The Pathway Logic Formal Modeling System: Diverse views of a formal representation of signal transduction.

Carolyn Talcott
SRI International
Menlo Park, CA 94025

***Abstract*—The core of the Pathway Logic signal transduction model (STM) is a theory in the rewriting logic language Maude. This theory provides a language for representing the signaling state of a cell and its components, and rewrite rules representing possible signaling events. Used as a theory in rewriting logic, statements about signal propagation can be proved. The theory can also be viewed as a database that can be queried, for example, to find the events in which a given protein might participate or to retrieve all signaling events (rules) of a given type. Given a representation of a cell state, an executable model can be derived from the theory. This model can be executed to observe a possible behavior, or model checked to study properties of signal propagation pathways. Finally, the theory can be viewed as term in the meta-theory. Using reflection, the theory can be mapped to terms in other formalisms, to access additional reasoning tools; to annotated grapical representations for visualization; or to an external representation such as JSON, SBML or BIOPAX for sharing with other formal systems.**

The talk will begin with a perspective on formal modeling. We will then discuss identification and representation of elements of a theory of signal transduction motivated by experimental evidence. Finally we show how the views of the resulting theory are used in practice and how the ideas generalize to modeling other cellular processes such as glycosylation and immune system.

1. Introduction

The goal of the Pathway Logic (PL) project is to develop models and tools to help understand how cells work. Cells respond to changes in their environment through biochemical pathways that detect, transduce, and transmit information to effector molecules within different cellular compartments. Signaling pathways involve the modification and/or hierarchical assembly in space and time of proteins and other molecules into complexes that coordinate and regulate the flow of information according to *logical* rules. Signaling pathways are organized in networks having stimulatory (positive) and inhibitory (negative) feedback loops, and cross talk to ensure that signals are propagated and interpreted appropriately in a particular cell or tissue. Signaling networks are robust and adaptive, in part because of combinatorial complex formation (several building blocks for forming the same type of complex), redundant pathways, and feedback loops.

The PL idea is to develop formal models of biomolecular processes that capture biologist intuitions, but unlike the usual cartoons PL models can be executed, so the biologist/modeler can trace possible flows information. The PL system provides tools to

- organize and analyze experimental findings
- carry out gedanken experiments
- discover/assemble novel execution pathways
- gain insights into the inner workings of a cell

A PL model can be thought of as a new kind of review article.

In this invited paper we present a perspective on formal modeling illustrated by the Pathway Logic system. Key points include representation of concepts, semantic anchoring of model components to biological entities and experimental evidence, inference of specific models from a knowledge base and formal specification of experimental setup.

2. A Modeling Perspective

Developing formal representations and models is both an art and a science. In the following we present a perspective with a focus on modeling naturally occurring processes such as those that make up biological systems. Most of the ideas are relevant to other modeling tasks.

2.1. Modeling 101

One of the first things to do when starting to build a model is to determine what questions you want the model answer. In the case of biological processes, you might want to determine causal relations in a gene regulation network, or the kinetics of a phosphorylation process, or the events leading from detection of a ligand to turning on a gene.

Another key issue is what data is available: what can be observed/measured and how does it relate to the questions of interest. Is the data sufficient to answer the questions?

How can this be explained to a computer? Using a formal representation system equipped with tools for reasoning will not only help in the model building process, but also in using the model to answer questions (the reasoning part).

In summary, the formal modeling of a process consists of defining concepts and their formal representations; gathering data and curating facts; using formal reasoning to study properties of the model, discovering gaps or inconsistencies; repeating to improve the model.

2.2. Executable Models and Symbolic Analysis

Executable models are particularly natural for modeling processes. A good overview can be found in [1]. An executable model represents system states and the possible behaviors—how states change over time—typically by some form of state transition rule. A model is symbolic if families of states can be represented, for example using place holders for some state components. Associated with each such executable formal model is a state transition graph whose nodes are states and edges are transitions connecting states. Formally, the state transition graph is the basis of many analyses.

There are many formalisms supporting executable formal modeling including automata based formalisms, Petri net based formalisms, and rewriting logic (see section 3).

Symbolic analysis – answering questions. What can we do with an executable specification? Once a model is defined, we can define specific system configurations and explore their behavior using a variety of symbolic analyses.

Forward simulation allows us to watch the system run by rewriting according to different builtin or user defined strategies to choose next steps. In this way, specific executions and their event traces can be examined.

Symbolic simulation generalizes basic forward simulation allowing exploring behavior of a possibly infinite family of states. This is only interesting is the case that transitions are specified by patterns that can be matched to a concrete state (by finding matching substitutions) or unified with a symbolic state (by finding substitutions, called unifiers, that equate the transition instance with the instance of the symbolic state). Symbolic simulation starts with a state pattern and transition patterns are applied using unification. As execution proceeds, unification likely leads to more concrete states, and it may turn into a search when there are multiple unifiers. This is called narrowing in logic languages.

Forward search from a given state is a breadth first search of the state transition graph, up to some depth. It is a reachability analysis that (in the case of finite executions) can be used to find **all** possible outcomes or to find only outcomes satisfying a given property. It can also be used to look for undesirable reachable state, for example states violating a required invariant. The kinds of properties analyzable by forward search can be expanded by instrument a model. This involves adding metadata to configurations to collect quantitative or history information, such as time elapsed between events, minimum or maximum values of some parameter, and so on.

Backward search from a given state S runs the model backwards from S, traversing transition edges in reverse direction, to find possible initial states leading to S. This technique can be used in protocol analysis to decide if a proposed attack/bad state is reachable from an honest/legal initial state.

Symbolic backward search like symbolic simulation starts with a state pattern and uses unification rather than matching to run the transitions in reverse. It has the potential to reduce an infinite search space to a finite search space.

Forward collection is a form of abstraction, where abstract states contain the information of the collected reachable states. In the simplest case states are multisets and accumulating information is multiset union. Starting from a given state all newly applicable rule instances are added to the collected rule set and conclusions of the newly applicable rule instances are added to the collected state. This is repeated until there are no newly applicable rule instances. The collected rule instances are those that are potentially applicable in some execution starting from the original state, dropping unreachable rule instances. The collected state gives an upper bound on reachability.

Backward collection works for states that are multisets. Starting by requiring some elements to be present in the multiset, add all rule instances whose conclusions include at least one of the required elements, and add the rule instance premiss elements to the set of required elements. This is repeated until there are no newly applicable rule instances are found. The resulting set of rules forms a submodel that, under suitable conditions contains all the executions leading to the original set of required elements. This is used in Pathway Logic to infer models of subprocesses [2].

Model checking refers to algorithms to determine if all executions of a model from a given initial state satisfy a given property. Forward search is a form of model-checking for simple reachability properties. More complex forms treat properties in temporal logics. Such languages can express properties about the order in which states or actions occur, or conditions under which changes can occur. Ideally, if a property does not hold, a counter example is returned witnessing the failure. An example class of property is: molecule X is never produced before molecule Y. A counter example would be: a execution pathway (set of rule instances) in which Y is produced after X.

Meta analysis is reasoning about the model itself, for example finding transitions that transform a given state element, finding transitions representing a specific modification such as phosphorylation, or checking that transitions satisfy some property such as stoichiometry. Additionally, meta analysis can be used to transform a model and property to another logic (for access to reasoning tools for that logic), or to transform a model to an external syntax for sharing information.

3. Rewriting Logic and Maude

Rewriting logic [3], [4] 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 data types 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 rewrite rule has the form $t \Rightarrow t'$ *if* c where t and t' are terms possibly containing 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 variables (using a matching substitution), 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).

As a simple example, consider a theory with two constants a and b, of sort $Atom$, an operator $p : Atom, Atom \rightarrow Atom$ and a rule $p(x, x) \Rightarrow x$ where s is a variable of sort $Atom$. Then $p(p(a, a), b)$ can be rewritten to $p(a, b)$ by matching the subterm $p(a, a)$ to the rule premiss using a substitution mapping x to a.

Maude is a language and tool based on rewriting logic [5], [6]. 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. Maude also supports reflection with a simple representation of modules and their components, and access to key functions of the core Maude system allowing the user to easily specify execution and search strategies and module transformation.

Other Formalisms for Executable Modeling. In addition to rewriting logic, there are a number of formalism systems available for developing executable models of cellular processes: Petri nets, BioCham, Kappa, BionetGen, and BioPepa, to mention just a few. Petri nets [7] are a general model of concurrent processes with a number of variants offering different expressiveness and analysis complexity. Overviews of different Petri net formalisms and their application to modeling biological processes can be found in [8], [9].

BioCham [10], [11] is a rule-based language for modeling biochemical systems together with simulators for qualitative and quantitative semantics, and a temporal logic language for specifying system properties.

Kappa [12], [13] is a rule-based language for modeling protein interaction networks. Kappa models represent details of protein state and binding interactions. The semantics is given by stochastic graph rewriting. BionetGen [14] is rule based modeling system whose language is similar to that of Kappa, while execution semantics is given by translation to differential equations, or using Gillespie style stochastic simulation.

Bio-PEPA [15] is a process algebra language for the modelling and the analysis of biochemical networks. Bio-PEPA supports different kinds of analysis, including stochastic simulation, analysis based on ordinary differential equations (ODEs) and model checking in PRISM [16].

The Cell Net Optimizer [17] is a software system for assembling Boolean logic models from signalling interaction networks and calibrating the models against experimental data.

These systems all work with executable models based on rules describing local changes in state of proteins and other cellular components. They all provide tools for analysis based on execution/simulation. BioCham and Bio-PEPA also provide support for model-checking based analysis. Pathway Logic shares with Kappa and BioNetGen a rich and controlled language for describing protein states. Distinguishing features of Pathway Logic include the focus on inference of rules from experimental data, a formal representation of experimental findings, mechanisms to link model elements to external sources and rules to supporting evidence, and symbolic representation of states and rules.

4. Pathway Logic representation system

Recall that the big question is: How do cells work? We think of cells as actors in an organ or organism level distributed system. As such, they function asynchronously, make decisions based on local information and communicate by transmitting/receiving signals (biochemicals) or by direct contact. Internally, cells themselves are tiny distributed systems, the actors are proteins, chemicals, genes acting concurrently, and interacting to exchange and propagate information. Here we discuss *PL STM (Signal Transduction Model)* which focuses on intra cellular processes starting with receiving signals from the environment. As we will see, the basic ideas work for multi-cell systems such as the immune system (see 4.5). Thus the questions we would like a model to answer concern the pathways of signal propagation: What are the possible events? What are essential events? What alternative routes are there?

A big challenge is naming model elements. This is important to communicate the information represented in a model. The trouble is that different biologists use different names for the same entity (protein, gene, metabolite ...). A few examples are: Egf vs EGF for the Epidermal Growth Factor; Erk (extracellular signal-regulated kinases) vs MAPK (mitogen-activated protein kinases) for a member of a family of kinases; EgfR vs ErbB1 vs HerbB1 for Epidermal Growth Factor Receptor. The PL solution is to choose names we like, and to link each unique entity name to an entry in a generally accepted public database such as UniProt [18] for proteins, HGNC [19] for genes, HMDB [20] for metabolites, and PubChem [21] for drugs. Such links give unambiguous meaning to the names by linking them to sequence, structure and other properties.

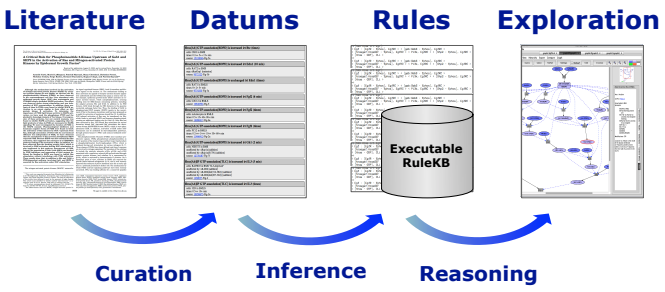

Figure 1. From data to models in PL.

Such names identify reference entities. In a signal transduction model we need to represent instances / occurrences of entities. For proteins, these are characterized by attributes such as post translational modifications (PTMs) and location. PTMs include binding to small molecules such as phosphorylation or acetylation, and cleavage. A protein may need to be is a specific state (active) to carry out its function, and activity may in turn rely on presence or absence of specific PTMs or binding partners. Location can be a compartment or organelle within a cell. It can also specify where in the compartment or be external to the cell. Locations can also be ephemeral, for example the evolving complex formed by a group of proteins attracted to an activated receptor to interact and propagate the signal.

A signal transduction pathway is a (partially ordered) collection of signaling event, each representing a local process. Examples include: ligand binds receptor; activation of an enzyme to initiate another process; translocation—between compartments, or attraction to an ephemeral location; addition or removal of a modifier (phosphate, acetyl group ...), exchange of GDP for GTP; or binding to scaffold. These events may be spontaneous, but more likely require help, for example from an enzyme, or from scaffold proteins that are bringing the reactants together.

4.1. PL from 1k feet

As shown in Figure 1, Pathway Logic models are founded on two formal knowledge bases: a curated datum knowledge base (DKB), and a rules knowledge base (RKB), that share a controlled vocabulary formalized in Maude. A datum formalizes an experimental observation of the state or location of protein or other biomolecule (RNA, Lipids, ...) either in some well-defined experimental condition, or a change in response to some signal or perturbation [22].

Signaling events are formalized as rewrite rules. They are generally inferred from datums, although rule sets can also be curated from review articles and text books, or simply hypothesized. A rule contains terms representing the change (before and after state) as well as terms representing the biological context in which the change takes place. A rule may be parametric, containing variables that can be instantiated in multiple ways to give different rule instances usable in different contexts.

The RKB can be thought of as a global model. Executable models of specific situations are generated by specifying initial conditions and constraints, formalized using a notion of *dish* (as in Petri dish). A dish can be thought of as representing an experimental setup: cell type, growth conditions, and treatments or other perturbations. The cell type and growth conditions are represented by specifying which proteins and other biomolecules are present, their location, and their modification and/or activity state. Given a dish and an RKB, the symbolic reasoning and abstraction techniques of Section 2 can be used to infer a minimal set of rule instances that cover all situations reachable from the initial state. The resulting concrete rule set naturally forms a network, linking rules by shared output/input elements.

The PL STM consists of rules concerning response to over 35 different stimuli (including Egf, IL1, Ngf, Tnf, Tgfb ...) as well as *common rules* that formalize local changes independent of a particular stimulus. The Dishes (one for each stimulus), Controlled vocabulary, and Rules can be exported as JSON for sharing with other tools or importing into a Mongo DB for traditional database type searches. A particular model (dishnet) can be exported in SBML format or APPN (for Petri Net tools). A model together with a reachability property can be exported to the language of LoLA [23] for efficient model checking. The PL system uses this to find execution pathways as counter-examples to claims that a given state can not be reached.

The PL DKB currently contains over 42k entries, and can be searched via a biologist friendly web form [24]. The published RKB has 1575 rule instances (generated from dishes for each of the stimuli) connecting 1378 occurrences. The development RKB, which grows daily, is much larger. The DKB, PL models, tools, and tutorials are available from the PL website [25].

4.2. Pathway Logic representation of state

In Pathway Logic, model elements and state are represented using a controlled vocabulary that is specified as a functional module in Maude. There is a core vocabulary shared by all PL knowledge bases/models and a model specific vocabulary that declares specific model element (proteins, chemicals, modifications, locations, ...). The PL controlled vocabulary has several roles: organizing concepts via a sort/type hierarchy; determining legal/well-formed/meaningful terms by specifying constants and typed term constructors, and giving meaning to constants by providing metadata linking constant symbols to external references (Uniprot, HGNC, HMDB, ...).

Entities. The top level sorts of entities are collected in the sort `Thing` which can be either a `SimpleThing` or a `Complex`, formed from simple things by repeated application of a commutative pairing operation. The following specifies the partial order on subsorts of `Thing`.

```
sorts  BProteint Protein Composite Gene
       Chemical Complex .
sorts  SimpleThing Thing  .
```

```
subsort BProtein < Protein .
subsort Protein Gene Composite Chemical
        < SimpleThing .
subsort Complex SimpleThing < Thing .
```

The sort `BProtein` represents gene products, while `Protein` extends this to include modified or cleaved versions. Individual entities are specified by `op` declarations that give the name, the least sort, and associated metadata. An example is the declaration of the protein `Hras` The sort hierarchy is refined to include a sort `RasS` for the Ras family, and a sort `HrasSort` for the specific protein `Hras`

```
sort HrasSort .
subsort HrasSort < RasS < BProtein .

op Hras : -> HrasSort [ctor metadata (
  (spnumber P01112)  (hugosym HRAS)
  (synonyms "GTPase HRas"
    "Transforming protein p21"
    "Harvey murine sarcoma virus oncogene"
    "H-Ras-1"  "c-H-ras"
    "HRAS1" "RASH1"
    "RASH_HUMAN"))] .
```

In Maude, metadata is a string. In PL we constraint the string to represent a list of key value pairs. Each protein is required to have an "spnumber" key with associated value the UniProt accession for the named protein. "hugosym" is the HGNC name for the gene coding for this protein. Although not required, there is generally a "synonyms" entry. This is useful to help model users understand the model using familiar terms. It can also help when reading a new paper, to disambiguate a name used in the text.

Protein families are groups of proteins that are indistinguishable for some purposes, usually because antibodies used to identify a protein bind to all members of the family. In PL we can give a name to a generic family member, and link this name with the names of actual family members, as in the following declaration of the `RasS` family representative `Rass`.

```
op Rass : -> RasS [ctor metadata (
 (category Family) (members Hras Kras Nras))] .
```

A group of proteins might be considered a family because the members are believed to have similar function and thus experimental results for one member might be assumed to hold for other members. In PL we introduce sorts for such group and use variables to represent members of the sort. Thus the variable `ras:RasS` can also be used to represent elements of the Ras family (RasS sort). Using a generic name in a rule entails a higher level of abstraction, forcing family members to be uniformly treated the same. Using a variable in a rule allows the modeler to say that the rule applies to any family member, but allows other rules in the same model to make distinctions by naming specific proteins.

The sort `Composite` represents a class of named complexes. Historically these complexes were identified by enzymatic function, and only later it was discovered the entities were not single proteins, but complexes with several proteins that function together. A well-known example is `Pi3k`:

```
op Pi3k : -> Composite [ctor metadata "(
  (subunits Pik3cs Pik3rs)
  (comment  "PI3 Kinase is a heterodimer of:"
    "a p110 catalytic subunit:"
        "Pik3ca, Pik3cb, Pik3cd or Pik3cg"
    "a p85 regulatory subunit:"
        "Pik3r1, Pik3r2, or Pik3r3"))] .
```

As the `Pi3k` example illustrates, composites have a defined structure of subunits playing different roles, but each subunit rule may be filled by any member of a family, not just a single protein. Thus there are 12 possible concrete forms of Pi3K!

Occurrences. A PL model state is multi-set of occurrences of entities (proteins, chemicals, genes, ...). An occurrence specifies an entity, its modifications and/or activity state, and its location. Modifications are specified using the elements of the sort `Modification`, collected into multisets (sort `ModSet`), and the syntax `[entity - modset]` according to the following declarations.[1]

```
sorts Site Modification ModSet .
subsort Modification < ModSet .
op __ : AminoAcid Nat -> Site .
op [_-_] : Protein ModSet -> Protein
                      [right id: none] .
op [_-_] : Gene ModSet -> Gene
                      [right id: none] .
```

Modification by phosphorylation is represented at different levels of abstraction

```
op phos :       -> Modification .
      *** phosphorylated
op phos : Site  -> Modification .
      *** phosphorylated on a specific site
op Yphos : -> Modification .
      *** phosphorylated on Tyrosine
```

The reasons for having multiple levels of detail are (a) to be able to express the level of detail provided by a given experimental design; and (b) to find a common level of detail in a collection of rules so they can function together. An experiment may use an antibody for a specific phosphorylation site of Gab1, for example tyrosine 627 (`[Gab1 - Y 627]`), but we may want to abstract this to simply phosphorylation on a tyrosine (`[Gab1 - Yphos]`) to fit the general level of detail in a developing model.

Pathway Logic treats binding to `GDP` or `GTP` as a modification rather than complex formation. Although biochemically less accurate, it better reflects the role in signalling.

```
op GDP : -> Modification .
  *** bound to GDP (Guanosine diphosphate)
op GTP : -> Modification .
  *** bound to GTP (Guanosine triphosphate)
```

The binding of Hras to GTP is represented as `[Hras - GTP]` rather than as a complex `Hras : GTP`.

1. The underscores are argument placeholders in Maude's mixfix syntax.

PL uses two kinds of location: the usual cellular locations (sort `LocName`), and ephemeral locations (subsort `CompName`). The latter represent groups of proteins that come together in the process of signal transmission. We name the location according to the entity that initiates the formation. For example `EgfRC` is the Egf receptor complex that forms around an Egf receptor after it binds to Egf and auto phosphorylates.

```
sorts LocName  CompName .
subsort CompName < LocName .

op CLm : -> LocName [ctor metadata
    ((definition "Plasma Membrane"))] .
op CLi : -> LocName [ctor metadata
    ((definition "Stuck to the inside" "
                 of the plasma membrane"))] .
op EgfRC : -> CompName [ctor] .
                          *** EgfR complex
```

It is important to distinguish between an entity as a reference to a class of biomolecules, and an instance of an entity participating in a signaling or other cellular process. We call the latter an *occurrence*. Formally an occurrence consists of a *Thing* (a possibly modified entity) and a *LocName*.

```
sorts Occ Occs .
subsort Occ < Occs .
op <_`,_> : Thing LocName -> Occ .
```

In PL, cellular states can be represented as a multiset of locations or as a multiset of occurrences. In the location form entities are syntactically grouped by location, making it easier for a curator to develop rules. The occurrence form (obtained by distributing the location name across the things in the location), has a direct mapping to Petri nets, which we use for analysis, and simplifies computational processing. We will use the only latter here to avoid confusion.

4.3. Pathway Logic representation of rules.

PL rules describe local change and specify the required context. The rule label gives a hint as to what happens. In addition rules must be annotated with evidence which could be a datum page, literature citations, pubmed ids, or simply curator notes. In the STM model we require rules to be linked to supporting evidence in the form of datums. The following is the STM rule for activating Hras (exchange of GDT for GTP) in response to Egf.

```
rl[529.Hras.irt.Egf]:
< Egf : [EgfR - Yphos],EgfRC >
< [gab:GabS - Yphos],EgfRC >
< [gef:HrasGEF - Yphos],EgfRC >
< Pi3k,EgfRC >
< [Shp2 - Yphos],EgfRC >
< [Hras - GDP],CLi >
=>
< Egf : [EgfR - Yphos],EgfRC >
< [gab:GabS - Yphos],EgfRC >
< [gef:HrasGEF - Yphos],EgfRC >
< Pi3k,EgfRC >
```

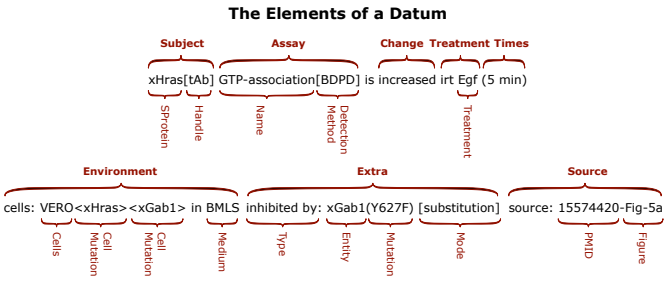

Figure 2. Datum elements.

```
< [Shp2 - Yphos],EgfRC >
< [Hras - GTP],CLi >
***evidence/Egf-Evidence/Hras.irt.Egf.529.txt
```

Symbolic rules represent a family of rule instances using sorted variables. `g:GabS` is a variable standing for `Gab1` or `Gab2`, `gef:HrasGEF` is a variable for any of several HrasGEFs (enzymes to exchange GDP for GTP).

How did we arrive at the Hras rule?. As already mentioned, STM rules are inferred from experimental findings. These are collected using a formal data structure call datums, that capture features of an experiment (design and outcomes) that are needed to interpret the result [22]. Datums are available in text (readable) or json (computable). The datum shown in Figure 2 below says that the amount of GTP bound to Hras is increased after addition of Egf (Epidermal Growth Factor) to VERO cells for 5 minutes. The 'first line of a datum captures the change being observed and the treatment causing the change. Each change type and treatment is mapped to a a rule pattern. The rule pattern for the datum of figure 2 is

```
EgfTC  C  < [G - gmods act ], Lg >
       < [Hras - GDP pmods], CLi >
=>
EgfTC  C < [G - gmods  act ], Lg >
       < [Hras - GTP pmods ], CLi >
```

Here `EgfTC` is the treatment complex formed when Egf binds to the Egf Receptor, `< Egf : [EgfR - Yphos],EgfRC >`. `G` is a variable ranging over Hras GEFs, representing the general knowledge that exchange of `GDP` for `GTP` requires a GEF (Guanine exchange factor). `gmods`, `pmods` are variables indicating that we don't know the exact state of `G` or `Hras`. `C` is a variable standing for requirements to be determined.

The rules of the STM KB are inferred by a biologist using additional information from datums with the same pattern to determine likely values of the variables. For example, from the datum shown in figure 2 a requirement for `Gab1` can be hypothesized, since mutation of the Gab1 protein (`xGab1(Y627F)`) inhibits the reaction. In fact this tells us that the tyrosine site 627 is important for the role that Gab1 plays, and leads the biologist to hypothesize that `[Gab1 - Yphos]` is required.

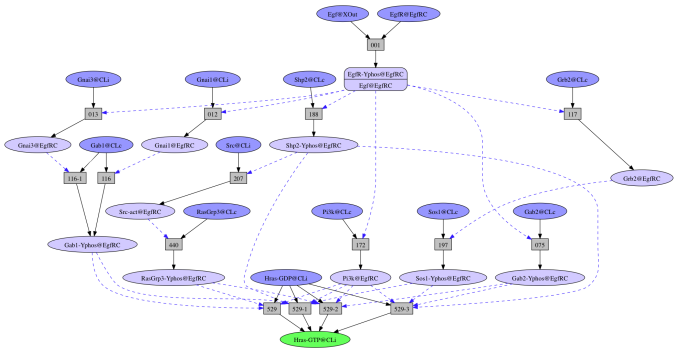

Figure 3. PL model of Hras response to Egf. Ovals represent occurrences (the darker ones are the initial state) and rectangles represent rules. The top rule is the binding of Egf to EgfR, the bottom occurrence is the goal, Hras loaded with GTP. Solid arrows connect occurrences that change to the transforming rule, dashed arrows connect occurrences that are required context to a rule.

In [22] we report progress towards automating this process. The idea is to formalize the rule pattern and the constraints on the pattern variables entailed by datum elements. Given the constraints generated by a set of datums, an answerset programming tool is used to find minimal models, i.e. rules with the variables instantiated to include only provable requirements. This paper also gives more detail on the derivation of the Hras rule.

4.4. Visualization and Analysis

An important part of the PL system is the Pathway Logic Assistant (PLA), which is a tool to visualize, browse, and analyse PL models. Recall that specific models are obtained by specifiying an initial state, called a Dish (as in Petri dish), and using forward collection to collect relevant rule instances from the RKB. The result is refered to as a *dishnet*. Figure 3 shows the graphical representation of the PL model of Hras activation in response to Egf treatment. It was derived in PLA from the Egf dishnet (which is much too large to put in a static figure) by making `< [Hras - GTP], CLi >` a goal and using backward collection to compute the *relevant* subnet. This subnet contains all execution pathways leading to the presence of `< [Hras - GTP], CLi >`. The initial state of the Hras subnet, `HrasDish`, defined as follows.

```
sort Dish .
op PD : Occs -> Dish [ctor] .
op HrasDish : -> Dish .
eq HrasDish =
 PD(< Egf,XOut > < EgfR,EgfRC >
    < Gnai1,CLi > < Gnai3, CLi >
    < [Hras - GDP],CLi > < Src,CLi >
    < Gab1,CLc > < Gab2,CLc > < Grb2,CLc >
    < Pi3k,CLc > < RasGrp3,CLc >
    < Shp2,CLc > < Sos1,CLc >  )
```

One can think of this dish as modeling an experimental setup to study activation of Hras in response to treatment with Egf. The dish includes two instances, `Sos1` and `RasGrp3`, of the `gef:HrasGEF` variable, and two instances, `Gab1` and `Gab2`, of the `gab:GabS` variable. The four rectangles at the bottom of the Hras net are instances of the Hras rule given above. They correspond to use of different combinations of choices of th GEF and GabS variables.

Within a subnet one can ask for all the the execution pathways leading to the goal, using an inference algorithm described in [26]. In the Hras subnet there are six distinct pathways. Knowing all the pathways one can compute properties such as single and double knockouts or essential rules. For example the pair of HrasGEFs, `[RasGrp3,Sos1]`, forms a double knockout. This means that removing both from the initial state makes the goal unreachable, while if only one of the pair is removed, the goal can be reached using the other as the required GEF.

Clicking on an occurence node gives access to the associated metadata with live links to external databases. Clicking on a rule node gives access to the underlying evidence, which for STM models is a page of datums.

More impressive is the subnet for TEY phosphorylation of `Erks` (the representative for the Erk family `Erk1,Erk2`), again too large for a static figure. There are more than 1440 different execution pathways for this goal. This is due to multiple symbolic rules with variables that can be instantiated independently in two or more ways.

It is easy to find pathways for goals such as `< [Erks - phos(TEY)], CLc >`. But, how do we represent the fact that this phosphorylation state is transient? That is we want to find a pathway in which `Erks -> [Erks - phos(TEY)] -> Erks`. Clearly not a simple reachability problem. It can be represented by a temporal logic formula expressing that a state with `Erks` (not phosphorylated) follows a state with `[Erks - phos(TEY)]`, (and that the phosphorylated state is reachable). However, model-checkers for such formulae do not return counter-examples corresponding to pathways, if they manage to return a counter-example at all. This is because in the general case the notion of counter-example is not well-defined. We can solve the problem by keeping a little history as part of the modification of a protein (at the risk of increasing the state space). For example a rule dephosphorylating `[Erks - phos(TEY)]` would produce `[Erks - not(phos(TEY))]`. By enforcing some constraints on when `not` can appear in a modification, we can capture the transient nature of many post translational modifications.

4.5. Beyond STM

In addition to STM, the PL system has also been used to curate models of protease signaling in bacteria, metabolic processes in *Mycobacterium tuberculosis* (Mtb), glycosylation pathways, and response of the immune system to a generic pathogen. The protease and metabolic models use basically the same kinds of entity and state as the STM model. The difference is in the expertise needed to understand the relevant experiments to infer signaling or reaction rules.

Glycosylation is the process by which a carbohydrate (glycan) is covalently attached to a target macromolecule, typically a protein or lipid. Glycosylation plays an important

role in protein folding and stability as well as mediating cell-cell communication. The PL glycosylation knowledge base was curated starting with the KEGG [27] collection of maps concerning glycan biosynthesis, and using additional literature resources to refine the rules. Glycans are synthesized by an incremental process of adding component sugar moieties to a growing carbohydrate. Each step is formalized as a PL rule which is linked to the corresponding KEGG page. Rules can then be collected into executable models by defining an initial state, to explore pathways and carry out insilico/what-if experiments. In order to develop this model, new sorts of protein were introduced to represent the carriers of the different types of glycan assembly. Also a new modification constructor

```
op glyc : Gcode -> Modification .
```

was introduced to model the attachment of a glycan as a modification. `Gcode` is the sort of KEGG codes for glycans and each Gcode is linked to the corresponding KEGG page. For visualization each Gcode is mapped to an icon representing the construction of the glycan from basic sugar entities according to a standard representation system [28].

The immune system model was curated from the Janeway Immunology text book [29] as an exercise in trying to understand how the immune system works. In contrast to signaling and metabolism where the occurrence of proteins and chemicals are the elements that make up the system state and are the subject of rules for change, in the immune system model cells are the main players. Thus we introduce a sort `Cell` as a subsort of `SimpleThing` and subsorts for different types of immune system cell such as Dendritic cells (D-cells, sort `DC`). In analogy to introducing constants to name a generic member of a protein family (for example `Ras`, or `Erk`) we introduce constants to name generic cells of different sorts. For example

```
 op Mac : -> Cell [ctor] .
*** Tissue Macrophage
op DC1 : -> DC [ctor] .
*** a DC that secretes IL12 and Ifng
***                    when bound to TcR
op DC2 : -> DC [ctor] .
*** a DC that secretes IL4
***                    when bound to TcR
```

Activity modifications are introduced to capture the informal concepts used by immunologists. For example a `mature` Dendritic cell is one that has recognized a pathogen and is capable of presenting an associated antigen to a T-cell. An `active` macrophage is capable of killing intracellular pathogens. Other relevant aspects of an immune system cell include proteins being expressed on the cell surface, and proteins (cytokines and chemokines) being secreted. These are formalized by injecting proteins into the appropriate modification subsort. Thus proteins become modifiers rather than modifiees.

Locations are now locations within the organism. Some examples are

```
op BLD : -> LocName [ctor metadata
      ((definition "Blood stream"))] .
op PTS : -> LocName [ctor metadata
      ((definition "Peripheral tissue"))] .
op LN : -> LocName [ctor metadata
      ((definition "Lymph node"))] .
```

Finally, here is an example rule for macrophage response to meeting a pathogen.

```
rl[014.Mac.exposed.to.Path]:
 < [Mac - macmods resting], PTS >
 < Path, PTS >
  =>
 < [Mac - macmods presenting
          sTnf xMhcI* xMhcII* xB7], PTS >
 < Path, PTS >
----------------------------------------
***(The expression of MhcII and B7 is
    induced by ingestion of Pathogen [p340].
    The secretion of Tnf is induced by
    Lps-bearing pathogens [p83].)
```

Even with the partial text book model of the immune system we can find different execution pathways in response to detection of a pathogen: the pathogen is engulfed by a macrophage and killed; the macrophage fails to activate and dies; or neutrophils are activated and kill the pathogen.

The takeaway is that the basic structure of occurrence being a located modifiable entity can be instantiated to model a wide variety of biological processes. All one needs is data and a biologist that understands the data!

Except for the immune system model, these models are available from the PL website. The immune system model, still in its infancy, is not published on the website, but can be made available upon request.

5. Conclusion

We presented a perspective on the process of building models, and on formal representation using executable symbolic models. We then described the Pathway Logic (PL) modeling approach and its formal representation system based on rewriting logic. Distinguishing features of PL include formal representation of experimental evidence, formal links from symbols representing biomolecules to external reference resources, and links from rules to supporting evidence. The PL approach focuses on curation of symbolic rules representing local signaling events and the conditions under which they occur. Specific executable models are derived from a rule knowledge base by specifying the initial state, for example a representation of an experiment setup.

Ongoing challenges include increased automation of the process of collecting datums and inferring rules, and then finding methods to scale symbolic analysis to larger models and more complex questions.

Acknowledgments

The author would like to thank the workshop organizers for inviting this paper. Also, thanks to the Pathway Logic team and especially Merrill Knapp for all their contributions to the Pathway Logic system. The work has been partially supported by funding from NIH, NSF, and DARPA.

References

[1] J. Fisher and T. A. Henzinger, "Executable cell biology," *Nature Biotechnology*, vol. 25, no. 11, 2007.

[2] C. Talcott and D. L. Dill, "Multiple representations of biological processes," *Transactions on Computational Systems Biology*, 2006.

[3] J. Meseguer, "Conditional Rewriting Logic as a Unified Model of Concurrency," *Theoretical Computer Science*, vol. 96, no. 1, pp. 73–155, 1992.

[4] ——, "Twenty years of rewriting logic," *J. Logic Algebraic Program*, vol. 81, no. 7-8, pp. 721–781, 2012.

[5] M. Clavel, F. Durán, S. Eker, P. Lincoln, N. Martí-Oliet, J. Meseguer, and C. Talcott, *All About Maude: A High-Performance Logical Framework*. Springer, 2007.

[6] Maude-Team, "The Maude System," 2016, accessed: 2016-11-10. [Online]. Available: http://maude.cs.uiuc.edu

[7] C. A. Petri, "Introduction to general net theory." in *Net Theory and Applications, Proceedings of the Advanced Course on General Net Theory of Processes and Systems, Hamburg, 1979*, ser. LNCS, W. Brauer, Ed., vol. 84. Berlin, Heidelberg, New York: Springer-Verlag, 1980, pp. 1–19.

[8] D. Gilbert, M. Heiner, and S. Lehrack, "A Unifying Framework for Modelling and Analysing Biochemical Pathways Using Petri Nets," in *Proc. CMSB 2007*. LNCS/LNBI 4695, Springer, 2007, pp. 200–216.

[9] C. Chaouiya, "Petri net modelling of biological networks," *Briefings in Bioinformatics*, vol. 8, pp. 210–219, 2007.

[10] F. Fages and S. Soliman, "Formal cell biology in BIOCHAM," in *8th Int. School on Formal Methods for the Design of Computer, Communication and Software Systems: Computational Systems Biology SFM08*, ser. LNCS, M. Bernardo, P. Degano, and G. Zavattaro, Eds. Springer, 2008, vol. 5016, pp. 54–80.

[11] B. team, "The biochemical abstract machine BIOCHAM," 2016, accessed: 2016-11-08. [Online]. Available: https://lifeware.inria.fr/biocham/

[12] V. Danos, J. Feret, W. Fontanta, R. Harmer, and J. Krivine, "Rule based modeling of biological signaling," in *Proceedings of CONCUR*, ser. LNCS, L. Caires and V. T. Vasconcelos, Eds., vol. 4703. Springer, 2007, pp. 17–41.

[13] K. Team, "KaSim development homepage," 2016, accessed: 2016-11-10. [Online]. Available: http://dev.executableknowledge.org

[14] J. R. Faeder, M. L. Blinov, and W. S. Hlavacek, "Rule-based modeling of biochemical systems with BioNetGen," *Methods in Molecular Biology*, vol. 500, pp. 113–167, 2009.

[15] F. Ciocchetta and J. Hillston, "Bio-PEPA: a framework for the modelling and analysis of biochemical networks," *Theoretical Computer Science*, vol. 410, no. 33-34, pp. 3065–3084, 2009.

[16] M. Kwiatkowska, G. Norman, and D. Parker, "PRISM 4.0: Verification of probabilistic real-time systems," in *Proc. 23rd International Conference on Computer Aided Verification (CAV'11)*, ser. LNCS, G. Gopalakrishnan and S. Qadeer, Eds., vol. 6806. Springer, 2011, pp. 585–591.

[17] J. Saez-Rodriguez, L. G. Alexopoulos, J. Epperlein, R. Samaga, D. A. Lauffenburger, S. Klamt, and P. K. Sorger, "Discrete logic modelling as a means to link protein signalling networks with functional analysis of mammalian signal transduction," *Molecular systems biology*, vol. 5, no. 1, 2009.

[18] "UniProt consortium," 2016, accessed: 2016-11-14. [Online]. Available: http://www.uniprot.org

[19] "Hugo gene nomenclature committee," 2016, accessed: 2016-11-14. [Online]. Available: http://www.genenames.org

[20] "The human metabolome database," 2016, accessed: 2016-11-14. [Online]. Available: http://www.hmdb.ca

[21] "Pubchem resource for information on biological activites of small molecules," 2016, accessed: 2016-11-16. [Online]. Available: http://pubchem.ncbi.nlm.nih.gov

[22] V. Nigam, R. Donaldson, M. Knapp, T. McCarthy, and C. L. Talcott, "Inferring executable models from formalized experimental evidence," in *13th Computational Methods in Systems Biology*, ser. Lecture Notes in Computer Science, O. F. Roux and J. Bourdon, Eds., vol. 9308. Springer, 2015, pp. 90–103.

[23] K. Schmidt, "LoLA: A Low Level Analyser," in *Application and Theory of Petri Nets, 21st International Conference (ICATPN 2000)*, ser. LNCS, M. Nielsen and D. Simpson, Eds., vol. 1825. Springer-Verlag, 2000, pp. 465–474.

[24] "PLA Datums," 2016, accessed: 2016-11-10. [Online]. Available: http://pl.csl.sri.com/datumkb.html

[25] "Pathway Logic," 2016, accessed: 2016-11-10. [Online]. Available: http://pl.csl.sri.com

[26] R. Donaldson, C. Talcott, M. Knapp, and M. Calder, "Understanding signalling networks as collections of signal transduction pathways," in *Computational Methods in Systems Biology*, 2010.

[27] "KEGG: Kyoto Encyclopedia of Genes and Genomes," 2016, accessed: 2016-11-14. [Online]. Available: http://www.genome.jp/kegg/

[28] "Symbol and text nomenclature for representation of glycan structure nomenclature," 2016, accessed: 2016-11-15. [Online]. Available: http://www.functionalglycomics.org/static/consortium/Nomenclature.shtml

[29] K. Murphy and C. Weaver, *Janeway's Immunobiology, 9th Edition*. Garland Science, 2016.