processes such as gene regulation, signal transduction, and metabolism that interact with one another in complex ways. Genes are regulated by proteins (and other molecular entities) binding to promoter regions. This determines which genes are expressed (turned on) and thus which new proteins are produced. These proteins may in turn regulate the same or other genes. A cell senses its environment by receptors in the membrane that recognize specific types of molecules or conditions. This results in *signal transduction* that transmits the information to appropriate components inside the cell. Mechanisms underlying the flow of information include modification of protein state, formation of complexes, and change of location. The flow is controlled by mechanisms that activate or inactivate proteins in the signaling path. Metabolism involves both synthesis and degradation of chemicals to generate energy, synthesize protein building blocks (amino acids), and cell structure components amongst other things. Metabolic processes are controlled by enzymes which may in turn be activated or inhibited by signaling processes. Furthermore, metabolites such as glucose play a role in controlling signal flow.

* This work was partially supported by NSF grant IIS-0513857. The development of Pathway Logic was partially supported by grants from NIH NIGMS and NCI.

Oceans of experimental biological data are being generated, from both traditional and emerging high throughput techniques. How can we use this data to develop better models? Important intuitions are captured in mental models that biologists build of biological processes and the cartoons they draw. The trouble is that these models are not amenable to computational analysis.

High level statistical models can be developed, for example, to discover possible correlations and causal relations. Such models may suggest useful insights, but have many limitations such as features that can not be modeled. Low level, detailed kinetic or stochastic models can be developed for small subsystems, but often require parameter fitting, so that the reaction rates used reflect unknown biological context.

Symbolic systems biology is the *qualitative and quantitative* study of biological processes as integrated systems rather than as isolated parts. Our focus is on modeling causal networks of biomolecular interactions in a logical framework at multiple scales. The aim is to develop formal models that are as close as possible to domain experts (biologists) mental models. Furthermore, it is important to be able to compute with and analyze these complex networks. The latter includes techniques for abstracting and refining the logical models; using simulation and deduction to compute or check postulated properties; and make testable predictions about possible outcomes, using experimental results to update the models.

There are many challenges in developing symbolic systems models. One challenge is choosing the right abstractions. Biological networks (metabolic, protein, or regulatory, for example) are large and diverse. It is important to balance computational complexity against model fidelity and to be able to move between models of different levels of detail, using different formalisms in meaningful ways. Biological networks combine to produce high levels of physiological organization, for example, circadian clock subnetworks are integrated with metabolic, survival, and growth subnetworks. A second challenge is to be able to compose different views or models of different components into integrated system models.

Symbolic/logical models allow one to represent partial information and to model and analyze systems at multiple levels of detail, depending on information available and questions to be studied. Such models are based on formalisms that provide language for representing system states and mechanisms of change such as reactions. These languages come with a well-defined semantics, and tools for analysis are based on this underlying semantics. Of particular interest are symbolic models that are *executable*. An executable model describes system states and provides rules specifying the ways in which the state may change. Such models can be used for simulation of system behavior. In addition, properties of processes can be stated in associated logical languages and checked using tools for formal analysis.

Given an executable model, the path graph of a given initial state is a graph whose nodes are the reachable states and whose edges are the rules connecting them. Paths through the graph then correspond to possible ways a system can evolve. An execution strategy picks out a particular path among those possible. For such a model, there are many kinds of analysis that can be carried out, including: static analysis, forward simulation, forward search, backward search, model checking, constraint solving, and meta analysis.

Static analysis allows one to examine the structure of the model and to understand how the elements are related and organized (the sort structure). It can be used to infer flow of control and dependencies. Static analysis also provides a means to check for inconsistencies or ill-formed declarations and to look for missing information.

Forward simulation runs the model from a given initial state using a specified strategy either for a fixed number of steps, or until no more rewrites apply. This is extremely fast, and very useful for initial exploration.

Forward search is a breadth-first search of all paths through the transition graph for a given initial state. If the graph (number of states) is finite search will find *all* possible outcomes from a given initial state. Assuming finite branching, search will find all outcomes at a finite depth. Search can also be constrained to find only states satisfying a given property.

Backward search runs the model backwards. For models satisfying certain constraints, backwards search can answer the question: "From what initial states can we get to this state?". For example it can be used to find all possible precursors to a particular checkpoint, or to prove that a bad state cannot be reached from an initial state.

Model checking expands the collection of properties that can be investigated. Search concerns only properties of individual states. Model-checking tools are based on algorithms to determine if all computations of a system (pathways / sequences of steps) satisfy a given property. For example we can ask if molecule X is never produced before molecule Y has been produced. If not, a pathway that fails to satisfy the property (molecule Y is produced and molecule X is produced before it) is returned. Turning this around, to find a pathway satisfying a property of particular interest, one asserts that no such pathway exists and a counterexample will be one of the desired pathways. An example of another kind of property that can be model checked is: "If we reach a state that satisfies P then do we always later reach a state satisfying Q?"

Constraint solving attempts to find values for a set of variables that satisfy a given set of constraints. Maximal satisfiability (MaxSat) problems are a generalization of constraint satisfaction problems where there may be conflicting constraints, and hence no assignment of values to variables that will satisfy them all. Weights (importance measures) are assigned to constraints and a MaxSat solver finds a solution maximizing the total weight of the satisfied constraints. Many static analysis problems can be formulate as constraint systems. Steady state analyses such as determining possible flows of information or chemicals through a system can be formulated as constraint problems.

Meta analysis allows us to reason about the models themselves. Essential features of models can be abstracted to form families of related models, allowing us to work with uncertainty about reactions. Starting with a base set of known reactions, different instantiations of sets of reactions can be explored. For example, we can search for models where a given path property is true in a given initial state. In addition, rules themselves can be abstracted into families of rules, each family corresponding, for example, to a particular type of reaction, such as activation, inhibition, or translocation. It also allows the knowledge base to be queried as data base, for example finding all rules that involve a given protein (in any or a specified state or location). Finally, using mappings of logics a model can be mapped to another formalism to take advantage of additional tools.

2 A Sampling of Symbolic Modeling Approaches

A variety of formalisms initially developed to model and analyze concurrent computer systems have been used to develop symbolic models of biological systems, including: Petri nets [38, 48]; the pi-calculus [34, 35] and its stochastic variants [40]; membrane calculi [43, 36, 28]; statecharts [20, 10], life sequence charts [24]; rule-based systems including Rewriting Logic [33, 7] and P-systems [37]; and hybrid systems [21]. For a recent review of 'executable specification approaches' see [15]. A series of abstract machines each suited to modeling biological process associated to a different class of macromolecules is presented in [4] giving an nice introduction to the concepts to be modeled.

There are many variants of the Petri net formalism and a variety of languages and tools for specification and analysis of systems using Petri nets. Petri nets model networks of reactions that describe processes as well as process execution. Petri nets have a graphical representation that corresponds naturally to conventional representations of biochemical networks. They have been used to model metabolic pathways and simple genetic networks (e.g., see [22, 42, 19, 26, 32, 16]). In [29] timed Petri nets are used to model cellular signaling. These studies have been largely concerned with dynamic or kinetic models of biochemistry. In [55] a more abstract and qualitative view is taken, mapping biochemical concepts such as stoichiometry, flux modes, and conservation relations to well-known Petri net theory concepts. Overviews of different Petri net formalisms and their application to modeling biological processes can be found in [18, 6].

In contrast to Petri nets in which system state is explicit and processes emerge from rules/transistions that change the state, process calculi model molecular components as as processes. State is implicit in the interactions that processes may participate in. A pi-calculus model for the receptor tyrosine kinase/mitogen-activated protein kinase (RTK/-MAPK) signal transduction pathway is presented in [44]. BioSPI, a tool implementing a stochastic variant of the pi-calculus, has been used to simulate both the time course and probability of biochemical reactions [40].

BioAmbients [43], an adaptation of the Ambients formalism for mobile computations has been developed to model dynamics of biological compartments. BioAmbient type models can be simulated using an extension of the BioSPI tool. A technique for analysis of control and information flow in programs has been applied to analysis of BioAmbient models [36]. This can be used, for example, to show that according to the model a given protein could never appear in a given compartment, or a given complex could never form.

Statecharts naturally express compartmentalization and hierarchical processes as well as flow of control among subprocesses. They have been used to model T-cell activation [23, 10]. Life Sequence Charts [8] are an extension of the Message Sequence Charts modeling notation for system design. This approach has been used to model the process of cell fate acquisition during C.elegans vulval development [24].

Like Petri nets, rule-based formalisms model the state of molecular components directly, and state change is specified by rules. As will be described in the following sections, Pathway Logic [11, 12, 49, 51] represents biological processes using theories in rewriting logic. System state is represented as an algebraic term, and behavior is

specified by rewrite rules. Models can be directly analyzed by execution, search, and model-checking, or by mapping to other formalisms, such as Petri Nets. P-systems is a multiset rewriting formalism that provides a built in notion of location. A continuous variant of P-systems is used in [41] to model intra-cellular signaling. The model can be used to predict concentration of components, for example phosphorylated Erk, over time by a discrete step approximation method. A simple formalism for representing interaction networks using an algebraic rule-based approach very similar to the Pathway Logic approach is presented in [14, 5]. The language has three interpretations: a qualitative binary interpretation much like the Pathway Logic models; a quantitative interpretation in which concentrations and reaction rates are used; and a stochastic interpretation. Queries are expressed in a formal logic called Computation Tree Logic (CTL) and its extensions to model time and quantities. CTL queries can express reachability (find pathways having desired properties), stability, and periodicity. Techniques for learning new rules to achieve a desired system specification are described in [3].

Hybrid systems techniques are important for modeling processes where one wants to capture both continuous and discrete aspects. Models of glucose/insulin metabolism and B. subtilis sporulation are described in [30]. Hybrid system abstraction methods (see [53]) are used to analyze the model, for example to develop parameters for insulin control in diabetic patients. In [17] hybrid system models of the delta-notch system in Drysophila are studied using control theory and hybrid abstraction methods.

Symbolic executable models can be mapped to alternative logical formalisms for analysis. As will be discussed later, certain rewriting logic models can be mapped to Petri Nets for analysis by special purpose, efficient model checkers. In [2] a continuous stochastic logic and the probabilistic symbolic model checker, PRISM, is used to express and check a variety of temporal queries for both transient behaviors and steady state behaviors. Proteins modeled as synchronous concurrent processes, and concentrations are modeled by discrete, abstract quantities. Metabolic or signaling networks can be analyzed using a constraint-based technique that generalizes the well-known flux balance analysis [9] by representing the network as constraints on the reactions, rather than on the reacting components. In [54] this technique is used to compute preferred steady states under different conditions, also represented as constraints. Apart from understanding the steady-state configurations, constraint-based analysis can also be used to identify modules in the network, trace the flow of information in the network, and identify cross talk and conflicts.

3 Pathway Logic Overview

Pathway Logic (PL) [11, 12, 49, 52, 50, 51] is a symbolic systems biology approach to the modeling and analysis of molecular and cellular processes based on rewriting logic [33]. Such formal theories can include both specific facts and general principles relating and categorizing data elements and processes. New data structures for representing biological entities and their relations and properties can easily be defined. Theories concerning different types of information can also be combined using well-understood operations for combining logical theories. A wide range of analytical tools developed

for the analysis of computer system specifications is being adapted to carry out new kinds of analysis of experimental data curated into formal theories.

In PL, biological molecules, their states, locations, and their roles in molecular or cellular processes can be modeled at very different levels of abstraction. For example, a complex signaling protein can be modeled either according to an overall state, its post-translational modifications, or as a collection of protein functional domains and their internal or external interactions. Similarly biological processes can be represented at different levels of granularity using rewrite rules. Each rule represents a step (at the chosen level of granularity) in a biological process such as metabolism or intra-/inter-cellular signaling. A rule may represent a family of reactions using variables to stand for families of molecular components. Rules express dependencies on biological context; for example, a scaffold needed to hold proteins in position to interact productively.

A collection of rules together with the underlying data type specifications forms a PL knowledge base. Each biological molecule that is declared in a PL rewrite theory has associated metadata linking it to standard database entries, for example HUGO or UniProt/Swiss-Prot for proteins, along with other information such as category and synonyms. This information is part of the knowledge base. It is important to place the knowledge in a broader context and to be able to integrate it with other knowledge sources. Each rule has associated evidence used to justify the rule, which is also part of the knowledge base.

A PL model consists of a specification of an initial state (cell components and locations) interpreted in the context of a knowledge base. Such models are executable and can be understood as specifying possible ways a system can evolve. Logical inference and analysis techniques are used for simulation to study possible ways a system could evolve, to assemble pathways as answers to queries, and to reason about dynamic assembly of complexes, cascading transmission of signals, feedback-loops, cross talk between subsystems, and larger pathways. Logical and computational reflection are used to transform and further analyze models.

Pathways are not predefined. Instead they are assembled by applying the rules starting from an initial state, searching for a state meeting given conditions. For example, a pathway leading to specific conditions, such as activation of a Ras protein can be generated as the result of a logical query. A subnet (subset of reactions) composed of all possible relevant pathways can also be generated. A subnet consisting of connections to a given set of molecular components can be generated by graph exploration techniques.

PL knowledge is represented and analyzed using Maude [7], a rewriting-logic-based formalism. The Pathway Logic Assistant (PLA) [52] provides an interactive visual representation of PL models. In PLA, models are represented as graphs with nodes for rules and components, and edges connecting reactant components to rules and rules to product components (formally these graphs are Petri Nets). These models can be queried and in silico experiments can be performed to study the effects of perturbations on these networks. Using PLA a biologist can:

- ask for a list of dishes available for study, and modify or create dishes;
- display the network of signaling reactions for a specified model;

- formulate and submit queries to find pathways, for example, activating one protein without activating a second protein, or exhibiting a phenotype signature such as apoptosis;
- compare two pathways;
- find knockouts—proteins whose omission prevents reaching a specified state;
- incrementally explore network connections to given rules or components;
- visualize gene expression data in the context of a network (by coloring the coded proteins according to expression level)

PLA, sample models, tutorial material, papers and presentations are available from the Pathway Logic web site, `http://pl.csl.sri.com/`.

4 Introduction to Formal Executable Specification and Maude

As mentioned in Section 3, Pathway Logic models of biological processes are developed using the Maude system, a formal language and tool set based on rewriting logic. Rewriting logic [33] is a logical formalism that is based on two simple ideas: states of a system are represented as elements of an algebraic data type, specified in an equational theory, and the behavior of a system is given by local transitions between states described by *rewrite rules*. An equational theory specifies a data type by declaring constants and constructor operations that build complex structured data from simpler parts. Functions on the specified data types are defined by *equations* that allow one to compute the result of applying the function. A term is a variable, a constant, or application of a constructor or function symbol to a list of terms. A specific data element is represented by a term containing no variables. Assuming the equations fully define the function symbols, each data element has a canonical representation as a term containing only constants and constructors. The canonical representation is obtained by using the equations to rewrite a data term until only constants and constructor symbols remain. For example the natural numbers are constructed from the constant 0 by application of the successor function $s(0), s(s(0)) \ldots$ (usually written as $1, 2, \ldots$). The plus function, $+$, can be defined by two equations: $n + 0 = n$ and $n + s(m) = s(n) + m)$, where n and m are variables standing for arbitrary numbers. Using these equations we can compute the canonical form (value) of $s(s(0)) + s(s(s(0)))$ $(2 + 3)$. An equation is applied to a term by matching the left hand side of the equation to a subterm and replacing that subterm the the corresponding righthand side of the equation. The second equation matches $s(s(0)) + s(s(s(0)))$ with $n := s(s(0))$ and $m := s(s(0))$ and the result of rewriting with this equation is $s(s(s(0))) + s(s(0))$. Using the second equation twice more we get $s(s(s(s(0)))) + s(0)$ and then $s(s(s(s(0))))) + 0$. Now we apply the first equation to obtain $s(s(s(s(s(0)))))$ (i.e. 5).

One data type might be a subtype (subsort / subset) of another. For example the non-zero numbers are a subset of all numbers. Elements of one data type might consist of lists or multisets of elements from another type. For example a system might be represented by a set of pairs such as `{(A,2) (B,5) (C,0)}`.

A rewrite rule has the form $t \Rightarrow t'$ *if* c where t and t' are patterns (terms possibly containing place holder variables) and c is a condition (a boolean term). Such a rule applies to a system in state s if t can be matched to a part of s by supplying the right

values for the place holders, and if the condition c holds when supplied with those values. In this case the rule can be applied by replacing the part of s matching t by t' using the matching values for the place holders in t'. The process of application of rewrite rules generates computations (also thought of as deductions). In the case of biological processes these computations correspond to pathways. Note that rewriting with rules is similar to rewriting with equations, in that we match the lefthand side and replace the matched subterm by the instantiated righthand side. The difference is in the way the rewriting is used. Equations are used to define functions by providing a means of compution the value of a function application. This means that the equations of an equational theory should give the same result independent of the order in which they are applied. Furthermore, equational rewriting should terminate. In contrast, rules are used to describe change over time, rather than computing the value of a function. They often describe non-deterministic possibly infinite behavior.

To summarize, a rewriting logic specification naturally has two parts: an equational part that specifies data types and functions on these types, and a rules part, specifying how systems may evolve. To query the specification, one may further specify one or more terms representing systems (initial states) of interest, to be analyzed using formal tools such as execution, search and model checking.

Maude is a language and tool based on rewriting logic `http://maude.cs.uiuc.edu`. Maude provides a high performance rewriting engine featuring matching modulo associativity, commutativity, and identity axioms; and search and model-checking capabilities. Thus, given a specification S of a concurrent system, one can execute S to find one possible behavior; use search to see if a state meeting a given condition can be reached; or model-check S to see if a temporal property is satisfied, and if not to see a computation that is a counter example.

In the following we use a simple example to introduce Maude notation and give some intuition about how to represent and analyze the structure and behavior of concurrent systems using Maude. We call the example *Magic Marbles*. In the world of magic marbles, a marble can be plain or have some magical potential: positive, negative, or (positive or negative) activator. A positive (resp. negative) activator marble can give positive (resp. negative) potential to a plain marble. When it does so, it changes parity and becomes a negative (resp. positive) activator. If a marble with negative potential contacts a marble with positive potential the potential is cancelled and they both become plain. A marbles world consists of a collection (formally a multiset) of marbles interacting according to the laws described above.

We formalize the marbles world in Maude by defining three modules. The modules `MAGIC-MARBLES-DATA` and `MAGIC-MARBLES-STATE` specify data types representing marbles and marbles world states. The module `MAGIC-MARBLES-RULES` specifies the rules governing magic marble behavior. The modules `MAGIC-MARBLES-DATA` and `MAGIC-MARBLES-STATE` are functional modules specifying an equational theory. They form the equational part of the marbles world specification. The module `MAGIC-MARBLES-RULES` is a system module forming the rules part.

A Maude module begins with the keyword `fmod` (a functional module, specifying one or more data types) or `mod` (a system module, with rules specifying system behav-

ior), followed by the module name, and ends with a corresponding keyword `endfm`, or `endm`, respectively.

The module `MAGIC-MARBLES-DATA` begins by declaring a sort `Marble`, the data type consisting of all marbles, and a subsort (think subset or subtype) `PlainMarble`, plain marbles. This is followed by an `ops` declaration naming several specific plain marbles, for example a red marble `redM`. Next a sort `MagicMarble` of marbles with magical potential is declared. It is also a subsort of `Marble`. Magic potential is represented abstractly by a sort `Potential`. Four different potentials are defined (the second `ops` declaration):

- `+`, `-` represent positive and negative potentials
- `*`, `@` represent the potential of an activator marble to generate a positive or negative potential respectively.

A marble with potential p is constructed by annotating a plain marble m with p, written `[m | p]`.

```
fmod MAGIC-MARBLES-DATA is
  sort Marble .
  sort PlainMarble . subsort PlainMarble < Marble .
  ops redM blueM greenM purpleM whiteM blackM
        : -> PlainMarble [ctor] .

  sort MagicMarble .  subsort MagicMarble < Marble .

  sort Potential .
  ops + - * @ : -> Potential [ctor] .
  op `[_|_`] : PlainMarble Potential -> MagicMarble [ctor] .
endfm
```

The declaration beginning `op `[_|_`]` is an example of *mixfix* syntax, where an operator is collection of symbols including _s that are place holders for the arguments. In this case there are two arguments, the first of sort `PlainMarble`, the second of sort `Potential`. (The backquotes are to ensure the brackets are parsed as part of the operator symbol.)

As examples, we have

- `[whiteM | *]` a white marble with positive activator potential
- `[whiteM | @]` a white marble with negative activator potential
- `[greenM | +]` a green marble with positive potential

The module `MAGIC-MARBLES-STATE` extends `MAGIC-MARBLES-DATA` (using the inclusion statement beginning `inc`) specifying a sort `Mix` of multisets of marbles. (A multiset or bag is a collection of elements where the number of occurrences of an given element matters, but the order does not.) The constant `none` represents the empty multiset, and multiset union is represented by a binary operator, _ _, that is associative and commutative with identity `none` (ACI).

```
fmod MAGIC-MARBLES-STATE is
  inc MAGIC-MARBLES-DATA .
sort Mix .
subsort Marble < Mix .
op none : -> Mix [ctor] .
op _ _ : Mix Mix -> Mix [assoc comm id: none] .
endfm
```

Thus `redM blueM [whiteM | *]` is a mix of three marbles, two plain and one with positive activating potential.

The syntax, _ _, is an example of empty syntax, a special case of mixfix syntax in which application of the operator is juxtaposition. The attribute `[assoc comm id: none]` at the end of the declaration specifies the ACI property, which axiomatizes the notion of multiset. Associative means that the two ways of joining three elements with two applications of the operator give the same multiset. For example,

```
((redM blueM whiteM) = (redM (blueM whiteM)) .
```

This means that parentheses can be omitted, grouping of arguments doesn't matter.

Commutative means that permuting the order of arguments gives the same result. For example,

```
redM blueM = blueM redM .
```

The "identity `none`" property means that adding `none` to the mix does not change the mix. For example,

```
redM blueM none = blueM redM .
```

The module `MAGIC-MARBLES-RULES` specifies how marbles interact using three rules. A rule begins with the key word `rl` followed by the rule label enclosed in `[]`s. The lefthand side (premiss) and righthand side (conclusion) of a rule are separated by the `=>` sign. The rules labeled `plus` and `minus` formalize the informal statements "A positive (resp. negative) activator marble can give positive (resp. negative) potential to a plain marble." "a marble with the `*` potential is a positive activator", and "a marble with the `@` potential is a negative activator." The rule labeled `cancel` formalizes the statement "If a marble with negative potential contacts a marble with positive potential the potential is cancelled and they both become plain".

```
mod MAGIC-MARBLES-RULES is
  inc MAGIC-MARBLES-STATE .
  vars pm0 pm1 : PlainMarble .
  rl[plus]:  pm0 [ pm1 | *] => [ pm0 | + ] [ pm1 | @ ] .
  rl[minus]:  pm0 [ pm1 | @] => [ pm0 | - ] [ pm1 | * ] .
  rl[cancel]:  [pm0 | +] [ pm1 | -] =>  pm0   pm1 .
endm
```

The variables `pm0`, `pm1` stand for arbitrary plain marbles. The change of activator potential in rules `plus`,`minus` formalize "When it does so, it changes parity and becomes a negative (resp. positive) activator."

Now we have an executable formal specification of magical marbles. What can we do with it? The simplest thing to do is to pick a starting state and use the rewrite and continue commands to watch it run. The command `rew [1] t .` rewrites the term `t` one step. The command `cont 1 .` continues rewriting one more step. Suppose we have an initial state `[whiteM | *] redM blueM` with two plain marbles and a positive activator. The rule `plus` applies to the subterm `[whiteM | *] redM` matching `pm0` to `redM` and `pm1` to `whiteM` (using multiset matching where the order of multiset elements doesn't matter), and replacing the matched subterm by the corresponding instance of the rule's righthand side, `[redM | +] [whiteM | @]`.

```
Maude> rew [1] [whiteM | *] redM blueM .
result Mix: blueM [redM | +] [whiteM | @]
```

Rewriting can be continued by rewriting with the `minus` rule and then by the `cancel` rule.

```
Maude> cont 1 .
result Mix: [redM | +] [blueM | -] [whiteM | *]   *** by [minus]
Maude> cont 1 .
result Mix: redM blueM [whiteM | *]               *** by [cancel]
```

This computation could be continued as many steps as you like. There are many other possible computations starting from our initial state, each making different choices of which plain marble to use in the `plus` step.

The command `search [n] istate =>+ pattern` searches the states reachable from `istate` for states matching pattern, stopping when it has found `n` solutions, or it runs out of states. It starts by finding all states that result from application of one rewrite rule, the finding all states that result from application of one rewrite rule to each of these states, and so on. Using the search command we can ask whether it is possible to make the red and blue marbles simultaneously positive, starting from the our initial state.

```
Maude> search [1] [whiteM | *] redM blueM
         =>+ M:Mix [redM | + ] [blueM | + ] .
Maude> no Solution .
```

The answer is no. If we add another positive activator then getting two positive marbles is easy.

We can make the structure of magic marble states a little more interesting by introducing boxes that can contain marbles, or other boxes, and such that under certain conditions a marble can enter or leave a box. Specifically, only plain marbles can enter or leave a box. For a marble to enter a box, the box must contain a negative activator, while for a marble to leave a box, there must be a positive activator outside the box.

The module `BOXED-MARBLES-DATA` extends `MAGIC-MARBLES-STATE` with new sorts `BoxId` (box identifier) and `Box`. The subsort declaration `Box < Mix` says that boxes can appear in mixes. A box has an identifier and contains a mix. For example {`B0 | redM blueM`} is a box with identifier `B0` and contents `redM blueM`. For convenience we define a constant `bMix` to be a box with identifier `B0` that contains a white

positive activator marble, and two nested boxes. This is done by an equating `bMix` to a term representing the described box. In Maude an equation is introduced with the keyword `eq`, followed lefthand and righthand side terms separated by the equality sign `=`.

```
fmod BOXED-MARBLES-DATA is
  inc MAGIC-MARBLES-STATE .
  sorts Box BoxId .
  subsort Box < Mix .
  ops B0 B1 B2 : -> BoxId .
  op `{_|_`} : BoxId Mix -> Box [ctor] .

**** sample box mix
  op bMix : -> Mix .
  eq bMix = { B0 | [whiteM | *]
                  {B1 | redM blueM [blackM | @]}
                  {B2 | greenM purpleM [blackM | @]}} .
endfm
```

The module `BOXED-MARBLES-RULES` gives the rules for moving marbles in and out of boxes.

```
mod BOXED-MARBLES-RULES is
  inc BOXED-MARBLES-DATA .
  inc MAGIC-MARBLES-RULES .

  vars pm0 pm1 : PlainMarble .
  var mx : Mix .
  var bid : BoxId .

  rl[in]:
   { bid | mx [ pm0 | @] } pm1 => { bid | mx [ pm0 | @] pm1 } .
  rl[out]:
   { bid | mx  pm1 } [ pm0 | *] => { bid | mx } [ pm0 | *] pm1 .
endm
```

To reason about location of marbles we define a predicate `inBox` which, given a mix and a box identifier, checks whether a box with that identifier contains the given mix. The term `contains(mx,mx0)` evaluates to true if every element of `mx0` is in `mx`. For example `inBox(bMix,B1,redM) = true` and `inBox(bMix,B0,whiteM) = false`. The function `contains` is defined by two equations.

```
  var mx mx0 mx1 : Mix .
  op contains : Mix Mix -> Bool .
  eq contains(mx0 mx1, mx0) = true .
  eq contains(mx,mx0) = false [owise] .
```

The first equation uses multiset matching to identify the true case. The pattern `mx0 mx1` matches a mix term `mx` just if `mx` contains `mx0` (`mx1` is the rest of the mix). In this case `contains(mx,mx0)` will rewrite to `true`. An equation, such as the second equation

above, tagged with the [owise] attribute, can only be used if no other equation applies. In our case the second equation applies to a term contains(mx,mx0) just if mx does not contain mx0. In this case the term rewrites to false.

The equations for inBox describe the following algorithm: select a box in the outer mix; if the box has the given identifier, then check whether the box contains the target mix, otherwise look for nested boxes and look in the rest of the outer mix. Again the [owise] tagged equation is used for cases in which inBox can not be true, such as when the mix is empty or contains no boxes.

```
op inBox : Mix BoxId Mix -> Bool .
var mx mx0 mx' : Mix .
var bid bid' : BoxId .
eq inBox({bid' | mx} mx', bid,mx0) =
   (if bid == bid' and contains(mx,mx0)
    then true
    else (if inBox(mx',bid,mx0)
          then true
          else inBox(mx,bid,mx0)
          fi) fi) .
eq inBox(mx,bid,mx0) = false [owise] .
```

Suppose we want to know if, in bMix, the marbles in boxes B1 and B2 can change places. This can be answered by searching, using the inBox predicate. There is only one solution, shown below.

```
Maude> search [1] bMix =>+ M:Mix such that
  inBox(M:Mix,B1,greenM purpleM)  and inBox(M:Mix,B2,redM blueM) .

Maude> Solution 1 (state 1387)
M:Mix --> {B0 | [whiteM | *]
                {B1 | greenM purpleM [blackM | @]}
                {B2 | redM blueM [blackM | @]}}
```

An alternative to search is to use model checking. A model checker checks properties of the possible computations starting from a given initial state. The properties are expressed in Linear Temporal Logic (LTL). The module MARBLES-MC defines model checking states for Magic Marbles and defines a state proposition based on the inBox predicate. The {_} operator encapsulates a mix, thus defining a boundary. Propositions are defined using the relation, {mx}|= prop, read the mix mx satisfies the proposition prop. A predicate on mixes (and other arguments) can easily be turned into a proposition, by defining a corresponding operator that maps the remaining arguments to the sort Prop. For example, inBoxP(bid,mx0) is the proposition corresponding the predicate inBox(mx,bid,mx0) and {mx} satisfies inBoxP(bid,mx0) if inBox(mx,bid,mx0).

```
mod MARBLES-MC is
  inc BOXED-MARBLES-RULES .
  inc MODEL-CHECKER .
```

```
  op `{_`} : Mix -> State .
  op inBoxP : BoxId Mix -> Prop .

  vars mx  mx0 : Mix .
  var bid : BoxId .
  eq {mx} |= inBoxP(bid,mx0) = inBox(mx,bid,mx0) == true .
endm
```

If `P` is a state proposition, then the property `[]P` says that every state in a computation satisfies `P` and `[]~P` says that no state in a computation satisfies `P`. Thus to see if a state satisfying `P` can be reached, we can use the Maude model checker to evaluate `modelCheck({mx},[]~P)`. If a state can be reached satisfying `P`, the model checker will return a counter-example showing the transitions (state and rule label) of a computation containing such a state. For example, we can find a way to move `redM` from box `B1` to `B2` as follows.

```
red modelCheck({bMix}, []~inBoxP(B2,redM)) .
result ModelCheckResult: counterexample(
  {{{B0 | [whiteM | *]
     {B1 | redM blueM [blackM | @]}
     {B2 | greenM purpleM [blackM | @]}}},
  'out}
  {{{B0 | redM [whiteM | *]
     {B1 | blueM [blackM | @]}
     {B2 | greenM purpleM [blackM | @]}}},
  'in}
  {{{B0 | [whiteM | *]
     {B1 | blueM [blackM | @]}
     {B2 | redM greenM purpleM [blackM | @]}}},
  ...}
  ...
```

The first transition applies the `out` rule to move `redM` out of box `B1`. The second transition applies the `in` rule to move `redM` into box `B2`. The `...`s indicate that the counter example continues. This is an artifact of the model checker requirement that counter examples are infinite. The remainder of the computation is building a loop and can be ignored for our purposes. The above is a reachability question that can also be answered by search, although it is harder to extract the computation, it can be done.

5 Signal Transduction: What to Model

We will focus on modeling signal transduction networks. The Wikipedia article on signal transduction `http://en.wikipedia.org/wiki/Signal_transduction` contains an excellent overview and is a good place to start reading to learn more [**?**].

To illustrate key signaling concepts and modeling ideas, we will use epidermal growth factor receptor (EgfR) signaling, which regulates growth, survival, proliferation, and differentiation in mammalian cells. In particular we will look at the MAPK

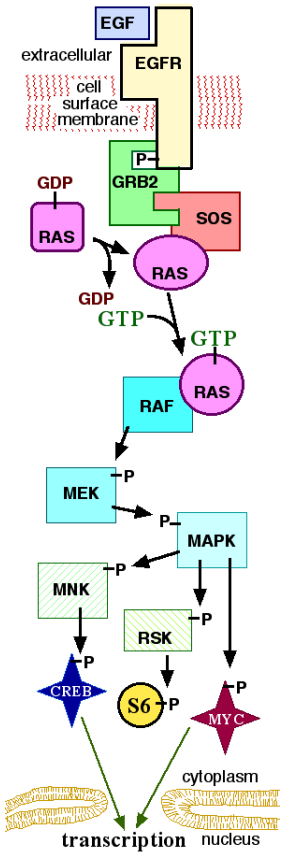


Fig. 1. Cartoon of Egf stimulated MAPK Pathway.

(Mitogen-Activated Protein Kinase) pathway [46, 13, 27, 25]. Figure 1 shows the cartoon drawing of the MAPK pathway (taken from Wikipedia). The pathway is also often represented as a linear sequence of events:

$$\text{Egf} \rightarrow \text{EgfR} \rightarrow \text{Grb2} \rightarrow \text{Sos1} \rightarrow \text{Ras} \rightarrow \text{Raf1} \rightarrow \text{Mek} \rightarrow \text{Erk}$$

Here is a biologist style explanation of what this picture or sequence represents. The explanation uses PL terminology, with corresponding names from the figure in parentheses.

"In this canonical pathway, Egf (EGF) binds to the Egf receptor (EgfR) and stimulates its protein tyrosine kinase activity to cause auto-phosphorylation, thus activating EgfR. Next, the adaptor protein Grb2 (GRB2) and the guanine nucleotide exchange factor Sos1 (SOS) are recruited to the membrane and bind to the activated EgfR. The Sos1-containing EgfR complex activates a Ras family GTPase, and the activated Ras protein

activates Raf1, a member of the RAF serine/threonine protein kinase family. Raf1 then activates the protein kinase Mek1/2 (MEK), which then activate Erk1/2 (MAPK)."

Even without understanding the terminology, it should be clear that much of the actual model remains in the mind of the biologist and is not captured by the picture. In the remainder of this section we will look at the steps leading to activation of Ras in some detail, to explain the terms used in the biologists style description of the pathway, and introduce the concepts and mechanisms that we want to model. A PL model of this pathway is discussed in Section 6, and the full PL model of Egf stimulation is discussed in Section 8.

One of the first things to notice is that a protein may have many names, depending on who is talking about it. The simplest variation is capitalization. PL uses the convention that the name of a protein is capitalized like a proper name, while in the figure protein names are all-caps. The numbers in PL names make explicit the fact that there are numbered variants of a protein, for example Sos1 (as opposed to Sos2) or Mek1/2 (meaning either Mek1 or Mek2). The figure uses a more abstract representation. The figure uses MAPK, which abbreviates Mitogen-Activated Protein Kinase, rather than Erk (or Erk1/2). Sometimes a protein name is an acronym of a name that is related to the proteins function or how it was discovered. For example, Egf abbreviates "Epidermal Growth Factor", indicating that it is a protein involved in signaling related to decisions about growth. EgfR (Epidermal growth factor receptor) is also known as ErbB1 or HerbB1. Grb2, abbreviates "Growth factor receptor-bound protein 2" and Sos1 abbreviates "Son of sevenless 1" (first discovered in Drysophila and named for its connection to the tyrosine kinase receptor "sevenless".)

One way to determine if two names refer to the same protein is to link the name to a database entry that is accepted as a standard (of course there are several standards). PL links all protein names to their Swiss-Prot entry. Swiss-Prot is a manually curated biological database of protein sequences. In addition to the protein sequence, the Swiss-Prot entry for a protein includes synonyms, literature references, information about function, location, interactions, links to databases containing special purpose information such a protein functional domains and gene annotations. For example the Swiss-Prot name for EgfR is `EGFR_HUMAN` and the Swiss-Prot entry for EgfR can be found at `http://www.expasy.ch/cgi-bin/niceprot.pl?P00533` where `P00533` is the EgfR Swiss-Prot accession number. Non-protein signaling molecules, such as sugars or lipids, also have many names. To reduce ambiguity, PL names of this molecules are linked to entries in standard data bases such as KEGG where chemical structure and other information can be found.

Adaptor proteins play key roles in signaling pathways. They serve to hold interacting proteins in spatial configurations that make interaction possible. In the case of the adaptor Grb2, this is represented in the figure by nitches in the Grb2 icon so that it brings EgfR and Sos1 together enabling Sos1 to carry out its function to activate Ras.

GTP (Guanosine triphosphate) is an important molecule in metabolism, protein synthesis, and signal transduction. In our example, binding of GTP activates Ras, and subsequent hydrolysis of the bound GTP to GDP and phosphate inactivates Ras, thus acting as a kind of switch. The switch can be turned on by proteins, such as Sos1, known as *guanine nucleotide exchange factors* (GEFs), and can be turned off by GTPase-

activating proteins (GAPs) that accelerate hydrolysis of GTP to GDP (guanosine diphosphate). GEFs act by binding Ras-GDP forcing it to release the bound GDP. Once released from the GEF, Ras quickly binds fresh GTP from the cytosol. Ras is called a GTPase because of its ability to bind and hydrolize GTP.

The notion of location plays an important role in cellular signaling. Proteins need to be co-located to interact. *Compartments* in a cell serve to collect interacting groups of proteins (and other molecular components). Each compartment has a membrane and an interior and compartments may be nested. A cell is itself a compartment. Its membrane is called the *cell membrane* and its interior is called the *cytoplasm*. In the cytoplasm there are many other compartments, most importantly, the nucleus, where the cell's DNA resides. In the figure 1 we can trace the Egf signal from the outside of the cell, through the cell membrane, traversing the cytoplasm and eventually reaching the nucleus. In the process Grb2 and Sos1 are *recruited* from the cytoplasm to the interior of the membrane, to bind to the inner part of EgfR.

Proteins and other molecules are categorized according to their function. A *receptor* is a protein that receives a signal by recognizing and binding to a signaling molecule called a *ligand*. This results in a *complex* in which the two proteins are linked together, likely causing a change in shape and activity of the receptor thus initiating a signaling process. The first step of our example pathway is activation of EgfR. The EgfR protein is a receptor that has three regions: one that sticks outside the cell, one traversing the cell membrane, and one that sticks into the cytoplasm. Thus, it receives signals from outside the cell and transmits them to the inside. In our example, the ligand Egf binds to external portion of Egf and then the Egf-EgfR complex dimerizes (pairs with another Egf-EgfR complex).

A signal is propagated by changes in the *state* of involved proteins. One way to change state is complexing with other proteins. Another important form of state change is *post-translational modification*. This is a change in the chemical structure of a protein after its translation. Phosphorylation, attaching a phosphate group to one of the amino acid sites of a protein, is an example of post-translational modification. A *kinase* is a protein that facilitates *phosporylation*. Usually kinases have specific proteins or classes of proteins as targets and act on specific amino acid sites. Dually a *phosphatase* facilitates removal of a phosphate group. Phosphorylation (de-phosphorylation) changes the state of a protein, and is one of the ways that signals get propagated (or blocked). In our example, EgfR is not only a receptor, it is a kinase and capable of phosphorylating other EgfRs. When the Egf-EgfR homo-dimer forms EgfR *auto-phosphorylates* and becomes active.

Now we can explain the phrase "mitogen-activated protein kinase" (MAPK) pathway. A *mitogen* is a molecule that signals a cell to trigger mitosis and thus commence cell division. A MAPK pathway activates MAPK proteins such as Erk, which propagate the mitotic signal to the nucleus. We note that Mek is a kinase kinase, (also called MAPKK or MAPK Kinase), as it phosphorylates the kinase Erk. Continuing the trend, Raf1 is a kinase kinase kinase also called MAPKKK.

6 Building a Pathway Logic Knowledge Base

Now we describe a small PL knowledge base, SmallKB, that represents initial signaling events in response to Epidermal Growth Factor (Egf) stimulation discussed in Section 5. The full Egf stimulation model is discussed in Section 8.

Recall that a rewriting logic specification has two parts: an equational part specify structure and static properties of system states, and a rules part specifying system behaviors. A PL Model system is structured in four layers: (1) sorts and operations, (2) molecular components, (3) rules, and (4) queries. Layers 1-3 make up a Pathway knowledge base (PL KB) with layers 1 and 2 being the equational part.

6.1 The Equational Part

The *sorts and operations* layer declares the main sorts, subsort relations, and operators to construct representations of cellular states. The sorts of entities include Chemical, Protein, Complex, and Location (position within cellular compartments), and Cell. These are all subsorts of the sort, Soup, that represents 'liquid' mixtures, as multisets of entities. The sort Modification is used to represent post-translational protein modifications. They can be abstract, to specify that a protein is activated, bound, or phosphorylated, or more specific, for example, phosphorylation at a particular site. Modifications are applied using the operator `[_-_]`. (Note the similarity to the annotation of marbles with potential in section 4.) For example, the term `[Raf1 - act]` represents `Raf1` in an activated state, and `[Hras - GTP]` represents the protein `Hras` in its "on" state (loaded with GTP). (Hras is a specific member of the Ras family.) The term `[Gab1 - Yphos]` represents `Gab1` phosphorylated on a tyrosine site while `[Gab1 - phos(Y 627)]` represents `Gab1` phosphorylated on tyrosine 627. Complex formation is represented by the operation `(_:_)`. For example, the term `(Egf : [EgfR - act])` represents the complex resulting from binding of Egf to EgfR and subsequent activation of EgfR. A cell state is represented by a term of the form

```
[cellType | locs].
```

The symbol `cellType` specifies the type of cell, for example Macrophage or Fibroblast. The symbol `Cell` is used to indicate an unspecified cell type. The symbol `locs` represents the contents of a cell organized by cellular location. Each location is represented by a term of the form `{ locName | components }` where `locName` identifies the location, for example `CLm` for cell membrane, and `components` stands for the mixture of proteins and other compounds in that location. For example,

```
[Cell | {CLm | EgfR  PIP2}
        {CLi | [Hras - GDP]  Src}
        {CLc | Gab1 Grb2 Pi3k Plcg Sos1}]) .
```

represents a generic cell with three locations: the membrane (location tag `CLm`) contains `EgfR` and a chemical `PIP2` (see below); the inside of the membrane (location tag `CLi`) contains `Hras` loaded with `GDP` and `Src`; and the cytoplasm (location tag `CLc`) contains `Gab1`, `Grb2`, `Pi3k`, `Plcg`, and `Sos1`.

The *components* layer specifies particular entities (proteins, genes, chemicals) and introduces additional sorts for grouping proteins in families. For example `ErbB1L` is a subsort of `Protein` whose elements are ErbB1 (`EgfR`) ligands. Components are declared as constants, giving their sort, and metadata giving synonyms and standard names that can be linked to databases providing other information. For example the epidermal growth factor `Egf` with sort `ErbB1L`, and metadata giving its HUGO and Swiss-Prot names, its Swiss-Prot accession number, and its *category*. in addition to two synonyms.

```
op Egf : -> ErbB1L [metadata "(\
  (spname EGF_HUMAN)\
  (spnumber P01133)\
  (hugosym EGF)\
  (category Ligand)\
  (synonyms \"Pro-epidermal growth factor precursor, EGF\" \
       \"Contains: Epidermal growth factor, Urogastrone \"))"] .
```

Similarly, `EgfR` is declared simply to be a protein.

```
op EgfR : -> Protein [metadata "(\
  (spname EGFR_HUMAN)\
  (spnumber P00533)\
  (hugosym EGFR)\
  (category Receptor)\
  (synonyms \"Epidermal growth factor receptor precursor\" \
       \"Receptor tyrosine-protein kinase ErbB-1, ERBB1 \"))"] .
```

PIP2 is a chemical (a lipid) residing in the membrane. Its phosphorylated form, PIP3, plays an important role in a number of signaling pathways, either directly or through its cleavage products. Chemicals have metadata linking them to a KEGG database entry, where much information can be found.

```
op PIP2 : -> Chemical [metadata "(\
   (category Chemical)\
   (keggcpd C04569)\
   (synonyms \"Phosphatidylinositol-4,5P \" ))"] .
```

The *rules* layer is the heart of a PL KB. It contains rewrite rules specifying individual reaction steps. In the case of signal transduction rules represent processes such as activation, phosphorylation, complex formation, or translocation. The rules layer is discussed in Section 6.2 below.

The *queries* layer specifies initial states (called dishes) to be studied. Initial states can be thought of as describing "in silico experiments". They represent in silico Petri dishes containing a cell and ligands of interest in the supernatant. An initial state is represented by a term of the form

```
PD(out cell)
```

where `cell` represents a cell state and `out` represents a soup of ligands and other molecular components in the cells surroundings. In fact a dish can contain many cells, however the current PL analysis tools only treat single cells. For example an initial state to study Ras activation in `SmallKB` is given by the dish term

```
op rasDish  : -> Dish .
eq rasDish =
     PD(Egf [Cell | {CLm | EgfR  PIP2}
                    {CLi | [Hras - GDP]  Src}
                    {CLc | Gab1 Grb2 Pi3k Plcg Sos1}]) .
```

representing a dish containing `Egf` and the cell discussed above.

6.2 The Rules Part

PL rules are curated from the literature, and each rule has associated evidence items describing experimental data that serve as evidence for the rule. Discussion of evidence is beyond the scope of the present document, as it requires some understanding of experimental methods to be meaningful. The rules for the initial response to Egf signaling closely parallel the biologist's informal explanation of Figure 1 given in Section 5.

```
rl[1.EgfR.act]:
  ?ErbB1L:ErbB1L
  [?CellType:CellType | ct {CLm | clm EgfR}]
  =>
  [?CellType:CellType | ct
    {CLm | clm ([EgfR - act] : ?ErbB1L:ErbB1L)} ] .
```

Rule 1 (label `1.EgfR.act`) describes the binding of an ErbB1 ligand to EgfR. The term `?ErbB1L:ErbB1L` is a variable that matches any ErbB1 ligand, for example `Egf`, and `?CellType:CellType` is a variable that matches any cell type. `{CLm | clm EgfR}` matches any cell membrane location that contains `EgfR`, since `clm` is a variable that will match the rest of the membrane contents. Thus the left hand side subterm

```
[?CellType:CellType | ct {CLm | clm EgfR}]
```

matches any cell that contains `EgfR` in the cell membrane, since the variable `ct` will match any additional locations. For example, it matches the initial state `rasDish` with

```
?CellType:CellType := Cell
?ErbB1L:ErbB1L := Egf
clm := PIP2
ct := {CLi | [Hras - GDP]  Src} {CLc | Gab1 Grb2 Pi3k Plcg Sos1}
```

and the result of rewriting `rasDish` with rule 1 is

```
PD([Cell |
   {CLm | ([EgfR - act] : Egf)  PIP2}
   {CLi | [Hras - GDP]  Src}
   {CLc | Gab1 Grb2 Pi3k Plcg Sos1}]) .
```

which is obtained by instantiating the rules right hand side using the variable bindings from the left hand side match.

Once the receptor is activated it can recruit `Grb2` to the membrane interior. This is describe by the rule labeled `5.Grb2.reloc`.

```
rl[5.Grb2.reloc]:
  {CLm | clm ([EgfR - act] : ?ErbB1L:ErbB1L)  }
  {CLi | cli                   }
  {CLc | clc Grb2              }
  =>
  {CLm | clm ([EgfR - act] : ?ErbB1L:ErbB1L) }
  {CLi | cli [Grb2 - reloc]    }
  {CLc | clc                   } .
```

Notice that on the left, `Grb2` is in the `CLc` location (cytoplasm) while on the right it is in `CLi` location. The modification `reloc` makes the change in location explicit. It is not strictly necessary, but makes the changes easier to follow. Continuing the rewriting of `rasDish` with rule `5.Grb2.reloc` we get

```
  PD([Cell |
     {CLm | ([EgfR - act] : Egf)  PIP2}
     {CLi | [Hras - GDP]  Src [Grb2 - reloc]}
     {CLc | Gab1 Pi3k Plcg Sos1}]) .
```

Now `Sos1` can be recruited to the membrane complex by binding to `Grb2`. This is described by rule `13.Sos1.reloc`. Note that in this particular representation the complex formation is abstracted to colocation. We could also make the complex explicit if it were needed for some analysis.

```
rl[13.Sos1.reloc]:
  {CLi | cli [Grb2 - reloc]               }
  {CLc | clc Sos1                         }
  =>
  {CLi | cli [Grb2 - reloc] [Sos1 - reloc] }
  {CLc | clc                               } .
```

The resulting state is

```
  PD([Cell |
     {CLm | ([EgfR - act] : Egf)  PIP2}
     {CLi | [Hras - GDP]  Src [Grb2 - reloc] [Sos1 - reloc]}
     {CLc | Gab1  Pi3k Plcg }]) .
```

In the next section we will see how to transform the Maude terms in to a graph representation that makes it easier to visualize and understand reaction networks and their evolution. In particular, a graphical representation of the above three step computation is shown below in Figure 3.

7 Computing with a PL KB

The Pathway Logic Assistant (PLA) provides interactive graphical access to a PL knowledge base. For this purpose, rule sets and computations are represented using Petri nets [39, 38, 48], which have a natural graphical representation, additionally, there are very efficient tools for analyzing the Petri net models generated by PLA. (See Section 2 for discussion of other uses of Petri nets in systems biology.)

7.1 PL Petri nets

Petri Nets were invented to model execution of concurrent processes and thus are nicely suited to modeling signals propagating through a cell. A Pathway Logic Petri net (simply called Petri net, in what follows) can be thought of as graph with two kinds of nodes: rule nodes (shown as squares) and occurrence nodes (shown as ovals). Rule nodes, called transitions in the Petri net community, represent reactions, and occurrence nodes, called places in the Petri net community, represent reactants, products, or modifiers. Occurrences can be thought of as atomic propositions asserting that a protein (in a given state) or other component (small molecule, complex, ...) occurs in a given compartment. In this view, rules are logical implications.

An occurrence oval is labeled by a string representation of the corresponding Maude term. For example the string representation of `Efg` outside a cell is `Egf-Out` and `Egf:EgfR-act-CLm` is the string representation of `Egf : [EgfR - act]` in the cell membrane. The reactants of a rule are the occurrences connected to the rule by arrows from the occurrence to the rule. The products of a rule are the occurrences connected to the rule by arrows from the rule to the occurrence. The modifiers of a rule (enzymes and other components that must be present but are unchanged) are the occurrences connected to the rule by a dashed arrow. For example, the Petri net representation of the rule for recruitment of `Sos1` is shown in Figure 2.

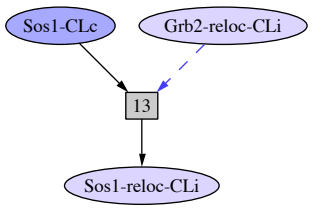


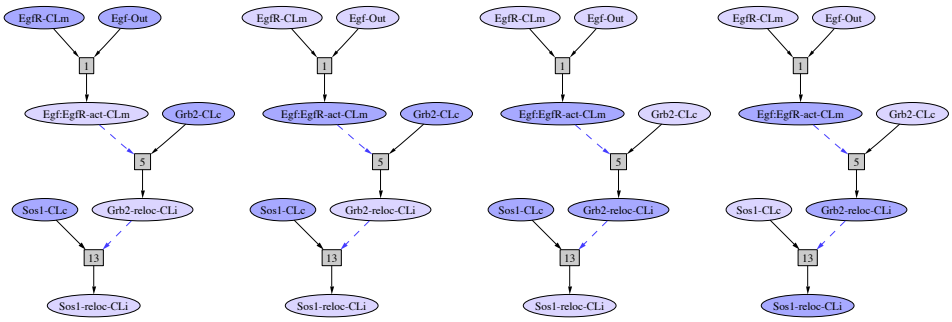
Fig. 2. Petri net transitions for rule 13.Sos1.reloc

The rule is represented by the rectangle labeled `13` (short form of `13.Sos1.reloc`). The reactant `Sos1` in the cytoplasm is represented by the oval labeled `Sos1-CLc` with an arrow from the oval to the rule rectangle. The product `[Sos1 - reloc]` at the membrane interior is represented by the oval labeled `Sos1-reloc-CLi` with an arrow from

the rule rectangle to the oval. `[Grb2-reloc]` drives the reaction but is not changed (at our level of representation), thus it is represented by the oval labeled `Grb2-reloc-CLi` with a dashed arrow from the oval to the rule rectangle.

A set of Petri net rules corresponding to the rules of a PL knowledge base is called a transistion knowledge base (TKB). The analog of a PL dish is a PL Petri net state, which specifies which occurrences are present, that is, it specifies the state and location of each molecular component. Given a state, a Petri net rule is enabled if all of its occurrences connected by incoming arrows (reactants and modifiers) are present in the state. When an enabled rule fires, the reactant occurrences are removed from the state and the product occurrences are added. The modifier occurrences are left unchanged.

Corresponding to a PL model, a Petri net model consists of a set of rules (a TKB) and an initial state. To execute a Petri net model one puts tokens on the ovals corresponding to occurrences present in the initial state, and moves tokens as rules become enabled and fired. Figure 3 shows the execution of a Petri net model of the process that recruits `Sos1` to the membrane interior.



(a) initial state (b) step 1 (c) step 2 (d) step 3

Fig. 3. Execution of the Sos1 recruitment pathway

There are three rules (corresponding to the rewrite rules discussed in Section 6). Darker ovals represent occurrences that are present (marked with a token). Figure 3(a) shows the initial state with `Egf-Out`, `EgfR-CLm`, `Grb2-CLc`, and `Sos1-CLc` marked as initially present. The only rule enabled is rule 1. Figure 3(b) shows the result of firing rule 1, removing tokens from `Egf-Out` and `EgfR-CLm` and adding a token to `Egf:EgfR-act-CLm`. Now rule 5 is enabled and Figure 3(c) shows the result of firing rule 5. This enables rule 13 and Figure 3(d) shows the final state.

Starting with a PL KB, we convert it to a Petri net TKB, and convert dishes to occurrence sets, in a way that preserves the possible executions. We can then analyze models by finding subnets relevant to a desired state (goal), finding pathways reaching a goal, comparing subnets and/or pathways, finding knockouts (omissions from the initial state that prevent reaching a goal), or exploring a network of rules by incrementally adding connected components and rules to a given starting subnet. This is explained

in more detail in the following subsections. Details, including proof that the Petri net representation is equivalent to the rewriting logic representation, can be found in [52].

7.2 Converting a PL KB to a Petri net TKB

Transformation to Petri net representation accomplishes several things. One is support for graphical representation. Another is making things concrete. The full PL representation allows for rules that express families of reactions and for multiple cells and cell types. In PLA we restrict attention to systems with a single cell and for each variable that stands for a single component, we fix a specific (finite) set of components that can be values of that variable. For example, in the `SmallKB` knowledge base, there are two proteins that can be values of the variable `?ErbB1L:ErbB1L` of sort `ErbB1L`, namely `Egf` and `Tgfa`. Producing a Petri net representation of the Pathway Logic knowledge base proceeds in two steps. The first step transforms rules to occurrence form, by transforming the dishes or cells appearing in PL rules into occurrence sets. The second step instantiates remaining variables with known values.

Formally, an occurrence is a pair consisting of a component (a protein, possibly modified, a small molecule, or a complex) and a location name. For example, `<Egf, Out>` is an occurrence representing Efg outside a cell and `< Egf : [EgfR - act], CLm>` is a occurrence representing Egfr complexed with Egf and activated in the cell membrane. The left or right side of a rule is transformed by pairing each component with its location (the name of the enclosing location), dropping the location container, and dropping variables such as `ct` or `clm` that serve only to name location contents that are not important for the rule. Thus rule 1 (`1.EgfR.act`) becomes

```
rl[1.EgfR.act.pn]
    < ?ErbB1L:ErbB1L, Out >  < EgfR , CLm >
   =>
   < ?ErbB1L:ErbB1L : [EgfR - act], CLm >
```

When we instantiate remaining variables, we also convert rules (logical statements) into elements of a data type called `PNetTransition`. This allows us to compute with and reason about the Petri net rules directly. An interpreter to execute rules represented as data can be defined by a single rewrite rule. A pnet transition term has the form

```
pnTrans(label,iOccs,oOccs,bOccs)
```

where `label` is a quoted identifier, and `iOccs`, `oOccs` `bOccs` are multisets of occurrences: `iOccs` are the occurrences required and removed by the transition (connected to the rule by incoming arrows), `oOccs` are the occurrences produced by the transition (connected to the rule by outgoing arrows), and `bOccs` are the occurrences required but not removed by the transition (connected to the rule by dashed arrows). As an example, two pnet transitions are obtained by instantiating the occurrence form of rule 1, the first by instantiating the variable `?ErbB1L:ErbB1L` with `Egf`

```
pnTrans('1.EgfR.act,
         < Egf,Out > < EgfR,CLm >,
         < Egf :[EgfR -  act],CLm >,
         none)
```

and the second by instantiating with `Tgfa`.

```
pnTrans('1.EgfR.act#1,
        < EgfR,CLm > < Tgfa,Out >,
        < Tgfa :[EgfR - act],CLm >,
        none)
```

The transition label is the rule label, suffixed with `#1`, `#2`, ... if there are multiple instantiations. Since in rule 1 there are no unchanged occurrences, `iOccs` is simply the instantiated occurrences from the rule lefthand side, `oOccs` is the instantiated occurrences from the rule righthand side, and `bOccs` is `none`, the empty occurrence set.

As another example, the Sos1 recruitment rule (`13.Sos1.reloc`) is transformed into the following pnet transition.

```
pnTrans('13.Sos1.reloc,
        < Sos1,CLc >,
        <[Sos1 - reloc],CLi >,
        <[Grb2 - reloc],CLi >)
```

In this case `bOccs` is `<[Grb2 - reloc],CLi >` which is necessary for the rule to fire, but not used up.

The process of converting a rule set into a list of pnet transitions uses Maude's meta-level, where rules are represented as data and one can manipulate terms with variables (which are also just data in the meta-level).

7.3 PL PNet models

Once we have a TKB we can derive models and compute with them, asking for subnets, pathways, knockouts, and making comparisons. A model consists of a pnet transition list (specifying the possible transitions) and a set of occurrences representing the initial (or current) state. It is derived from a dish and a TKB by transforming the dish into a set of occurrences and doing a *forward collection* in the TKB from the occurrence set to derive the set of transitions that could possibly be enabled in a computation starting from the initial state. The idea of the forward collection is to iteratively augment the occurrence set with all occurrences that could be produced by firing enabled transitions (from TKB), without removing the `iOccs` part of the transition, and then add enabled transitions to the accumulating list of transitions.

To give a flavor of the functions underlying PLA, we describe the forward collection process in a little more detail. The reader should feel free to skip to the next paragraph if this seems like too much detail. The collection process operates on a tuple `(tkb,occs,pntl,pending,more?)` where `tkb` is list of possible transistions, `occs`, is the accumulated occurrence set, `pntl` is the accumulated transition list. `pending` is list of `tkb` elements that are not yet enabled in `occs`, and `more?` is a boolean which remembers if any new occs have been added to the accumulated set. Initially the triple is `(TKB,dOccs,nil,nil,false)` where `dOccs` is the dish occurrence set, and `nil` is the empty list. In one pass, each collection step transforms the tuple by removing a transition from `tkb` and adding it to `pntl` if the transition is enabled in `occs`. Otherwise it is added to `pending`. If there are any occurrences in the transition `oOccs` part

that are not in occs, they are added to occs and the done? becomes true. The pass ends when there are no more transitions in tkb. If more? is true then a new pass is started. Otherwise the accumulated pntl is returned. This can be expressed as a function fwdCollect defined by the following equations.

```
eq fwdCollect(tkb,dish)
     = fwdCollect(tkb,dish2occs(dish),nil,nil,false) .
eq fwdCollect(pnTrans(rid,iOccs,oOccs,bOccs) tkb,
              occs, pntl,pending,more?)
    =
    if contains(occs,union(iOccs,oOccs)) .
    then fwdCollect(tkb, union(occs,oOccs),
            (pntl, pnTrans(rid,iOccs,oOccs,bOccs)),pending,
            (if (oOccs - occs == none) then more? else true fi)
    else fwdCollect(tkb, occs, pntl,
            (pending, pnTrans(rid,iOccs,oOccs,bOccs)),more?)
    fi .
eq fwdCollect(nil,occs,pntl,pending,false) = pntl.
eq fwdCollect(nil,occs,pntl,pending,true) =
        fwdCollect(pending,occs,pntl,nil,false) .
```

This forward collection produces a transition list that is possibly an over approximation. That is, any transition of TKB that becomes enabled in some computation from the initial state will be in the accumulated transition list but, there may be some transitions that do not become enabled in any computation from the initial state. The crucial point is that we don't loose any possible computations by restricting the set of transitions to be considered, and the simple over approximation makes the model derivation feasible.

Figure 4 shows a screen shot of the Petri net model of Raf activation, generated by PLA from the dish rafDish whose occurrence set is rafDishOccs

```
eq rafDishOccs =
  < Egf, Out > < EgfR, CLm >  < PIP2, CLm >
  < [Hras - GDP], CLi> < Src, CLi> < [Ube213 - ubiq], CLi >
  < Cbl, CLc >  < Gab1, CLc >  < Grb2, CLc >  < Pi3k, CLc >
  < Plcg, CLc >  < Sos1, CLc >  < 1433x1, CLc >  < Pak1, CLc >
  < Raf1, CLc >  < PP2a, CLc > .
```

As discussed above, ovals are occurrences, with initial occurrences darker. Rectangles are transitions. Dashed arrows indicate an occurrence that is both input and output. The thumbnail sketch in the upper right shows the full network. The main frame shows a magnified version of the portion of the network in the red rectangle. The view in the main frame can be changed by dragging the red rectangle around in the thumbnail frame. It can also be changed using the scroll bars. The Finder in the lower right allows one to locate occurrences and rules by name, and center the view on the selected node.

PLA provids a simple query language for specifying signaling pathways of interest. A query specifies three sets: goals, avoids, and hides. Goals are a set of occurrences that should appear at the end of a pathway, as they represent properties of a desired state. Avoids are a set of occurrences that should not appear in any state in the execution of the pathway. Hides are a set of rules that should not fire in the pathway. To make a

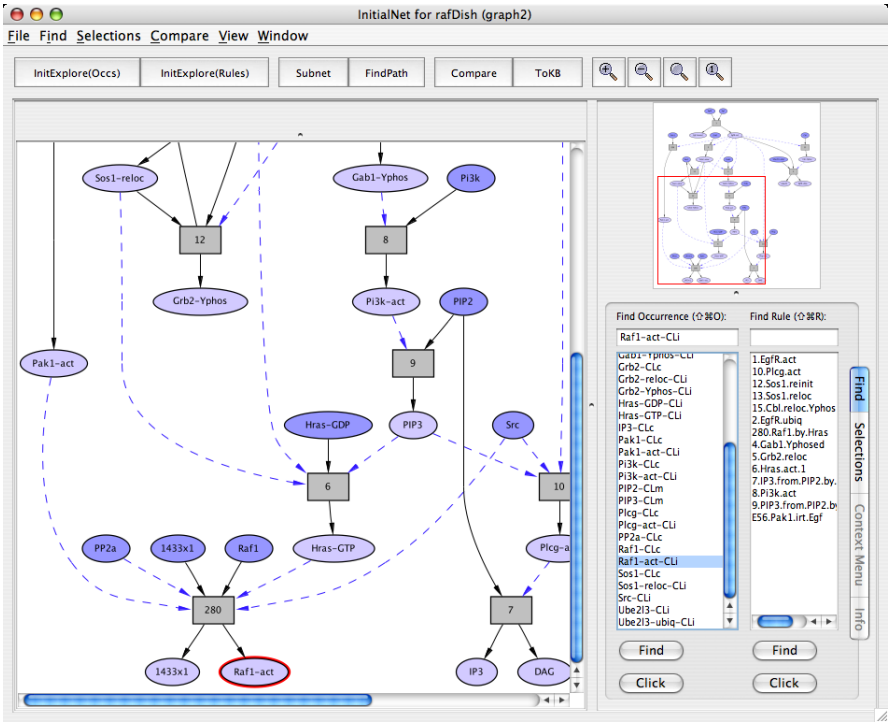


Fig. 4. Raf Activation Model viewed in PLA.

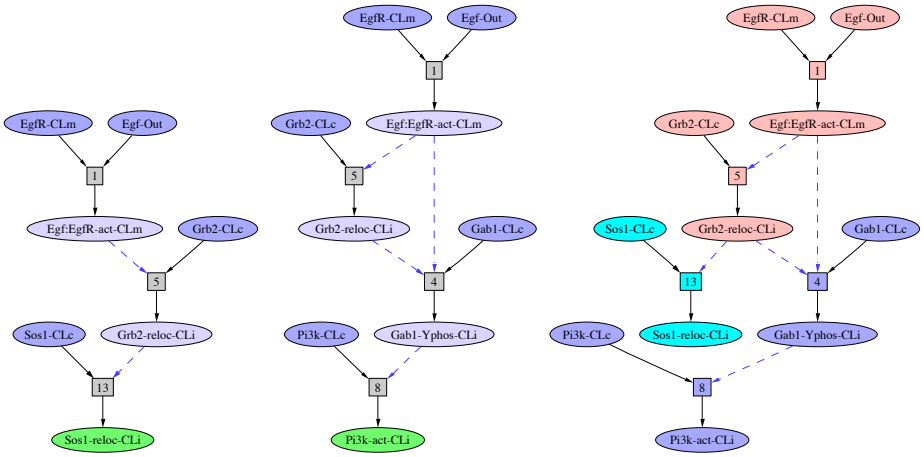
query, goals, avoids, and hides sets can be selected by clicking the occurrence or rule to select, and pressing the corresponding button in the information window that appears. Once query elements have been selected, the user can ask to see the relevant subnet or to find a path. The relevant subnet contains all of rules needed for any (minimal) pathway satisfying the query, while the path is just the first path found by the analysis tool. The relevant subnet is computed directly from the pnet transitions list, together with the initial state and query elements in a manner similar to the forward collection function above.

The logic underlying the query language is a temporal logic Goal queries are answered by model-checking the assertion that the goal set is not reachable, from the initial state `ioccs` in a transition list `pntl*` from which transitions that produce an avoid or are in the hides set are removed.

```
(pntl*,ioccs) |= []~ goal
```

A pathway satisfying a query is obtained by translating the reduced pnet transition list and query into the language of the LoLA model checker [45, 31], asserting that no such pathway exists. If a pathway does exist LoLA returns a list of transitions in the pathway, which PLA converts to a Petri net for display and possibly further analysis. The LoLA

model checker is highly optimized for Petri nets, and thus allows use to compute with very large models.



(a) Sos1 Path (b) Pi3k Path (c) Comparison

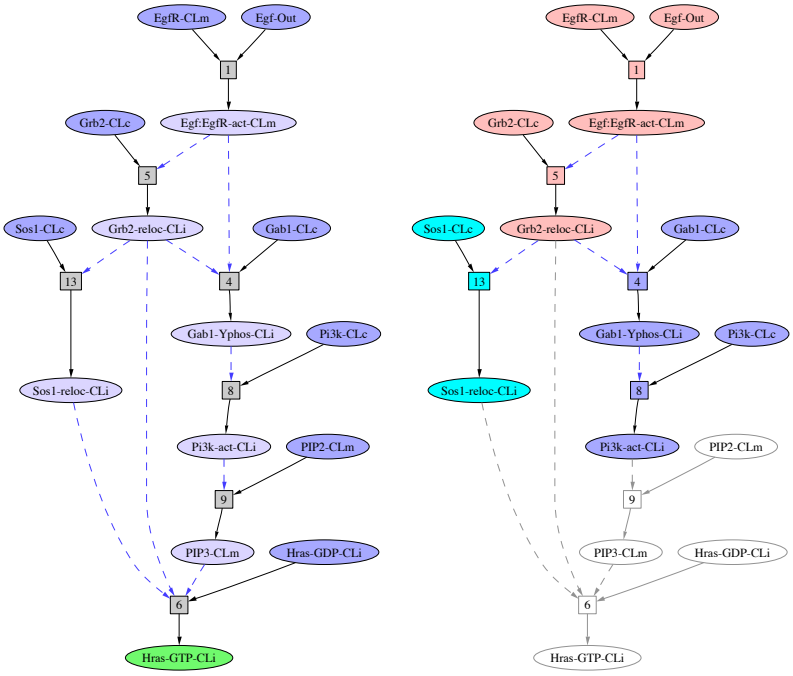
Fig. 5. Sos1 and Pi3k activation paths and comparison

Figure 5 shows pathways in the Raf1 model that recruit `Sos1` (a), and activate `Pi3k` (b), obtained by making `Sos1-reloc-CLi` or `Pi3k-act-CLi` a goal (indicated by coloring the oval green) and using FindPath. The key property of a pathway is that executing the pathway Petri net starting from the initial state leads to a state in which the goal(s) are among the occurrences, that is, a state satisfying the goal is reached. Furthermore, none of the avoids or hides appear in the pathway.

In addition to generating pathway subnetworks, two subnets can be compared. For this, the two networks are merged into one. Figure 5(c) shows the result of comparing the `Sos1` and `Pi3k` pathways. Nodes in both pathways are colored pink, nodes only in the `Sos1` pathway are colored cyan, and nodes only in the `Pi3k` pathway are colored dark lavendar.

The `Sos1` and `Pi3k` pathways are part of the model of Hras activation, and ultimately of Raf activation. Figure 6(a) shows a pathway activating Hras, obtained by specifying `Hras-GTP-CLi` as a goal. Figure 6(b) shows the `Sos1` and `Pi3k` comparison as a subnet of the `Hras` activation pathway (nodes only in the `Hras` path are white).

In principle is it possible to formulate more complex queries, for example expressing that a particular element is a check-point, or that a particular activation state is always eventually reachable. In [1] a study was carried out in which Pathway Logic models were exported to the SAL language [47] and comparison of the effectiveness of several model-checkers in answering temporal logic queries was made. For the large models that we are interested in querying, bounded model checking was able to find counter-examples and thus to generate pathways for goals/avoids queries, but none of



(a) rasPath (b) Sos1 and Pi3k SubPaths

Fig. 6. Ras activation pathway and Sos1,Pi3k subnets

the general model checkers was able to check more complex formulas on large models. The special purpose Petri net analysis seems to scale much better, and the goals/avoids queries are easy for the biologists to understand.

The SmallKB and the Ras and Raf1 activation initial states are available as part of the Pathway Logic Demo available from the Pathway Logic web site `http://pl.csl.sri.com/` along with papers, tutorial material and download of the Pathway Logic Assistant tool.

8 PL Model of Egf Stimulation

As an example of a non-trivial signaling model, we describe a Pathway Logic model that includes all known early responses to Epidermal growth factor (Egf) stimulation. As mentioned in Section 5 Epidermal growth factor receptor (EgfR) signaling regulates growth, survival, proliferation, and differentiation in mammalian cells.
Figure 7 show a cartoon version of molecular components and interactions involved in EfgR signaling. Although the cartoon summarizes a lot of information, the representation is not suited for computational analysis. The PL model of Egf stimulation is

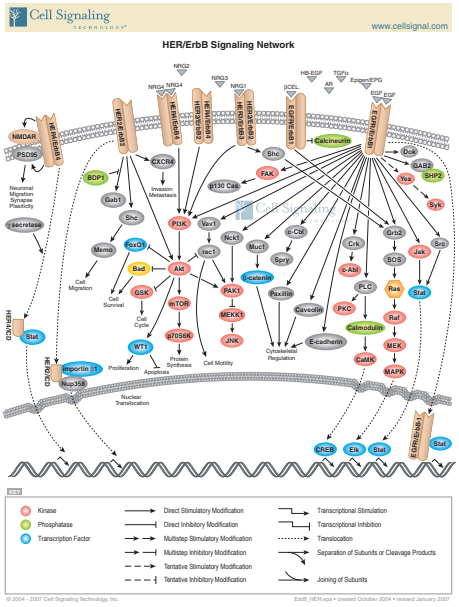
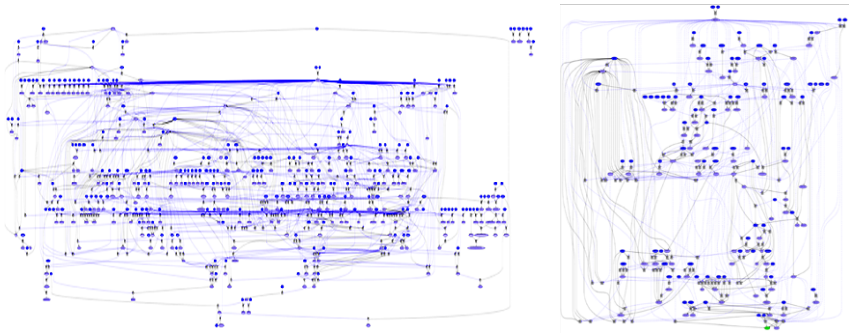


Fig. 7. Cartoon drawing of Egf signaling

based on a PL knowledge base of early response events in adherent cells expressing Egf receptors. Rules in the knowledge base are based on experimental results and data curated from the published scientific literature. The first step in the construction of the knowledge base was to collect the data: 174 papers were searched for appropriate experiments and the results were listed as 1373 evidence items that contain information about state changes. The evidence items are used to determine the components of a reaction rule. The reaction network assembled from data supporting events that might be downstream of EgfR signaling includes over 370 reactions involving more than 460 occurrences (signaling molecules in different states and locations). These rules were combined with a collection of Common Rules curated from additional experimental data from experiments not specific to Egf stimulation.

As explained in Section 6, given a knowledge base, a model is obtained by defining an initial state—the cellular components (proteins, chemicals, or nucleic acids), their modifications, and locations. For the Egf model, the initial state represents a serum starved, adherent cell expressing EgfR and was curated from published experimental data. An impression of the Pathway Logic Assistant (PLA) rendering of the model as a Petri net is shown in Figure 8 (a). Clearly this is a complex model. PLA can be used to browse the model and to ask for subnets or pathways satisfying goals of interest. For example, the subnet of all reactions relevant to activation of Erk in response to a stimulus by Egf is obtained by making Erk1 (and/or Erk2) a goal and asking PLA for the subnet. This is shown in Figure 8 (b).



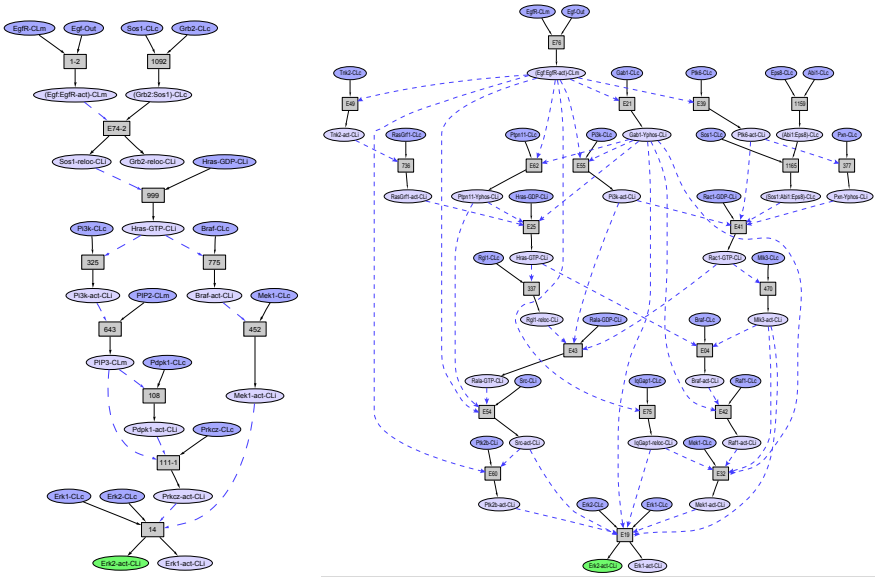
(a) Egf stimulation net (b) Erk activation subnet

Fig. 8. Model of Egf stimulation.

Unfortunately, there are so many potential paths to Erk activation that the Petri net is still too complicated to comprehend without visualization tools or additional simplifying constraints. The result of asking PLA to find a pathway activating Erk is shown in Figure 9 (a). This pathway is similar to the canonical pathway that extends our Raf activation pathway of Section 7. But, the subnet for Erk activation contains many possible paths that activate Erk. Which is the "correct" path? Currently there is no tool that will produce all paths for individual inspection and further analysis.

Our approach is to use additional biological knowledge to further constrain the network. A list of 85 state changes demonstrated experimentally to occur in response to a short stimulus with Egf was collected as part of the curation process. These occurrences (protein states) were set as goals. This ensured that the paths used to reach specific chosen goals are consistent with other observed events. In addition, Egf specific rules were given precedence over Common Rules abstracting these rules. The Egf specific rules that contain requirements specific to Egf signaling that must be satisfied before they can fire. This ensures that any pathways found will include events that must happen before Erk is activated. Figure 10 shows the pathway satisfying all of the additional constraints. The existence of the constrained network containing all 85 events observed in response to Egf stimulation is a form of model validation (or more accurately failure of invalidation). Figure 9 (b) shows the path to Erk activation within the constrained network. This is also a pathway in the full network, but it differs from that found by searching in the unconstrained network, as we have forced the model-checking tool to work in the context of realizing all the other observed events. We see that it is a great deal more complicated than the canonical pathway. This result is not surprising, given the large number of molecular components involved in the collected evidence items.

The constrained path from Egf to Erk contains many unfamiliar events in comparison to the canonical pathway. For example, Rala is required for Src activation in response to Egf, but Sos1 is not required for Hras activation in response to Egf. More generally, models based on a knowledge base such as the curated PL knowledge base demonstrate that the series of events between activation of EgfR by Egf and activation of Erk and other goals may not be as simple as those described in canonical pathways.



(a) Pathway activating Erk (b) Erk activation constrained

Fig. 9. Pathways activating Erk.

9 Conclusion

Pathway Logic is a symbolic systems biology approach to modeling biological processes based on rewriting logic. We have described the use of Pathway Logic to model signal transduction processes, and the use of the Pathway Logic Assistant to browse and analyse these models. Pathway logic can also be used to model and analyze metabolic networks and to interpret experimental data. Future challenges include integration of signaling and metabolic network models, and new abstractions to simplify networks and identify meaningful modules.

Acknowledgements. The author would like to thank the SFM-Bio organizers for inviting this tutorial paper; and the members of the Pathway Logic team for their contributions to the development of modeling techniques and the analysis and visualization tools. Particular thanks to Merrill Knapp, our curator-in-chief, for always being willing to explain the biology, help with illustrations, and give constructive criticisms.

References

1. R. Apolzan. Rapid prototyping applications of formal reasoning tools to biological cellular signalling networks, 2005. ⟨`http://mcs.une.edu.au/~iop/Data/Papers/`⟩.
2. Muffy Calder, Vladislav Vyshemirsky, David Gilbert, and Richard Orton. Analysis of signalling pathways using the PRISM model checker. In G. Plotkin, editor, *Proceedings of the Third International Conference on Computational Methods in System Biology*, 2005.

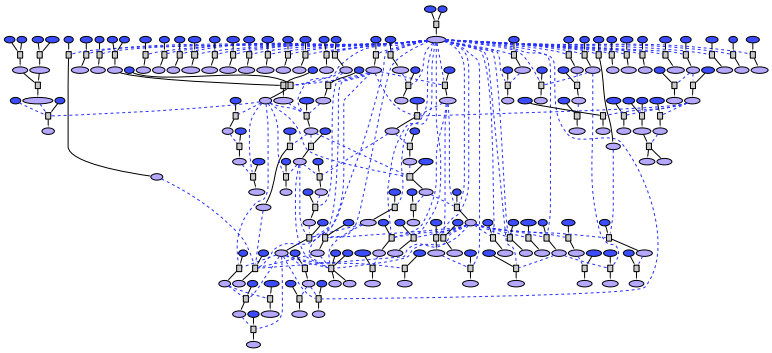


Fig. 10. Constrained model of Egf stimulation.

3. Laurence Calzone, Nathalie Chabrier-Rivier, Francois Fages, Lucie Gentils, and Sylvain Soliman. Machine learning bio-molecular interactions from temporal logic properties. In G. Plotkin, editor, *Proceedings of the Third International Conference on Computational Methods in System Biology*, 2005.
4. Luca Cardelli. Abstract machines of systems biology. In *Transactions on Computational Systems Biology III*, volume 3737 of *LNCS*, pages 145–168. 2005.
5. N. Chabrier-Rivier, M. Chiaverini, V. Danos, F. Fages, and V. Schächter. Modeling and querying biomolecular interaction networks. *Theoretical Computer Science*, 351(1):24–44, 2004.
6. C. Chaouiya. Petri net modelling of biological networks. *Briefings in Bioinformatics*, 8:210–219, 2007.
7. Manuel Clavel, Francisco Durán, Steven Eker, Patrick Lincoln, Narciso Mart´-Oliet, José Meseguer, and Carolyn Talcott. *All About Maude: A High-Performance Logical Framework*. Springer, 2007.
8. W. Damm and D. Harel. Breathing life into message sequence charts. *Formal Methods in System Design*, 19(1), 2001.
9. J. S. Edwards, M. Covert, and B. O. Palsson. Metabolic modelling of microbes: The flux-balance approach. *Environmental Microbiology*, 4(3):133–140, 2002.
10. S. Efroni, D. Harel, and I.R. Cohen. Towards rigorous comprehension of biological complexity: Modeling, execution and visualization of thymic t-cell maturation. *Genome Research*, 2003. Special issue on Systems Biology, in press.
11. Steven Eker, Merrill Knapp, Keith Laderoute, Patrick Lincoln, José Meseguer, and Kemal Sonmez. Pathway Logic: Symbolic analysis of biological signaling. In *Proceedings of the Pacific Symposium on Biocomputing*, pages 400–412, January 2002.
12. Steven Eker, Merrill Knapp, Keith Laderoute, Patrick Lincoln, and Carolyn Talcott. Pathway Logic: Executable models of biological networks. In *Fourth International Workshop on Rewriting Logic and Its Applications*, volume 71 of *Electronic Notes in Theoretical Computer Science*. Elsevier, 2002.
13. G. Pearson et al. Mitogen-activated protein (MAP) kinase pathways: regulation and physiological functions. *Endocr. Rev.*, pages 153–183, 2001.
14. F. Fages, S. Soliman, and N. Chabrier-Rivier. Modelling and querying interaction networks in the biochemical abstract machine BIOCHAM. *Journal of Biological Physics and Chemistry*, 4(2):64–73, 2004.

15. Jasmin Fisher and Thomas A. Henzinger. Executable cell biology. *Nature Biotechnology*, 25(11), 2007.
16. H. Genrich, R. Küffner, and K. Voss. Executable Petri net models for the analysis of metabolic pathways. *Software Tools for Technology Transfer*, 3, 2001.
17. R. Ghosh, A. Tiwari, and C. Tomlin. Automated symbolic reachability analysis with application to delta-notch signaling automata. In O. Maler and A. Pnueli, editors, *Hybrid Systems: Computation and Control HSCC*, volume 2623 of *LNCS*, pages 233–248. Springer, April 2003.
18. David Gilbert, Monika Heiner, and Sebastian Lehrack. A unifying framework for modelling and analysing biochemical pathways using petri nets. In Muffy Calder and Stephen Gilmore, editors, *CMSB*, volume 4695 of *Lecture Notes in Computer Science*, pages 200–216. Springer, 2007.
19. P. J. Goss and J. Peccoud. Quantitative modeling of stochastic systems in molecular biology using stochastic Petri nets. *Proceedings of the National Academy of Science*, 95:6750–6755, 1998.
20. D. Harel. Statecharts: A visual formalism for complex systems. *Science of Computer Programming*, 8:231–274, 1987.
21. T.A. Henzinger. The theory of hybrid automata. In *11th IEEE Symposium on Logic in Computer Science*, pages 278–292, 1996.
22. R Hofestädt. A Petri net application to model metabolic processes. *Systems Analysis Modelling Simulation*, 16:113–122, 1994.
23. N. Kam, I.R. Cohen, and D. Harel. The immune system as a reactive system: Modeling t cell activation with statecharts. In *Visual Languages and Formal Methods (VLFM'01)*, pages 15–22, 2001.
24. N. Kam, D. Harel, H. Kugler, R. Marelly, A. Pnueli, J. Hubbard, and M. Stern. Formal modeling of C.elegans development: A scenario-based approach. In *First International Workshop on Computational Methods in Systems Biology*, volume 2602 of *Lecture Notes in Computer Science*, pages 4–20. Springer, 2003.
25. W. Kolch. Meaningful relationships: The regulation of the Ras/Raf/MEK/ERK pathway by protein interactions. *Biochem J.*, 351:289–305, 2000.
26. R. Küffner, R. Zimmer, and T. Lengauer. Pathway analysis in metabolic databases via differential metabolic display (DMD). *Bioinformatics*, 16:825–836, 2000.
27. J. M. Kyriakis and J. Avruch. Mammalian mitogen-activated protein kinase signal transduction pathways activated by stress and inflammation. *Physiol. Rev.*, 81:807–869, 2001.
28. L.Cardelli. Brane calculi interactions of biological membranes. In *Computational Methods in Systems Biology*, volume 3082 of *LNCS*. Springer, 2004.
29. C. Li, Q. W. Ge, M. Nakata, H. Matsuno, and S. Miyano. Modelling and simulation of signal transductions in an apoptosis pathway by using timed petri nets. *Journal of Bioscience*, 32:113–127, 2007.
30. P. Lincoln and A. Tiwari. Symbolic systems biology: Hybrid modeling and analysis of biological networks. In R. Alur and G. Pappas, editors, *Hybrid Systems: Computation and Control HSCC*, volume 2993 of *LNCS*, pages 660–672. Springer, March 2004.
31. LoLA: Low Level Petri net Analyzer, 2004. `http://www.informatik.hu-berlin.de/~kschmidt/lola.html`.
32. H. Matsuno, A. Doi, M. Nagasaki, and S. Miyano. Hybrid Petri net representation of gene regulatory network. In *Pacific Symposium on Biocomputing*, volume 5, pages 341–352, 2000.
33. J. Meseguer. Conditional Rewriting Logic as a unified model of concurrency. *Theoretical Computer Science*, 96(1):73–155, 1992.
34. R. Milner. *Communication and Concurrency*. Prentice Hall, 1989.
35. R Milner. *Communicating and Mobile Systems: The pi-Calculus*. Cambridge University Press, Cambridge, UK, 1999.

36. F. Nielson, H. R. Nielson, C. Priami, and D. Rosa. Control flow analysis for bioambients. In *BioConcur*, 2003.
37. Gh. Päun. *Membrane Computing. An Introduction*. Springer-Verlag, 2002.
38. J. L. Peterson. *Petri Nets: Properties, analysis, and applications*. Prentice-Hall, 1981.
39. C. A. Petri. Introduction to general net theory. In Brauer, W., editor, *Net Theory and Applications, Proceedings of the Advanced Course on General Net Theory of Processes and Systems, Hamburg, 1979*, volume 84 of *LNCS*, pages 1–19, Berlin, Heidelberg, New York, 1980. Springer-Verlag.
40. C. Priami, A. Regev, E. Shapiro, and W. Silverman. Application of a stochastic name-passing calculus to representation and simulation of molecular processes. *Information Processing Letters*, 80:25–31, 2001.
41. M.J. Prez-Jimnez and F.J. Romero-Campero. Modelling EGFR signalling cascade using continuous membrane systems. In G. Plotkin, editor, *Proceedings of the Third International Conference on Computational Methods in System Biology*, 2005.
42. V. N. Reddy, M. N. Liebmann, and M. L. Mavrovouniotis. Qualitative analysis of biochemical reaction systems. *Computational Biological Medicine*, 26:9–24, 1996.
43. A. Regev, E. Panina, W. Silverman, L. Cardelli, and E. Shaprio. Bioambients: An abstraction for biological compartments, 2004.
44. A. Regev, W. Silverman, and E. Shapiro. Representation and simulation of biochemical processes using the pi-calculus process algebra. In *Pacific Symposium on Biocomputing*, volume 6, pages 459–470. World Scientific Press, 2001.
45. Karsten Schmidt. LoLA: A Low Level Analyser. In Mogens Nielsen and Dan Simpson, editors, *Application and Theory of Petri Nets, 21st International Conference (ICATPN 2000)*, volume 1825 of *Lecture Notes in Computer Science*, pages 465–474. Springer, 2000.
46. R. Seger and E. G. Krebs. The mapk signaling cascade. *FASEB J.*, 9(9):726–735, 1995.
47. N. Shankar. Symbolic analysis of transition systems. In *Proceedings of the International Workshop on Abstract State Machines, Theory and Applications*, pages 287–302. Springer, 2000.
48. M.-O. Stehr. A rewriting semantics for algebraic nets. In C. Girault and R. Valk, editors, *Petri Nets for System Engineering – A Guide to Modelling, Verification, and Applications*. Springer-Verlag, 2000.
49. C. Talcott, S. Eker, M. Knapp, P. Lincoln, and K. Laderoute. Pathway logic modeling of protein functional domains in signal transduction. In *Proceedings of the Pacific Symposium on Biocomputing*, January 2004.
50. Carolyn Talcott. Formal executable models of cell signaling primitives. In Tiziana Margaria, Anna Philippou, and Bernhard Steffen, editors, *2nd International Symposium On Leveraging Applications of Formal Methods, Verification and Validation ISOLA06*, pages 303–307, 2006.
51. Carolyn Talcott. Symbolic modeling of signal transduction in pathway logic. In L. F. Perrone, F. P. Wieland, J. Liu, B. G. Lawson, D. M. Nicol, and R. M. Fujimoto, editors, *2006 Winter Simulation Conference*, pages 1656–1665, 2006.
52. Carolyn Talcott and David L. Dill. Multiple representations of biological processes. *Transactions on Computational Systems Biology*, 2006.
53. Ashish Tiwari. Abstractions for hybrid systems. *Formal Methods in Systems Design*, 32(1):57–83, 2008.
54. Ashish Tiwari, Carolyn Talcott, Merrill Knapp, Patrick Lincoln, and Keith Laderoute. Analyzing pathways using sat-based approaches. In Hirokazu Anai, Katsuhisa Horimoto, and Temur Kutsia, editors, *Algebraic Biology 2007*, volume 4545 of *LNCS*, pages 155–169, 2007.
55. Ionela Zevedei-Oancea and Stefan Schuster. Topological analysis of metabolic networks based on Petri net theory. *In Silico Biology*, 3(0029), 2003.