Inferring Executable Models from Formalized Experimental Evidence

Vivek Nigam[1], Robin Donaldson[2], Merrill Knapp[2], Tim McCarthy[2], and Carolyn Talcott[2]

[1] Federal University of Paraíba, João Pessoa, Brazil
[2] SRI International, USA

Abstract. Executable symbolic models have been successfully used to analyze networks of biological reactions. However, the process of building an executable model from published experimental findings is still carried out manually. The process is very time consuming and requires expert knowledge. As a first step in addressing this problem, this paper introduces an automated method for deriving executable models from formalized experimental findings called *datums*. We identify the relevant data in a collection of datums. We then translate the information contained in datums to logical assertions. Together with a logical theory formalizing the interpretation of datums, these assertions are used to infer a knowledge base of reaction rules. These rules can then be assembled into executable models semi-automatically using the Pathway Logic system. We applied our technique to the experimental evidence relevant to Hras activation in response to Egf available in our datum knowledge base. When compared to the Pathway Logic model (curated manually from the same datums by an expert), our model makes most of the same predictions regarding reachability and knockouts. Missing information is due to missing assertions that require reasoning about the effects of mutations and background knowledge to generate. This is being addressed in ongoing work.

1 Introduction

Executable models of signal transduction provide insights into how cells work, and a means to understand and predict the effects of perturbations and mutations, key for cellular understanding of disease and therapeutics. For example, using an *executable* model one can apply algorithms to determine how one can prevent a given state from being reached or to compute alternative execution paths that reach a given state. Developing such models is extremely difficult. It requires collecting, organizing and interpreting experimental evidence, and assembling rules representing hypothesized biochemical reaction that make up a signaling network. This is very labor intensive and inferring a rule from experiments requires substantial biological knowledge. Several curated models of signaling and metabolic pathways are available [3, 13, 20–22]. However, there is a great need for tools to help automate the curation of executable models.

The problem of automatically constructing executable models from experimental evidence has several aspects including: (1) formal representation of experimental findings, (2) formal representation of rules as elements of executable models, (3) extracting

findings from papers, (4) algorithms for inferring rules from findings and (5) algorithms for assembly of executable models. This paper addresses aspects (1), (2) and (4). The contribution is three fold:

1. We describe a formal representation of experimental evidence called *datums*. Each datum captures relevant information about one or more experiments recording conditions under which a specific state or change in state (modification, activity, location) of a protein or other biochemical happens.
2. We define a language of logical assertions that corresponds to the elements of a datum, and a translation from datum syntax to logical assertions.
3. We define axioms that capture the semantics of datums interpreted as partial information about rules to be used as components of an executable model. The logic is that of Answer-Set Programs [11] and we use an existing engine (DLV [14]) to derive minimal models called *answer-sets*. Each answer-set corresponds to one reaction rule. These models are then parsed into rules of an executable model.

Aspect (3) is being addressed as part of an ongoing DARPA project [7] to advance machine reading and reasoning techniques. We use Pathway Logic (PL) [16] as the formal system for representing and querying executable models of cellular processes. Automated analysis techniques such as forward collection and model-checking are used to assemble executable models and execution pathways by specifying a problem of interest (experimental conditions, targets, ...). The PL algorithms rely crucially on the fact that the rules are curated to work together, for example rules that connect must use the same level of detail concerning location and modifications of participants. In contrast, automatically inferred rules capture all the relevant available experimental information, resulting in a knowledge base that is more precise and extensible. However the model assembly process will require automation of the process of transforming rules to work together, without loosing information unnecessarily. This is the topic of ongoing work.

We applied our algorithms to a collection of datums supporting a model of activation of Hras in response to Egf. The model is part of the PL collection of models manually curated by an expert. Although this first version of the rule generation logic does not accaount from some of the information in datums, the resulting model makes the same predictions as the curated model concerning response to Egf stimulation and effects of knockouts, with a small number expected exceptions.

Plan. Section 2 gives a brief overview of Pathway Logic executable models and an informal introduction to datums. Section 3 gives an informal introduction to the rule inference process using an Hras activation rule as an example. Section 4 presents the answer set programming axioms/rules of the datum logic. Section 4.2 describes the mapping of datums to assertions in the logic. Section 5 presents the Hras case study. Section 6 concludes with related and future work.

2 About Pathway Logic and Datums

2.1 Pathway Logic

Pathway Logic (PL) [16] is a system for modeling and reasoning about cellular processes such as signal transduction, metabolism, and cell-cell communication in the immune system. The PL execution model is based on *rewriting logic* [17, 18]. In PL a cell

state is represented as a 'soup' of occurrences, where each occurrence has three components: a protein or other biomolecule (gene, metabolite, ...), a modifier, and a location. The modifier indicates the state of the protein, including binding of small molecules or phosphates, or ability to act on other proteins (enzyme activity). For example the term `< [Hras - GTP], CLi >` is the occurrence of the protein `Hras` modified by binding to the small molecule `GTP` (Guanosine-5'-triphosphate), attached to the inside of the cell membrane (`CLi`). The names used to form occurrences are semantically grounded using meta-data to provide links to standard databases.

Signal transduction steps are formalized as local rewrite rules operating on the relevant part of the cell state. Each rule describes a change in state of a small number of biomolecules (often just one) and the biological context that enables the change. A PL *Rule Knowledge Base* (RKB) consists of symbolic rules containing variables that range over a finite set of proteins or modifications or locations. *STM* (Signal Transduction Model) is a curated PL RKB that constitutes an executable model of signal transduction in the following sense. Given an initial state, which is a set of occurrences representing an experimental setup, called a (Petri) *dish*, the rules can be applied repeatedly, using the Maude rewrite engine [6], to transform the state. This represents a possible sequence of signaling events in a cell. A set of rule instances that can be applied/fired in some order from an initial state is called an *execution pathway*. Specific model networks can be obtained from an RKB by starting with a dish and using *forward collection* [3] to collect all rule instances that might fire in an execution pathway of this dish. Such models can naturally be viewed as Petri Nets [24].

2.2 Datums: Formal Representation of Experimental Results

The PL *STM* model is an RKB whose rules are inferred from cell culture and test tube experiments. In cell-based experiments, cells are grown under known conditions. The cells may be modified by overexpressing some (possibly mutated) proteins, or knocking out some proteins (preventing expression). The resulting population of cells is treated with a stimulus or stress. Some property of the cells is measured before treatment and at one or more times after treatment to determine change in state, if any. The procedure that measures the property change is called an *assay*. Experiments can also be done in a test tube, and some experiments observe untreated cells.

Every rule in the STM RKB is associated with an evidence file, which contains the collected experimental findings giving evidence supporting the rule. These finding are presented in a formal language called *datums*. A datum describes a collection of experimental findings, all based on the same assay, including a main observation, and effects of perturbation of the experimental system. Technically, the collection consists of separate experiments, but they are intended to be interpreted together, so they are collected in a single datum with *extras*. There are two main types of datum, *state datums* and *change datums*, corresponding to two basic types of biological experiments. *State datums* concern properties of cells in a defined state. *Change datums* summarize the change in the state of something resulting from the addition of a stimulus to cells. Rules are derived from change datums.

[3] Forward collection in this case is application of rules without removing the premisses

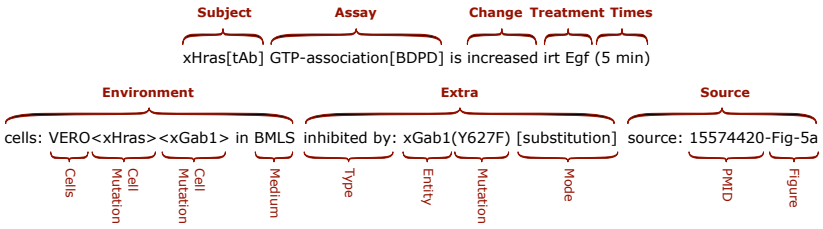


Fig. 1: The elements of a datum.

Datum Structure. The syntax of a datum is designed to be readable by an experimental biologist, but constrained by structure rules and controlled vocabularies so it can be automatically parsed into a formal data structure. The full collection of datums collected for the STM RKB can be accessed via a web query page at light.csl.sri.com/datum. A more detailed description and query examples can be found at pl.csl.sri.com/datumkb.html. The curators notebook (pl.csl.sri.com/CurationNotebook/index.html) contains an intuitive description of datum syntax, catalogs of assays (with their detection methods and other attributes) and cell lines, and a glossary of terms.

The datum in Figure 1 is a change datum that records an experiment in which the binding of GTP to the protein Hras is increased after addition of Egf (Epidermal Growth Factor) to a cell for 5 minutes. The first line contains the *subject* (Hras), the *assay* (GTP-association), the *treatment* (Egf) and the *change* (increased). The parenthetical text (*times*) at the end says the measurement was taken 5 minutes after the treatment. GTP-association is an assay that measures the amount of Hras bound to GTP. The first element of the second line describes the *cellular environment*. In this case VERO cells (a defined cell line) transfected with Gab1 (xGab1), grown in BMLS (Basal Medium Low Serum). The point of transfection is that it results in overexpression. The second element is called an "extra". It records the result of an experiment, that is a perturbation of the original experiment. In this case the cells were transfected with Gab1 with a point mutation (xGab1(Y627F)), in which the tyrosine (Y) at position 627 is replaced by Phenylalanine (F) instead of wild type Gab1 ([substitution]). The third element gives the Pubmed identifier of the paper in which the experiment was reported, and the figure where the experimental results were found (15574420-Fig-5a). Source information is not directly used to infer rules, but is crucial for review and update.

3 Inferring Rules from Datums: an Example

The key ingredients of a datum for rule inference are the subject, the assay, the treatment, the observed change, and the cellular environment. How do we use such experimental information? It is used to constrain the elements of a rule. Specifically, for each assay that measures a change in protein state or location, we associate a rule template that captures the change. The template uses variables for the assay parameters and for additional requirements. The additional requirements can be determined by extras, or by additional experiments. The rule template for a GTP-Association assay is

```
TC C < [G - gmods], Lg > < [P - GDP pmods], Lp > =>
                  TC C < [G - gmods], Lg > < [P - GTP pmods], Lp >
```
(1)

TC represents the treatment complex that forms to initiate the signal propagation, typically a ligand bound to its activated receptor. C stands for unknown requirements. P is the subject of the assay, Lp is a variable representing the cellular location of P, while G stands for some GEF (Guanosine Exchange Factor) that catalyzes the reaction. pmods and gmods represent the modification state of P and G, respectively. Finally Lp and Lg are the locations of P and G, respectively. Lp, Lg, pmod and gmod must be constrained by additional experiments, or background knowledge.

We can use the datum in Figure 1 to partially instantiate the GTP-association rule template as follows.

```
EgfTC C < [G - gmods], Lg > < [Hras - GDP pmods], CLi > =>
       EgfTC C < [G - gmods], Lg > < [HrasP - GDP pmodsd ], CLi >
```
(2)

where EgfTC is the complex that forms when Egf binds to the Egf receptor, which subsequently becomes active and autophosphorylates: < [EgfR - Yphos] : Egf, EgfRC >. We used background knowledge that Hras is anchored to the inside of the plasma membrane to instantiate Lp as CLi.

The next two datums provide evidence that Sos1 is a GEF for Hras.

```
rHras GDP-dissociation[3H-GDP] is increased by xSos1[tAb]IP
cells: none, source: 15039778-Fig-2c

xHras[tAb]IP GTP-association[TLC] is increased itpo xSos1
cells: HEK293 in BMS,   source: 10896938-Fig-1c
```

The first datum says that when you put recombinant Hras (rHras) in a test tube (cells: none) with Sos1 that has been immunoprecipitated (xSos1[tAB]IP) from HEK293 cells, [Hras - GTP] increases. This is direct evidence that Sos1 can act as a GEF in a test tube. We say Sos1 is a *ttGef* (a testtube Gef) for Hras.

Additional evidence that this happens in live cells is needed. The second datum provides such evidence. itpo is a treatment type in which a plasmid for the treatment (Sos1) is introduced into a cell culture and incubated for sufficient time for the treatment protein to become overexpressed. This datum tells us that it is possible that Sos1 can act as a GEF in a cellular environment. We say that Sos1 is an *itpoGef* for Hras. There are datums that report that knocking out Sos1 does not prevent the GDP-GTP exchange. This tells us that there are additional Gefs to be discovered.

Finally, the following datum is evidence for the gabs:GabS requirement.

```
Hras[Ab] GTP-association[BDPD] is increased irt Egf (times)
cells: mEFs in BMLS,  source: 12629518(D)
partially reqs: Gab1 [KO]
```

It says that the reaction partially requires Gab1, determined by removing Gab1 from the cellular environment ([KO]). This suggests that Gab1 has a role, but that their may be other proteins that can play the same role as Gab1 in the activation of Hras in response to Egf. To gain confidence in this hypothesis, and determine candidate similar proteins, more evidence or background knowledge is needed. This will be the topic of future work and extensions of the datum logic.

4 A Logical Specification for Datums

The interpretation of datums is formalized using *Answer Set Programming* (ASP). We start by briefly explaining ASP before proceeding with the logical specifications of datums.

Answer-Set Programs An ASP program is a collection of clauses of three forms:

`(1) D.` `(2) D :- b1,...,bn.` `(3) :- b1,...,bn.`

where `D` is either a ground fact, `a`, or a disjunction, of the form `a1 v a2`, of two ground facts `a1` and `a2`. The symbols `b1`, ..., `bn` are ground facts or negated ground facts written `not a`, where `not` is negation. The symbol `:-` should be interpreted as reversed implication and the symbol `v` as disjunction. Clauses of type `(3)` are called constraints, specifying that `b1`, ..., `bn` should not all be true.

The meaning of an ASP program is a set of ground facts called an *Answer-Set*. An answer-set of a program $\mathcal{P}$ contains a minimal number of facts that makes each clause of the program $\mathcal{P}$ true. For a formal definition see [11, 14].

There are a number of engines that can compute the answer-sets of an ASP program. In the present work we have used the DLV engine [14]. Following the usual convention, variables appearing in programs are considered to be shorthand for the set of all its possible ground instantiations using the constant and function symbols appearing in the program itself.

4.1 Assertions and Inference Rules for Datums

Some of the main predicates used in the logical theory are given below:

- `subject(S,Dt)` denotes that `S` is the subject of the datum `Dt`;
- `assay(Type,Aux,Dt)` denotes that `Type` is the assay type specified by `Dt`, for example, a phosphorylation or GTP-association, and `Aux`, is used for assay parameters such as modification sites `phos!Y627`, or hooks in a binding assay (`none` is used if there are no relevant parameters);
- `treatment(T,Dt)` denotes that `T` is the treatment specified by `Dt`;
- `increased(Dt)`, `irt(Dt)` denote that `Dt` specifies an increase in the changed state of the subject in response to the treatment.

For example, the assertions for the datum of Figure 1 (Section 2) are given below:

```
datum("hras39").                              subject("Hras","hras39").
assay("GTP-Association",none,"hras39").       irt("hras39").
treatment("Egf","hras39").                    increased("hras39").
```

We also have a collection of assertions that are common knowledge, or are implicit in datums collected from experiments by convention. The common knowledge assertions constitute a library used in the inference of the executable rules. Here we added enough to carry out our experiments. An example is the fact that `EgfR` and its modifications are located at `EgfRC`. This is specified by assertions of the form: `location(EgfR, EgfRC, ck)`, where `ck` stands for common knowledge.

Handling Multiple Datums As described in Section 3, some datums contain the evidence for the changes of the subject of a reaction rule. We call these main datums. Other datums, called Auxiliary Datums, contain evidence about non-subject elements of the reaction, for example, required biomolecules or GEFs. We distinguish these datums using the assertions of the form `useM(Dt)` and `useA(Dt)`, where the former specifies that `Dt` is the main datum and the latter that `Dt` is an auxiliarty datum. We specify using constraints that an answer set should have exactly one main datum. We do not show the rules here.

Inferred Assertions We implemented an ASP program that takes the assertions of a datum and generates answer-sets, each of which corresponds to a PL rule. In particular, the ASP will derive the following facts:

- `occBf(X1,L1)` denote that before the reaction, `X1` is located at `L1`;
- `occAf(X2,L2)` denotes that after the reaction, `X2` is at location `L2`;
- `occ(X,L)` denotes that the reaction requires `X` at location `L` in order to occur. Such an assertion can be used for a treatment complex or a require composite;
- `moveRule` and `reactRule` denote that the rule to be extracted is either a rule specifying that the subject moves from one location to another without changing its modifications or it is a rule specifying that the subject changes its modifications without changing its location. This separation between move and react rules provides a finer grain specification of a model that simplifies the (meta) reasoning.

These assertions are used to construct rules in our executable model of the form depicted in Eq. 1. Before we explain how these facts are derived, we illustrate how answer-sets correspond to rule by example. Consider two answer-sets M_1 and M_2, where M_1 contains the set of facts to the left and M_2 contains the set of facts to the right:

$$\left\{\begin{array}{c}\texttt{moveRule},\\ \texttt{occBf(Hras - mods(Hras),L(Hras))},\\ \texttt{occAf(Hras - mods(Hras),EgfRC)},\\ \texttt{occ(Egf:EgfR-Yphos,EgfRC)}\end{array}\right\}\left\{\begin{array}{c}\texttt{reactRule},\\ \texttt{occBf(Hras - mods(Hras) - GDP,L(Hras))},\\ \texttt{occAf(Hras - mods(Hras) - GTP,L(Hras))},\\ \texttt{occ(Egf:EgfR-Yphos,EgfRC)},\\ \texttt{occ(Sos1 - mods(Sos1),L(Sos1))},\\ \texttt{occ(Gab1 - mods(Gab1),L(Gab1))}\end{array}\right\}$$

Here `mods(X)` and `L(X)` are variables that can be instantiated in our executable model by any modifiers and locations, respectively. The answer-set M_1 specifies the rule below where `Hras - mods(Hras)` moves from a generic location `L(Hras)` to the location `EgfRC` in the presence of `Egf:EgfR-Yphos` at location `EgfRC`:

```
< Hras - mods(Hras), L(Hras) > < Egf:EgfR-Yphos, EgfRC > =>
          < Hras - mods(Hras), EgfRC > < Egf:EgfR-Yphos, EgfRC >
```
(3)

The answer-set M_2 specifies the following rule where the subject `Hras - mods(Hras) - GDP` at a generic location `L(Hras)` is modified to `Hras - mods(Hras) - GTP` in the presence of `Egf:EgfR-Yphos` at location `EgfRC`, `Sos1 - mods(Sos1)` and `Gab1 - mods(Gab1)` at the generic locations `L(Sos1)` and `L(Gab1)`, respectively:

```
< Hras - mods(Hras) - GDP, L(Hras) > < Egf:EgfR-Yphos, EgfRC >
< Sos1 - mods(Sos1), L(Sos1) > < Gab1 - mods(Gab1), L(Gab1) >
  =>
< Hras - mods(Hras) - GTP, L(Hras) > < Egf:EgfR-Yphos, EgfRC >
< Sos1 - mods(Sos1), L(Sos1) > < Gab1 - mods(Gab1), L(Gab1) >
```

Specification of Assertion reasoning. As illustrated above answer-sets specify reaction or move rules. This is specified by the following clauses and constraints:

```
reactRule v moveRule.
:- occBf(X1,L1), occAf(X2,L2), moveRule, X1 <> X2.
:- occBf(X1, L), occAf(X2, L), moveRule.
:- occBf(X1, L1), occAf(X2, L2), reactRule, L1 <> L2.
:- occBf(X, L1), occAf(X, L2), reactRule.
```

The first clause specifies that answer-sets must correspond to either move or react rules. The constraints say that in the specification of move rules, the subject should not be modified and it should move. Similarly for react rules, the location of the subject should not change and the subject should be modified. There are other constraints that are omitted, specifying that move rules only make sense when we know where the subject moves to.

We derive `occ`, `occBf` and `occAf` assertions by deriving the corresponding argument, namely the corresponding possibly modified protein and its location. This is done by using the following auxiliary predicates which will be used to infer the elements in a rule of the form in Eq. 1:

- `in(X)` says that there is a possibly modified protein in the rule context, *e.g.*, a treatment complex. `inBf(X)` and `inAf(X)` specify the state of the subject protein before and after the rule, respectively.
- `loc(X,L)` says that a non-subject element `X` is at location `L`. `locBf(X,L)` and `locAf(X,L)` say that the location of the subject `X` is `L` before and after the reaction.

Using these assertions, we derive `occ`, `occBf` and `occAf` assertions using the clauses below:

```
occBf(X,L(X)) :- inBf(X), reactRule.
occAf(X,L(X)) :- inAf(X), reactRule.
occ(X, L(X))  :-  in(X), not hasLocation(X).
occBf(X - mods(X),L) :- subject(X,Dt),useM(Dt),locBf(X,L),moveRule.
occBf(X - mods(X),L(X)) :- subject(X,Dt),useM(Dt),not hasLocBf(X),moveRule.
occAf(modBy(X - mods(X),L) :- subject(X,Dt),useM(Dt),locAf(X,L),moveRule.
```

Here `hasLocation(X)` is an auxiliary assertion (rule omitted), denoting that it is possible to infer a concrete location for `X`.

Datum assertions are used to derive the more basic assertions `in`, `inBf`, `inAf`, `loc`, `locBf` and `locAf`. For example, a GTP-association datum can be used in the following clauses to derive `inBf` and `inAf` facts:

```
inBf(X - mods(X) - GDP) :- irt(Dt), increased(Dt),
       assay(GTP-association, none, Dt), subject(X, Dt), useM(Dt).
inAf(X, mods(X) - GTP) :- irt(Dt), increased(Dt),
       assay(GTP-association, none, Dt), subject(X, Dt), useM(Dt).
in(X) :- tc(X, Dt), useM(Dt).
```

These clauses specify that if the main datum is a GTP-association, then the subject before the reaction should be modified with GDP and after with GTP. Moreover, the treatment complex should be in the dish, specified by the last clause. Similar clauses exists for the other types of datums, such as phosphorylation datums. In a similar way,

the location assertions `loc`, `locBf` and `locAf` are derived from datum assertions. Some of them might be derived from common knowledge. We do not show these clauses here.

As described in Section 3, other datums provide information about the non-subject elements in a reaction. For example, datums may provide information about GEFs. These are specified by the assertions `ttGEF(Q,S,Dt)` and `itpoGEF(Q,S,Dt)`. Both denote that the datum `Dt` specifies that `Q` could be a GEF for the subject `S`. The former, however, denotes that the experiment was carried out in the test tube, while the latter denotes that the experiment was carried out using cells transfected with Q. We infer these assertions from datum assertions as illustrated below.

```
itpoGEF(Q,X,Dt) :- assay(GTP-association,none,Dt), itpo(Dt),
                   increased(Dt), subject(X, Dt), treatment(Q,Dt), useA(Dt).
ttGEF(Q,X,Dt) :- assay(GTP-association, none, Dt), by(Dt),
                 increased(Dt), subject(X, Dt), treatment(Q,Dt), useA(Dt).
```

4.2 Mapping Datums to Assertions

Each datum is mapped to a set of logical assertions that captures the subject, assay, treatment, treatment type, and change elements of a datum. The mapping algorithm takes as input the JSON representation of datums produced by the datum parser and produces input for the DLV engine as described above.

We ignore datums where the interpretation is complex and often requires specific biological knowledge. We currently ignore any datum with no subject, a mutated subject, a mutated treatment or more than one treatment.

In version 1 of the mapping algorithm, only extras of type "reqs" are captured as their interpretation is relatively straightforward. Extending the mapping algorithm to use "inhibited by" extras is a topic of future work.

Many datums report the same basic experiment, i.e. the same subject, assay and treatment. If these datums also have the same change (result) then the mapping will *merge* them, otherwise the datums are reported to the user as a *conflict* for manual inspection. Conflicts may be particularly troubling because datums span many different cell lines and cell types.

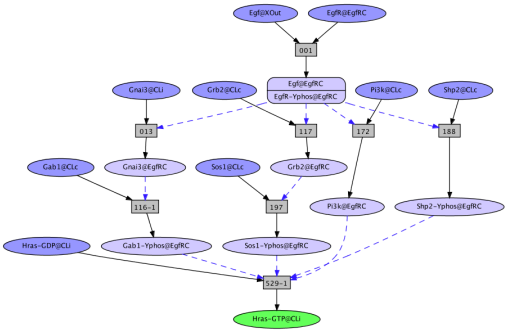
It is then a simple case of mapping each element of the datum (or merged datums) to their logical assertions. For example the datum from Figure 1 and the datum from Section 3 giving the requirment for Gab1 can be merged, omitting elements not used for generating assertions. The result is

```
xHras[tAb] GTP-association[BDPD] is increased irt Egf
inhibited by: xGab1(Y627F) [substitution] partially reqs: Gab1 [KO]
```

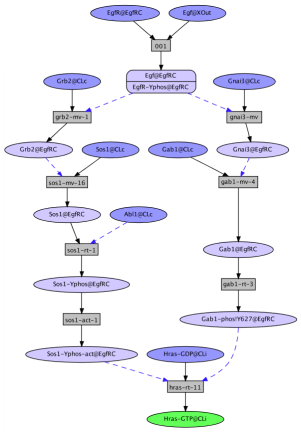
which maps to the following set of assertions:

```
datum("d1-d2").                 subject("Hras", "d1-d2").
irt("d1-d2").                   assay("GTP-association", none, "d1-d2").
increased("d1-d2").             treatment("Egf", "d1-d2").
reqs("Gab1", "d1-d2").
```

In the case of merged datums, the identifiers of the contributing datums are merged, thus `"d1-d2"` above. This allows us to track evidence and eventually reason about the



(a) A pathway in the STM Hras model.



(b) An inferred Hras pathway.

Fig. 2: Hras Models

quality/quantity of evidence used in generating a rule. The actual merged datum in our case study (Section 5) combines 51 datums from the datum knowledge base.

Because we merge all datums for the same change, each set of assertions corresponds to one rule in the model, and contains all information for the set of controls for the rule. Thus the `useOneDatum` directive can be used. Note that auxilliary datums will be used to find assay specific enzymes such as GEFs or Kinases.

5 Signaling Model of Hras Activation by Egf

To test our rule inference tool, we used a model of Hras activation (GTP binding) in response to Egf derived from the PL STM RKB as a 'gold standard'. The Hras model was derived by generating the subnet relevant to the goal `< [Hras - GTP], CLi >` (Figure 2a). The resulting Hras model has 16 rule instances. An execution pathway in this model is shown in Figure 2a. The datums used as input for the inferred model came from the evidence files for these rules together with files containing evidence for Hras GEFs. The JSON datum representation was generated using the datum parser, assertions were generated from the JSON using the assertion mapping tool, and rules were then generated using the logic engine, and automatially converted to Maude syntax.

As discussed in the section 1, the final step is assembly of these rules into a model, a connected set of rules that can be executed to reach expected goals, including the activation of Hras. The basic assembly process is carried out using the PL model generation process. We adapted the inital state for the STM Hras subnet to specify the desired model. The abstraction of details to form a connected rule network was carried out by hand guided by principles developed by the curator of the STM model. Abstracting includes dropping site details from modifications and formalizing knowledge/conjectures such as modification implies activation in specific cases.

The resulting model is more detailed than the STM Hras model (124 rules). This is expected, due to the separation of modification and translocation rules (the STM model typcially collapses these into one step), and the use of location and modification variables that have multiple possible instantiateions.

The inferred model answers most of the queries available supported by PL in the same way that the STM Hras model does. Examples include reachability of given states, existence of multiple execution paths to the Hras goal, and (RasGrp3,Sos1) as a double knockout pair. One of the execution pathwys is shown in Figure 2b.

What about differences? The inferred model has Abl1 as a requirement for Sos1 phosphorylation. There is a single the datum specifying this requirement. The STM curator did not consider one datum showing this requirement as sufficient evidence. Future work includes associating rules with some measure of quantity/quality of evidence. Thus we will be able to assemble models using different criteria for inclusion of rules. In the inferred model Gab1 is a single knockout. The STM Hras model allows Gab2 to substitute for Gab1, thus (Gab1,Gab2) are a double knockout. The rational for Gab2 is based on evidence for behavior similar to Gab1 in the context of other reactions.

Finally, the STM Hras model includes a requrement for `[Shp2 - Yphos]` and a requirement for `Pi3k` in the Hras rule, while the inferred rule does not. These requirements come from extras such as `inhibited by: xPik3r?(mnr)"DN"...` and `inhibited by: xShp2(mnr)"CIA"` that require substantial background knowledge to interpret. For example `xPik3r?(mnr)"DN"` refers to a subunit of `Pi3k` (a complex) that has been mutated to loose its function (DN stands for Dominant Negative). `CIA` stands for 'Constitutively InActive'. The inference is that if the endogenous protein is overwhelmed by a mutated form that is lacking some function then that protein (with that function) is required. Future versions of the assertion generation function will capture more of these inferences.

6 Related Work and Conclusion

Related work. An excellent survey of executable models of biological processes is given in [10] There are a number of network reconstruction algorithms based on statistical reasoning technics such as bayesean inference [12] or belief propagation [19]. They provide a means of elucidating the networks underlying transcriptomics and proteomics data generated from perturbation experiments. These methods postulate causal relations, but do not capture mechanistic details such as necessary conditions.

Methods more closely related to our approach include the following. Net-synthesis [1,2] is a software for synthesis, inference and simplification of signal transduction networks. The main idea is representing observed indirect causal relationships as network paths, introducing pseudo-vertices for unknown intermediaries of these paths and using techniques from combinatorial optimization to find the most parsimonious graph consistent with all experimental observations. The algorithm has been used to synthesize a guard cell signal transduction network for Abscisic Acid Signaling from experimental results [15]. A method based on Petri nets is described in [4]. The reactions of individual proteins are represented as Petri net modules, stored in a database. These modules are similar in spirit to datums. Each place of a module corresponds to a specific functional state of a specific protein domain (e.g. a phosphorylated or unphosphorylated side chain, a catalytically active or inactive domain etc.). For each module, literature references are annotated as part of the modules database entry. Selected modules can be combined to assemble executable Petri net models. The method has been applied to assemble a model of JAK/STAT signalling. In [8,9] an exact algorithm for reconstructing

extended Petri nets from time series data sets is presented. The algorithm finds all alternative minimal networks that are consistent with the data. The method has been applied to synthetic data as suitable times series data was unavailable. In [23] two methods to build signaling models from qualitative data (protein interactions from databases) are proposed, based on analyzing network connectivity and on non-linear optimization. Methods to convert BioPAX models into fully executable models have been proposed, including [5, 25].

Conclusion. We have presented an inference system for deriving signal transduction rules from formally represented experimental findings and applied the system to derive rules for a model Hras activation. [4] Future work includes: extending the mapping of datums to assertions to capture the meaning of experimentaion perturbations using mutations and fragmentation; extracting formal background knowledge from databases; extending the logic to cover more assays and capture more complex reasoning, such as hypothesizing rule requirments and alternatives by similarity; adding logic to generate *common rules* (rules about protein interactions independent of stimulus); and automating assembly of models from generated rules.

References

1. R. Albert, B. DasGupta, R. Dondi, and E. Sontag. Inferring (biological) signal transduction networks via transitive reductions of directed graphs. *Algorithmica*, 51(2):129–159, 2008.
2. R. Albert, B. DasGupta, and E. Sontag. Inference of signal transduction networks from double causal evidence. In D. Fenyo, editor, *Methods in Molecular Biology: Topics in Computational Biology*. Springer Science+Business Media, LLC, 2010.
3. Biocyc pathway/genome database collection, 2015.
4. M. A. Blätke, A. Dittrich, C. Rohr, M. Heiner, F. Schaper, and W. Marwan. Jak/stat signalling: An executable model assembled from molecule-centred modules demonstrating a module-oriented database concept for systems and synthetic biology. *BMC Bioinformatics*, 2012.
5. M. L. Blinov, J. C. Schaff, O. Ruebenacker, X. Wei, D. Vasilescu, F. Gao, F. Morgan, L. Ye, A. Lakshminarayana, I. I. Moraru, and L. M. Loew. Pathway commons at virtual cell: use of pathway data for mathematical modeling. *Bioinformatics*, 30(2):292–294, 2014.
6. 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.
7. DARPA Big Mechanism Project, 2015.
8. M. Durzinsky, W. Marwan, and A. Wagler. Reconstruction of extended petri nets from time-series data by using logical control functions. *J. Mathematical Biology*, 2012.
9. M. Durzinsky, A. Wagler, and W. Marwan. Reconstruction of extended petri nets from time series data and its application to signal transduction and to gene regulatory networks. *BMC Systems Biology*, page 113, 2011.
10. J. Fisher and T. A. Henzinger. Executable cell biology. *Nature Biotechnology*, 25(11), 2007.
11. M. Gelfond and V. Lifschitz. Logic programs with classical negation. In *ICLP*, pages 579–597, 1990.
12. S. M. Hill and et.al. Bayesian inference of signaling network topology in a cancer cell line. *Bioinformatics*, 2012.
13. KEGG: Kyoto encyclopedia of genes and genomes, 2015.

[4] The assertion mapping code and logic are currently being extended and improved. We are happy to make the current working version available upon request.

14. N. Leone, G. Pfeifer, W. Faber, T. Eiter, G. Gottlob, S. Perri, and F. Scarcello. The DLV system for knowledge redlvpresentation and reasoning. *ACM Trans. Comput. Logic*, 7:499–562, 2006.
15. S. Li, S. M. Assmann, and R. Albert. Predicting essential components of signal transduction networks: A dynamic model of guard cell abscisic acid signaling. *PLoS Biol*, 4(10), 2006.
16. P. D. Lincoln and C. Talcott. Symbolic systems biology and pathway logic. In S. Iyengar, editor, *Symbolic Systems Biology*, pages 1–29. Jones and Bartlett, 2010.
17. J. Meseguer. Conditional Rewriting Logic as a unified model of concurrency. *Theoretical Computer Science*, 96(1):73–155, 1992.
18. J. Meseguer. Twenty years of rewriting logic. *Journal of Logic and Algebraic Programming*, 81(7-8):721–781, 2012.
19. E. J. Molinelli and et. al. Perturbation biology: inferring signaling networks in cellular systems. *PLoS Comput Biol*, 9(12), 2013. PMID: 24367245 PMCID: PMC3868523.
20. Pathway logic, 2015.
21. Protein interaction database, 2015.
22. Reactome pathway database, 2015.
23. D. Ruths and R. University. *Deriving Executable Models of Biochemical Network Dynamics from Qualitative Data*. Rice University, 2009.
24. C. Talcott and D. L. Dill. Multiple representations of biological processes. *Transactions on Computational Systems Biology*, 2006.
25. T. Willemsen, K. A. Feenstra, and P. T. Groth. Building executable biological pathway models automatically from biopax. In *Proceedings of the 3rd International Workshop on Linked Science 2013 - Supporting Reproducibility, Scientific Investigations and Experiments (LISC2013) In conjunction with the 12th International Semantic Web Conference 2013 (ISWC 2013), Sydney, Australia, October 21, 2013.*, pages 2–14, 2013.

Appendix: Datums formalized

As discussed in section 2, datums are written in a natural looking language, constrained by a grammar and a set of controlled vocabularies. The datum parser checks well-formedness (and provides helpful feed back when errors are found), and generates formal representations in Maude and JSON. The top level structure of a datum object is given by the following pseudo JSON mapping keys to the value type.

```
datumObj ~
{"subject" : SubjectObj,
 "assay" : AssayObj,
 "change" : ChangeString,
 "treatement_type" : TreatmentTypeString,
 "treatment" : TreatmentObj,
 "environment" : EnvironmentObj,
 "times" : TimesObj,
 "source" : SourceObj,
 "extras" : Array[ExtraObj] }
```

A SubjectObj describes a subject – the biomolecule ("entity"), how it was identified ("handle"), an the orgin (endogenous, expressed, recombinant, or purified). A ProteinObj describes any modifications or mutations of a protein.

```
SubjectObj ~
{ "entity" : ProteinString | GeneString,
  ("protein" : ProteinObj || "gene" : GeneString)
  "handle" : HandleString,
  "origin" : OriginString }
```

An AssayObj describes the assay details: "assay" is a name from the AssayName controlled vocabulary and "detect" is a name from the DetectionMethod controlled vocabulary. The ... represents assay specific additional information, possibly none. For modification assays such as "phos", this includes "sites" which is a list of site names taken from the Sites controlled vocabulary; for binding assays such as "copptby" it includes "hooks", which is an array of SubjectObjs; and for activity assays such as "IVKA" it includes "substrates", an array of names taken from the Substrate controlled vocabulary.

```
AssayObj ~
{ "assay" : AssayName,
  "detect" : DetectionMethod
  ...
}
```

A ChangeString in the case of a treatment is one of "increased", "decreased", "unchanged", "detectable-but unchanged", and in the case of no treatment one of "detectable", "undetectable". A TreatmentTypeString is one of "irt" (in response to something added to the supernatant), "itpo" (in the presence of an over expressed protein),

“itao” (in the absence of a protein), “by” (a test tube experiment), or “none” (no treatment).

In a TreatmentObj “entities” lists the treatment entities (EntityString is a protein or chemical name from a controlled vocabulary); and “treatments” lists details for each treatment–modifcations, mutations, and origin (specified in a controlled vocabulary).

```
TreatmentObj :
  { "entities" : Array[EntityString],
    "treatments" : Array[Treatment] }
```

A TimesObj lists the observation times paired with an indication of the level of change, and specifies the unit of time.

```
TimesObj ~
 { "times" : Array[TLArrl],
   "unit" : {"min","hr"} }
```

The EnvironmentObj gives details of the experimental setup: “cells” defines the cell line or source; “cell-mutations” specifies any changes to the basic cell line, such as overexpressing a possibly mutated proteins, or nullifying a protein; “medium” specify the growing conditions; and “strings” is a place for informal notes.

```
  EnvironmentObj :
     {"cells" : CellString,
      "cell-mutations" : Array[CellMutationString],
      "medium" : MediumString,
      "strings" : String   }
```

Extras describe perturbations of a basic experiment. The ExtraObj scheme reuses structures from the top level in “treatment” and “entities” elements. ExtraTypeString and ExtraModeString are defined by the associated controlled vocabularies. ExtraTypeString includes “reqs” (requires) and “inhibited by”. ExtraModeString includes “addition” and “substitution”.

```
  ExtraObj ~
    { "type" : ExtraTypeString,
      "adjective" : ExtraAdjectiveString,
       "mode" : ExtraModeString,
       "treatment" : Array[TreatmentObj],
       "entities" : Array[EntityString]
  }
```

The controlled vocabularies mentioned are specified as algebraic data types (as Maude modules) and is shared by datums and the PL STM RKB. Datums also have a formal specification as an algebraic data type extending the controlled vocabulary module. Here we presented the corresponding JSON scheme, as this is the form used in the as input to the fuction mapping datums to logical assertions.