An action language approach for representing and reasoning about signaling networks

Chitta Baral, Karen Chancellor and Nam Tran
Dept. of Computer Sc. and Engg.
Arizona State University
Tempe, AZ 85287

Mike Berens and Nhan Tran
Translational Genomics Research Institute
400 N. Fifth Street, Suite 1600
Phoenix, AZ 85004

In this paper we propose to use recent developments in knowledge representation languages and reasoning methodologies for representing and reasoning about signaling networks. Our approach is different from most other qualitative systems biology approaches in that it is based on reasoning (or inferencing) rather than simulation. Some of the advantages of our approach are: we can use recent advances in reasoning with incomplete and partial information to deal with gaps in signal network knowledge; and we can perform various kinds of reasoning such as planning, hypothetical reasoning and explaining observations. Using our approach we have developed the system BioSigNet-RR for representation and reasoning about signaling networks. We use a NFκB related signaling pathway to illustrate the kinds of reasoning and representation that our system can currently do.

1 Introduction and Motivation

Our goal in this paper is to make progress towards developing a system (and the necessary representation language and reasoning algorithms) that can be used to represent signal networks and pathways associated with cells and reason with them. We are interested in qualitative representation and reasoning, which can emulate the reasoning done by a normal biologist (or biochemist) when he/she analyzes and navigates a signaling pathway, but at a much larger scale and with respect to larger networks. (This is in contrast to the approaches where signalling and transformation between various compounds are expressed through differential equations.)

In recent years several qualitative approaches have been proposed in systems biology. Most of these approaches are concerned with modeling and analysis of the model is done via simulation and perturbation. While certain questions about cell behavior can be easily answered using such an approach (such as the impact of a particular event), it is computationally prohibitively expensive to answer questions about explaining a particular observation, or planning to
alter the cell behavior in a particular way using simulation. In both these cases a simulation based approach would entail doing a large number of simulations to find the right explanation or the right plan.

Our approach in this paper is a knowledge based approach. We consider the signal network as a knowledge base which can be asked many different kinds of queries. For the various kinds of queries the knowledge base is augmented with various reasoning mechanisms that allow the answering of the queries. This approach is more general than answering standard database queries with respect a signal network database, as the later is limited to answering queries that can be expressed using a database query language. For example, no existing database query language can express a query whose answer is a plan or an explanation.

An important dimension of our approach is that it allows for reasoning mechanisms that gracefully handle incomplete or partial information. This is extremely important as existing signal networks and pathways often have suspected interaction links, proven interactions whose outputs are not certain (for example the yeast 2-hybrid interactions – as mentioned in ), and simply missing links. Besides being able to handle such incomplete information, our approach also allows for easy updating (referred to as ‘elaboration tolerant’) of the knowledge base when new knowledge becomes available. This avoids significant overhauling of the old model or scrapping of the old model and making a new model from scratch. Note that simple frame based approaches, which are a sub-class of classical logic, are monotonic and hence not elaboration tolerant.

The above mentioned issues of reasoning with incomplete information, elaboration tolerant representation, being able to predict, plan and explain are some of the main topics in the sub-area of ‘Knowledge Representation’ in the area of ‘Artificial Intelligence’. In the next section, we briefly describe the particular recent advances in this field that are important to our approach. In Section 3 we briefly describe the representation and reasoning system we have developed and illustrate its applicability by using it to analyze a NFκB related signaling pathway.

2 Overview of our approach

Our approach is based on two related fields in Artificial Intelligence: (i) Reasoning about actions and change; and (ii) Declarative programming. There has been a lot of recent developments in these fields, and these recent developments play a significant role in the system we have developed.
2.1 Reasoning about actions

Reasoning about actions\textsuperscript{8,9} is concerned with representing and reasoning about dynamic worlds. Research issues in this field involves:

- developing languages that allow for (a) succinct representation of the world, (b) succinct representation of causal relations between objects in the world, (c) succinct representation of the impact of actions (and their executability) on the world, (d) succinct representation of the way the world behaves, or the way we want the world to behave, and (e) representation of action-plans; and

- Reasoning algorithms that can answer a query expressed using (d) and (e) with respect a knowledge base expressed using (a), (b) and (c)

Different kinds of queries above leads to different kinds of reasoning. For example, when the query includes an action-plan (say a simple sequence of actions $a_1, \ldots, a_n$) and a property $p$, asking if $p$ will be true during (or after) the execution of $a_1, \ldots, a_n$ is referred to as plan verification. If $p$ is not given and we ask what happens if $a_1, \ldots, a_n$ is executed, then it is prediction. Now if $p$ is given, but the reasoning system has to find the appropriate $a_1, \ldots, a_n$ so that $p$ is achieved then we have planning. Finally, if a set of observations are given and we need to find a particular action occurrence and/or facts about the intermediate states of the world that explain the observations we have an explanation or a diagnosis. All the above kinds of reasoning may be needed to be done with partial or incomplete information in the knowledge base. The relevance of all these to the task in hand – representing and reasoning about signal networks – is as follows.

In a cell when a ligand binds to an appropriate receptor it activates the receptor. The activated receptor then triggers a series of events. The binding of a ligand to the receptor can be thought of as an action. The conditions under which a ligand can bind to a receptor is similar to the executability conditions of an action in a planning domain. In the context of a cell, by representing the cell behavior using actions, determining the effects and side effects of a drug (that manifests as a ligand) would correspond to prediction. Similarly, figuring out what interventions will change the cell behavior in a particular way corresponds to planning; and determining the causes behind some unexpected observations about the cell corresponds to diagnosis. Finally, since the research on cell decoding is ongoing we only have partial information about the happenings inside a cell. Thus the reasoning schemes need to be able to deal with incomplete information.
2.2 Declarative Programming

In declarative programming the intent of a program (‘what’) is described in a particular declarative programming language and the interpreter figures out the ‘how’ part. For example, a declarative program for sorting a set of numbers just describes what sorting is (and does not say how to sort a set of numbers), and the interpreter when given a set of numbers, figures out the ‘how’ part and does the sorting. Although this approach could be inefficient (in terms of run-time) for certain kind of programs, it has acceptable performance for query answering and reasoning programs. Its strong point is that it is much quicker and easier (and is often more robust) to write a declarative program than a procedural one. For these reasons declarative programming is used for database querying and we will use it for the reasoning associated with our system. (For example, SQL is a declarative language as in SQL we specify what we want, and not how the system should look and search the files to find the answer to our queries.)

Earlier we mentioned that the kind of queries we would like to ask are beyond the ability of database query languages. Moreover, we would like our system to be able to handle incomplete information, and be able to express normative and default statements. Among the various declarative and/or knowledge representation languages (most of which are declarative) that have been proposed one language that has the largest body of support structure (both theoretical results, efficient interpreters and developed applications) is the language of logic programming with the answer set semantics (or simply referred to as AnsProlog). We now give some simple examples showing the ability of AnsProlog (which is different from Prolog has many non-declarative constructs) as it relates to signal networks.

Let $df(m_1, m_2)$ denote the fact that there is direct flow of information from $m_1$ to $m_2$. Given such a collection of facts suppose we would like to make inference regarding whether there is flow of information (possibly through several nodes) from $m$ to $m'$. This notion of flow of information can be expressed in AnsProlog (and can not be succinctly expressed in first-order logic) as follows:

\[
\text{flow}(M, M') \leftarrow df(M, M').
\]
\[
\text{flow}(M, M') \leftarrow df(M, M^*), \text{flow}(M^*, M').
\]

Now one can take the above two rules and a set of facts about $df$ and can ask the above mentioned query (by just asking $\text{flow}(m, m')$) or other queries such as $\text{flow}(m, M)$ which asks to list all nodes to which information flows from the node $m$.

Now suppose we have incomplete information about $df$. For illustration purposes let us consider the case when we have information of the kind $df(m, m')$
and information of the kind \( \neg df(m, m') \). The later kind says that there is no direct flow of information from \( m \) to \( m' \). For a particular pair \( m, m' \), if we have neither \( df(m, m') \) nor \( \neg df(m, m') \), then it means that we do not know whether there is a direct flow of information from \( m \) to \( m' \) or not. Now suppose we would like to ask queries about \( flow \) and \( \neg flow \). Following the transformation\(^1\) we construct the following AnsProlog program which is able to answer the above mentioned queries. (Note that it has been formally proven\(^1\) that with the available information this program answers to the maximum extent possible.)

\[
\begin{align*}
m_df(M, M') &\leftarrow \neg \neg df(M, M'). \\
m_flow(M, M') &\leftarrow m_df(M, M'). \\
m_flow(M, M') &\leftarrow m_df(M, M'), m_flow(M', M'). \\
flow(M, M') &\leftarrow df(M, M'). \\
flow(M, M') &\leftarrow df(M, M'), flow(M', M'). \\
\neg flow(M, M') &\leftarrow \neg m_flow(M, M').
\end{align*}
\]

Note that in the above program we use the operator ‘not’, which is different from the classical negation operator \( \neg \). While ‘not \( f \)’ in the body of a rule intuitively means that \( f \) can not be proven to be true, \( \neg f \) means that \( f \) is known to be not true. The operator ‘not’ is very useful and powerful, and is one of the main feature that makes AnsProlog one of the most suitable knowledge representation language.

Now let us consider how we can use AnsProlog to reason about actions. Suppose we have the knowledge that: action \( a \) causes \( f \) to be true and \( q \) to be false (in the state reached after the execution of \( a \)) if \( p \) is true in the state where \( a \) is executed. Now suppose \( q \) and \( p \) are known to be true, and \( f \) is known to be false in time \( t \), and we are interested in finding out about the state corresponding to \( t+1 \). We can express this by the following AnsProlog rules.

\[
\begin{align*}
holds(p, t). &\quad holds(q, t). \\
holds(f, T + 1) &\leftarrow holds(p, T), occurs(a, T). \quad (1) \\
ab(q, T) &\leftarrow holds(p, T), occurs(a, T). \quad (2) \\
holds(X, T + 1) &\leftarrow holds(X, T), \neg ab(X, T). \quad (3)
\end{align*}
\]

Here we reason using the closed world assumption (CWA), which says if we can not prove \( p \), then we assume \( p \) to be false. Now the above facts and rules will allow us to infer that \( f \) (which was false in \( t \)) will be true in \( t+1 \) (by using rule 1), \( p \) will remain true in \( t+1 \) (by rule 3), and \( q \) will not be true in \( t+1 \) (by 2 we get \( ab(q, t) \) which blocks 3). Now consider the case where we do not know whether \( p \) is true in time \( t \) or not. This can be done\(^6\) by modifying the rules appropriately.
3 The BioSigNet-RR system

We have an initial implementation of a system for representing and reasoning about signal networks. We refer to our system as BioSigNet-RR, where BioSigNet means ‘biological signal networks’ and RR denotes ‘representation and reasoning’. The interface language for our system is based on action languages (Section 2.1), and the implementation of the various reasoning mechanisms is done using AnsProlog (Section 2.2). We now illustrate the usability of our system by analyzing a NFκB related pathway. We start with a short description of this pathway.

3.1 Overview of NFκB related signaling

We will represent some aspects of NFκB signaling, namely its activation by degradation of IκB, the TNF induced activation of NFκB, pro- and anti-apoptotic signaling via TNF-R1, and the regulation of apoptotic signaling by FLIP protein.

In the inactive state, NFκB is kept in the cytoplasm by IκB. IκB can be phosphorylated (at serine residuals Ser-32 and Ser-36), which signals the ubiquitination and degradation of IκB by the proteosome. Once IκB have degraded, NFκB is free, and becomes active.

Tumor necrosis factor-α (TNF-α) can trigger both cell death and cell survival signals. Upon binding with TNF-α, TNF-R1 undergoes trimerization. The DDs recruit an adapter protein, TNF-R-associated death domain (TRADD). TRADD serves as a platform to recruit adaptor proteins, including the Fas-associated death domain protein (FADD), TNF-receptor-associated factor 2 (TRAF2), and receptor-interacting protein (RIP).

The adapter protein FADD contains an N-terminal death effector domain (DED) that can bind to the homologous DEDs of procaspase-8. FADD recruits procaspase-8 to form a death inducing signal complex (DISC). Procaspase-8 is then cleaved to yield active caspase-8, which in turn triggers apoptosis.

IKK is a IκB kinase, which can induce NFκB activation by phosphorylating IκB. TRAF2 is capable of interacting with downstream signaling molecules such as NFκB-inducing kinase (NIK). NIK phosphorylates IKK at serine 176 thus activating IKK, which results in NFκB activation. RIP has been implicated to inhibit the NIK activity thus, inhibiting NFκB activation.
3.2 System overview

As we mentioned earlier, our system includes two components: (1) a script language by which users can conveniently describe signal network knowledge and queries; (2) an AnsProlog (Smodels) program to encode the knowledge and to compute answers for queries.

The system expects two input scripts from users: the first script encodes a signaling knowledge base and the second encodes a query. The system output looks like

\$./signet.pl nfkb-rep.sn prediction.sn
Evaluating prediction ...
Computing positive models ...
Positive model found!
Computing negative models ...
Negative model not found!
Answer: the prediction is true.

In the following we describe the script language through the example of NFκB signaling.
3.3 The knowledge base (KB) script

The alphabet - the set of fluent and action symbols - of the KB are defined by the following constructs:

- `<fluent> some_property.`
- `<action> some_action.`

The negation of a fluent `some_fluent` is denoted by `-some_fluent.` A conjunction of fluents is written as a list of the fluents separated by `;`. The causal laws describing effect of actions and causal relation between fluents have the following syntax:

- `first_property; second_property; ... <causes> some_property.`
- `some_action <causes> a_property <if> prop_A; prop_B; ... prop_Z.`

To describe triggering of an action by another, we write:

- `1st_property; 2nd_property; ... n_th_prop <triggers> some_action.`
- `prop_1; prop_2; ... m_th_prop <norm_triggers> some_action.`

Finally, we have the description for inhibitions of actions:

- `some_condition <inhibbits> some_action.`

Let us consider examples taken from the KB script of NFκ signaling. A NFκB related signal starts with the binding of TNF-α ligand to TNF-R1 receptor. Upon the binding, TNF-R1 is trimerized. Hence, we define an action for the binding and a fluent for the receptor being trimzerized:

- `<action> bind(alpha_tnf,rec_tnf).`
- `<fluent> trimerized(rec_tnf).`

Once TNF-R1 is trimerized, the protein TRADD will be recruited to the receptor:

- `<action> bind(rec_tnf,tradd).`
- `<fluent> bound(rec_tnf,tradd).`

Note that `-bound(rec_tnf,tradd)` is necessary in the left side of `<triggers>`, otherwise the action `bind(rec_tnf,tradd)` will be triggered forever (again and again) by `trimerized(rec_tnf).` Once `bind(rec_tnf,tradd)` occurs, `-bound(rec_tnf,tradd)` will become true, which turns off the trigger.

The protein NIK is normally recruited to the TNF-R1 signaling complex by TRAF after the recruitment of TRAF itself. We represent this fact using defeasible trigger as follows:

- `<action> bind(tradd,traf).`
- `<action> bind(traf,nik).`
- `<fluent> bound(tradd,traf).`
- `bind(tradd,traf) <causes> bound(tradd,traf).`
bound(rec_tnf, tradd) ; bound(tradd, traf);  
-bound(traf, nik) <norm_triggers> bind(traf, nik).

Besides naturally being turned off after the occurrence of bind(traf, nik) this trigger can also be suppressed as soon as it became enable. We will come back to this trigger later when discussing planning to block NFκB activation via the suppression of the trigger.

3.4 The query script

A query script contains observations and queries. First, we need to define the initial situation and the chronological order of situations in which observations are made. For example, we have the initial situation labelled by \( t_0 \) and some later situation \( t_1 \).

\[ <\text{initial}> \; t_0. \]
\[ t_0 \; <\text{precedes}> \; t_1. \]

Situation can also be a number, which will be understood as a time point. The initial situation corresponds to the time point 0, so instead of denoting \( t_0 \) to be the initial situation, we can use the number 0 in its place.

There are observations about properties of the cellular environment and about the occurrences of actions. They take the following general forms:

- **some_property <observed_at> some_situation.**
- **some_action <occurs_at> some_situation.**

For example, to state that at the beginning, NFκB is bound to IkB, we write

\[ \text{bound(nf\_kappa\_b, i\_kappa\_b) <observed_at> t_0.} \]

The fluent bound(nf\_kappa\_b, i\_kappa\_b) can be defined in the KB script or on the fly in the query script.

We can write the observation about the binding of TNF-α to TNF-R1 receptor as follows.

\[ \text{bind(alpha\_tnf, rec\_tnf) <occurs_at> t_0.} \]

Currently BioSigNet-RR supports three kinds of queries, namely prediction, explanation and planning. Usually, we are concerned if some fluent (or its negation) is true at some situation, or always true from some situation, or eventually will be true after some situation.

With prediction, we want to know whether some property will certainly follow from the KB and given observations. An example of prediction query is

\[ \text{<predict> <always> -bound(fadd, procasp) <from> t_0.} \]

The query asks if the protein FADD is never bound to procaspase-8.

In explanation, we want to find the unknown truth values of fluents in the initial state, which helps in explaining particular observations. For example,

\[ \text{<explain> <always> bound(nf\_kappa\_b, i\_kappa\_b) <from> t_0.} \]
Here, we want to know from what initial situation NFκB always remains bound to IκB.

In planning, we want to find a sequence of interventions to achieve a goal property. For example, starting from a well-defined initial state, we would like to find a sequence of actions to keep NFκB bound to IκB.

\[
\text{<plan> <always> bound(nf\_kappa\_b,i\_kappa\_b) <from> t0.}
\]

Let us discuss in detail some examples for the reasoning queries.

### 3.5 Example of prediction

FLIP is a member of the family of viral protein vFLIP and a related cellular protein cFLIP. FLIP contains dead effector domains (DEDs) that are similar to the DEDs on FADD and pro-caspase-8. But the DEDs on FLIP cannot cleave themselves to yield caspase-8. One would predict that FLIP suppresses TNF-α-induced apoptosis via blocking activation of caspase-8.

First, we describe biological facts about FLIP.

\[
\begin{align*}
\text{<action> bind(fadd,flip).} \\
\text{<fluent> bound(fadd,flip).} \\
\text{bind(fadd,flip) <causes> bound(fadd,flip).} \\
\text{bound(fadd,flip) <inhibits> bind(fadd,flip).} \\
\text{bound(fadd,flip) <inhibits> bind(fadd,procasp).}
\end{align*}
\]

Then, we describe the observation about the recruitment of FADD to TNF-R1 signaling complex.

\[
\begin{align*}
\text{<initial> t0.} \\
\text{t0 <precedes> t1.} \\
\text{t1 <precedes> t2.} \\
\text{t1 <precedes> t3.} \\
\text{t2 <precedes> t3.} \\
\text{bound(nf\_kappa\_b,i\_kappa\_b) <at> t0.} \\
\text{bind(alpha\_tnf,rec\_tnf) <occurs\_at> t0.} \\
\text{bind(rec\_tnf,tradd) <occurs\_at> t1.} \\
\text{bind(tradd,fadd) <occurs\_at> t2.}
\end{align*}
\]

Finally, we ask if the caspase-8 is always is inactive after FLIP binds to FADD.

\[
\begin{align*}
\text{<predict> <always> -active(caspase8) <after> bind(fadd,flip) <from> t0.}
\end{align*}
\]

The output of BioSigNet-RR is as follows.

\$./signet.pl nfkb-rep.sn prediction.sn

Evaluating prediction ...
Computing positive models ...
Positive model found!
Computing negative models ...
3.6 Example of explanation

This is a hypothetical example adapted from\textsuperscript{12}. Imagine that we observe the recruitment of TRAF to TNF-R1 signaling complex. Then we expect that NFκB would be activated (by NIK-induced IKK activation). However, we observed that the NFκB activation is blocked. We suspect that some mutated form of TRAF may have been the cause. The mutation of TRAF may have altered its ability to recruit NIK, which is needed for NFκB activation.

To capture our hunch, we define a new fluent for TRAF being mutated and its causal relationship with the recruitment of NIK to TRAF.

\[ \text{mutated(traf)} \xrightarrow{\text{inhibits}} \text{bind(traf,nik)} \]

We state that events are observed to happen as usual.

\[ \text{initial} \ t_0. \]
\[ t_0 \xrightarrow{\text{precedes}} t_1. \]
\[ t_1 \xrightarrow{\text{precedes}} t_2. \]
\[ t_1 \xrightarrow{\text{precedes}} t_3. \]
\[ t_2 \xrightarrow{\text{precedes}} t_3. \]
\[ \text{bound(nf\_kappa\_b,i\_kappa\_b)} \at t_0. \]
\[ \text{bind(alpha\_tnf,rec\_tnf)} \occurs\at t_0. \]
\[ \text{bind(rec\_tnf,tradd)} \occurs\at t_1. \]
\[ \text{bind(tradd,traf)} \occurs\at t_2. \]

Finally, we ask for the explanation why NFκB is always inactive, that is, it is always bound to IκB.

\[ \text{explain} \xrightarrow{\text{always}} \text{bound(nf\_kappa\_b,i\_kappa\_b)} \from t_0. \]

The BioSigNet-RR answers as follows.

\[ ./signet.pl nfkb-rep.sn explanation.sn \]
\[ Finding \ explanation \ ... \]
\[ Observations \ can \ be \ explained \ if ... \]
\[ TRUE: \ mutated(traf). \]

3.7 Example of planning

Now assume that we know that TRAF can be mutated and the mutation can block NFκB activation. We would like to know when the mutation can be done to have the blocking effect. We define the mutation as an intervention to plan.
<fluent> mutated(traf).
<intervention> mutate(traf).
mutate(traf) <causes> mutated(traf).
mutated(traf) <inhibits> bind(traf,nik).

The goal is to have NFκB remains bound to IκB. So the planning query is written as

<plan> <always> bound(nf_kappa_b,i_kappa_b) <from> t0.

The BioSigNet-RR will find such a plan:

$ ./signet.pl nfkb-rep.sn planning.sn
Planning ... Plan found ...
mutate(traf) at 3

That is, we can block NFκB activation by mutating TRAF at time 3.

References