Searching in Protein State Space

Dietmar Seipel
Department of Computer Science
seipel@informatik.uni-wuerzburg.de

Jörg Schultz
Department of Bioinformatics, Biozentrum
joerg.schultz@biozentrum.uni-wuerzburg.de

University of Würzburg, Am Hubland, D–97074 Würzburg, Germany

Abstract. The increasing complexity of protein interaction networks makes their manual analysis infeasible. Signal transduction processes pose a specific challenge, as each protein can perform different functions, depending on its state. Here, we present a PROLOG and XML based system which explores the protein state space. Starting with state based information about the function of single proteins, the system searches for all biologically reasonable states that can be reached from the starting point. As facts of general molecular biology have been integrated, novel reasonable states, not encoded in the starting set, can be reached. Furthermore, the influence of modifications like mutations or the addition of further proteins can be explored. Thus, the system could direct experiments and allow to predict their outcome.

1 Introduction

Current large scale projects like the sequencing of genomes and the unravelling of protein *interaction networks* produce a wealth of data and give new insights into the molecular architecture of cells. Still, it is a huge step from data generation to the understanding of the molecular details of cellular networks. One of the primary challenges is the structured representation of a proteins function, a prerequisite for any computational analysis. Although many protein and nucleotide databases still use mainly human readable text as annotation, different approaches have been developed to structure this knowledge. These were as straightforward as defining specific keywords or as complicated as setting up an *ontology* for different aspects of protein function [Rzhetsky *et al.*, 2000]; probably, the most widely used of the latter approaches is from the Gene Ontology project [Consortium, 2008].

An additional level of complexity arises when describing not only the function of single proteins, but their interplay within the cellular network. Here, the function of one protein has to be seen in the context of its interactions or the pathways it is involved in. In the case of metabolic networks, methods for the representation of knowledge and the computation with function have been developed [Keseler *et al.*, 2009,Karp, 2001]. Here, the function of a protein can be seen as more or less constant. This contrasts typical eukaryotic signaling pathways, where the function of a protein and therefore its influence on the cellular networks changes for different input signals. This flexibility in function is usually implemented by describing the state of a protein when performing specific actions [Duan *et al.*, 2002,Schacherer *et al.*, 2001,Ratsch *et al.*, 2003]. Systems like the π–calculus have been developed, which allow to encode signaling pathways and to explore the effects of *mutations* [Regev *et al.*, 2001]. Still, these approaches as well as Petri nets [Grafahrend–Belau *et al.*, 2008] represent rigid networks and do not allow to find states not given to the system. The Biocham system [Fages *et al.*, 2004] offers automated reasoning tools for querying the temporal properties of interaction networks under all possible behaviours using Computation Tree Logic (CTL). [Baral, 2004] uses an action–style, knowledge based approach for supporting various tasks of reasoning (including planning, hypothetical reasoning, and explanation) about signaling networks in the presence of incomplete and partial information.

Here, we describe a *simulation approach* which is based on a logic representation of protein states. Its goal is to find all possible and biological acceptable pathways which can be derived from the given starting situation. As the system is aware of general concepts of molecular biology, it can 1) generate states not known to the system and 2) simulate the outcome of modifications like mutations or addition of inhibitors.

The rest of this paper is organized as follows: In Section 2, we present the concepts and techniques for manipulating XML structures in PROLOG. Section 3 provides formal representations of protein function, general biological concepts, and then it describes the central PROLOG search algorithm for protein networks. Two practical applications are shown in Section 4. Finally, Section 5 discusses the new approach.

2 Manipulation of XML Structures in PROLOG

The logic programming language PROLOG is very useful for representing, manipulating and reasoning about *complex structures*, such as the states obtained during the simulation of biological processes [Bratko, 2001,Clocksin & Mellish, 2003]. In particular, *search algorithms* can be implemented very nicely based on PROLOG's built–in concept of backtracking, and the handling and manipulation of the relatively complex intermediate data structures obtained during the simulation can be done in a very declarative way.

XML has been increasingly used for representing various forms of complex, semi–structured data, and for exchanging these data between different applications. E.g., in molecular biology, XML languages like SBML [Hucka *et al.*, 2003], MAGE–ML [Spellman *et al.*, 2002] and PSI Ml [Hermjakob *et al.*, 2004] have been developed. XML data are used by many tools in different application domains, and to some extend they are human–readable.

Thus, we have used XML as the data format for the complex states of our PROLOG based search algorithm. For querying and manipulating XML elements in PROLOG, we have developed the XML query, transformation, and update language FNQUERY [Seipel, 2002]. This combination of XML and PROLOG facilitates knowledge representation and software engineering drastically. The elegant data access to and update of XML data by path expressions in connection with the powerful reasoning facilities and the declarative programming style of PROLOG makes our application compact, flexible, and extensible. Another Prolog based Xml query language has, e.g., been proposed in [Alemendros–Jiménez *et al.*, 2007], without the necessary update functionality, however. Our approach is also superior to standard Xml processing tools such as XQuery or XSLT, since it is fully interleaved with the declarative – and general purpose – programming language Prolog, and since the update capabilities are better.

2.1 Xml Representations and Path Expressions

There exist various ways of representing Xml structures in Prolog. We have developed a Prolog term representation for Xml elements, which we call field notation, and a fact representation, which we call graph notation. These internal representations are described in [Seipel, 2002]; for the purposes of this paper it is sufficient to read Xml data from files and to write them to Xml files.

We use path expressions for localizing sub–structures of an Xml structure. The path language FnPath, which we have developed, is an extension of the well–known path language XPath, and XPath expressions can be mapped to equivalent FnPath expressions. Moreover, we can allow for more general path expressions, since we use Prolog – including backtracking and unification – for their evaluation. In the following we will describe the fragment of FnPath which we have used in this paper. The path expressions in this fragment directly correspond to equivalent XPath expressions.

A path expression $\pi = \sigma_1 \sigma_2 \ldots \sigma_n$ consists of one or more steps σ_i for localizing child elements or attributes of Xml elements. E.g., the path expression

```
/proteins/protein::[@name=Protein]
```

has two location steps. When this path expression is applied to an Xml element `Cell`, then the first step $\sigma_1 = \texttt{/proteins}$ selects the child element `Ps` of `Cell` with the tag `proteins`. Subsequently, the second step $\sigma_2 = \texttt{/protein::[@name=Protein]}$ of the path expression selects a child element `P` of `Ps` with the tag `protein`, such that `P` has an attribute `name` with the value `Protein`.

In general, a location step can be of the forms $\sigma = \texttt{@}\ \nu$, $\sigma = /\nu$, and $\sigma = /\nu :: \tau$, where ν is a node test and τ is a list of predicates. The second form is a short hand for the third, if the list τ is empty. The predicate `@name=Protein` in the example above also contains a location step `@name`, which selects the value of the attribute `name` of the element `P`; the predicate succeeds, if this value is `Protein`.

2.2 The Query Language

The Prolog based Xml query, transformation, and update language FnQuery is based on FnPath, and it has three sublanguages:

- FnSelect: selection of subelements/attributes,
- FnTransform: transformations like Xslt,
- FnUpdate: updates (insertion, deletion) of elements/attributes.

In this paper, we have extensively used FnSelect and FnUpdate for computing successor states during the simulation process. For this purpose, FnUpdate has been extended by insertion and deletion operations, which are very useful here.

Selection of Substructures. If `X` is an XML element in PROLOG representation and π is a path expression, then the assignment `Y := X` π selects a substructure of `X` and assigns it to `Y`. E.g., the following rule selects the location `Loc` of a `state` element for a given protein with the name `Protein` within a given cell `Cell`. The path expression π consists of 5 location steps, which are applied successively to `Cell`. The predicate `@name=Protein` in the second location step assures that the selected child has the proper value for the attribute `name`:

```
get_location(Protein, Cell, Loc) :-
   Loc := Cell/proteins/protein::[@name=Protein]
      /current_state/state@location.
```

Modification/Update of Substructures. If `X` is an XML element in PROLOG representation and π is a path expression, then the assignment `Y := X*[`π`:v]` modifies the substructure of `X` that is selected by π to the new value `v`. E.g., the assignment statement in the following rule changes the value of the attribute `location` to `Loc`.

```
set_location(Protein, Loc, Cell_1, Cell_2) :-
   Cell_2 := Cell_1 * [
      /proteins/protein::[@name=Protein]
      /current_state/state@location:Loc].
```

Insertion and Deletion of Substructures. The following PROLOG rules are used for modifying a given cell `Cell_1`. The first rule adds a new interaction element with the attribute value `Int` for `protein` within the `protein` element with the name `P`. The second rule deletes such an interaction element.

```
add_interaction(P, Int, Cell_1, Cell_2) :-
   Cell_2 := Cell_1 <+> [
      /proteins/protein::[@name=P]
      /current_state/state/interactions
      /interaction:[protein:Int]:[] ].

delete_interaction(P, Int, Cell_1, Cell_2) :-
   Cell_2 := Cell_1 <-> [
      /proteins/protein::[@name=P]
      /current_state/state/interactions
      /interaction::[@protein=Int] ].
```

3 Representation and Searching

In the following, we provide formal representations of protein function and general biological concepts, and then we describe the central PROLOG search algorithm for protein networks.

3.1 Formal Representation of Protein Function

It is a fundamental feature of proteins, especially those involved in signal transduction, that their function is tightly regulated. Depending, for example, on its own phosphorylation status, a protein kinase might phosphorylate different target proteins. Therefore, any representation of a proteins function has to allow different functions for a single protein, depending on its status. This status can be described by three features, namely the localisation, modifications, and interactions [Duan *et al.*, 2002,Ratsch *et al.*, 2003].

In our approach, we combine these features with a detailed description of the function performed within this state. Currently, the *localisation* mainly describes the organelle a protein can be found in, for example the cytoplasm or the nucleus. Within the list of *modifications*, the type, but also the position in the sequence is stored. Finally, the *interactions* describe a list of bound proteins. Depending on the value of these three features, the protein might perform different functions. These are encoded in a list of actions. Each action itself consists of the type of the action, the name of the involved protein and possible parameters.

Type	Subtypes	Parameters
moving	none	protein new location
interaction	binding dissociation	protein$_{1,2}$
modification	phosphorylation dephosphorylation	protein$_{1,2}$ position

Table 1. Implemented Actions

Currently, three general types of actions are implemented, where each type can contain different subtypes and parameters (Table 1). Within this concept it is important to note that we do not use any descriptions like *activation*, *stimulation* or *inhibition*, as these are interpretations and depend on the experiment performed. We describe the molecular details of a function a protein performs when in a given state. This might be interpreted as an activation to perform a function within a given state.

Having defined the state as the core structure, different states and their actions have to be put together with additional information to describe a proteins function. First, there are constant, that is state independent, features which have to be represented. Second, as we aim for qualitative simulation, the actual state of a protein has to be stored. Following, all states which trigger a specific action are listed. Finally, we allow for so called forbidden states. Especially in molecular biology, the knowledge of a proteins function frequently contains negative information like the fact that a protein will not be localised within the nucleus if bound to a defined other protein. By introducing forbidden states, we allow the handling of this information. Figure 1 shows the XML structure describing the function of a single protein.

```
<protein name="receptor" sequence="MDCQLSILLLLSCSVLD...">
   <current_state>
      <state location="transmembrane">
         <modifications/>
         <interactions>
            <interaction protein="jak"/> </interactions>
         <actions/>
      </state>
   </current_state>
   <action_states>
      <state location="transmembrane">
         <modifications/>
         <interactions mode="identical">
            <interaction protein="ligand"/>
            <interaction protein="jak"/> </interactions>
         <actions>
            <action type="interaction" subtype="binding">
               <protein name="receptor"/>
               <parameter protein="receptor"/>
               <status state="todo"/> </action> </actions>
      </state>
   </action_states>
   <forbidden_states>
      <state location="transmembrane">
         <modifications/>
         <interactions>
            <interaction protein="Grb2"/> </interactions>
      </state>
   </forbidden_states>
</protein>
```

Fig. 1. XML Representation of a Proteins Function – To represent all functions of a protein and allow simulation, the actual state of a protein (current_state) as well as all states leading to an action (action_states) have to be known. Furthermore, states not acceptable for the protein can be stored (forbidden_states). Features invariant to the protein, like the name or its sequence, are given as parameter to the protein tag.

3.2 Representation of General Biological Concepts

If one aims at qualitatively simulating signal transduction processes, it is of vital importance to take into account general biological concepts which can help to restrain the search space and keep the derived states consistent as well as biologically correct. Therefore, we have implemented the following concepts into our system.

Localisation. To describe different cellular localisations, we first have defined a set of accepted keywords describing localisations. In a second step, we have built, comparably to the GeneOntology, a hierarchy of these localisations using `subclasses/2` facts.

As we are modelling changes of localisations (moves), we furthermore define between which compartments direct moves are possible using `allowed/2` facts. For example, although nucleus and mitochondrion are both organelles embedded in the cytoplasm, no direct transitions between them are possible. A protein is assigned to one localisation for each state. A special case are transmembrane proteins, which reside in up to three different localisations, for example extracellular, membrane and cytoplasmic; this is represented by a `contains/2` fact.

```
localisations([
   localisation, intra, extra,
   membrane, transmembrane,
   cytoplasma, nucleus, ... ]).

subclasses(localisation, [
   intra, extra, membrane ]).
subclasses(transmembrane, [membrane]).

allowed(intra, extra).
allowed(cytoplasma, nucleus).
allowed(X, X).

contains(transmembrane, [
   extra, membrane, cytoplasma ]).

...
```

Interaction. As protein interactions build the backbone of any signal transduction process, a substantial amount of rules was generated to assure, that the modelled interactions are biologically sound. First, only proteins not already interacting (`unbound/2`) are allowed to start a new interaction and only interacting proteins can dissociate. Second, the predicate `matching_location_/3` assures that both interacting proteins reside in the same localisation; the values of the localisations do not have to be identical. Third, reciprocity of binding as well as dissociation events is assured by `add_interaction/3`. If one of multiple interacting proteins is moving, that is its localisation changes, all interacting proteins have to move. This assures consistency of localisation information; furthermore a translocation will not happen if any of the interacting proteins is not allowed within the new localisation. E.g., the PROLOG module `interaction` contains the following rule for bindings:

```
binding(S1, Action, S2) :-
   P1 := Action/protein@name,
   P2 := Action/parameter@protein,
   Proteins = [P1, P2],
   unbound(S1, Proteins),
   matching_location_(S1, Proteins, SA),
   add_interaction(SA, Proteins, SB),
   !,
   \+ forbidden_state(Proteins, SB),
   add(SB, Action, Proteins, S2).
```

Based on the facts implemented for localisation, we can handle cases where the information about the localisation of one protein is more specific than the other. In an example scenario, it might be known for protein A, that it is localised within the cytoplasm whereas protein B is only known to be somewhere within the cell (intracellular). Our system does not only allow this interaction, as the cytoplasm is a specification of intracellular, it updates the localisation information for protein B. This is also done iteratively for all interacting proteins in the case of complexes. Thereby, the system can improve on current knowledge.

Modification. Currently, the only implemented modification is phosphorylation. Here, our rules assure that within the phosphorylated position resides a correct amino acid, S, T or Y. Furthermore, it is assured, that the side which becomes phosphorylated is not already phosphorylated. Complementarily, only phosphorylated sites can become de–phosphorylated. In both cases, a modification includes the interaction (predicate `interaction:unbound/2` and first call of predicate `interaction/4`), the actual modification (predicate `phosphorylation_/3`) and a dissociation (second call of predicate `interaction/4`). Therefore, only proteins which are able to interact can be involved in a phosphorylation or de–phosphorylation reaction.

```
phosphorylation(S1, Action, S2) :-
   Protein_1 := Action/protein@name,
   Protein_2 := Action/parameter@protein,
   Proteins = [Protein_1, Protein_2],
   interaction:unbound(S1, Proteins),
   interaction(S1, Proteins, binding, SA),
   phosphorylation_(SA, Action, SB),
   interaction(SB, Proteins, dissociate, S2).
```

3.3 Searching Through Protein States

Thus far, we have described a static system for the representation of protein function based on states and some general rules concerning biological concepts. As it is the aim of the system to search through protein state space and to allow for qualitative simulations, the function of each protein has to be connected to and depend on the state of other proteins. This has been implemented for example by LiveDIP [Duan *et al.*, 2002] and the Molecule Pages database [Saunders *et al.*, 2008] by linking states with state transitions. Here, no state transitions are encoded. That is, no information like *protein*

A gets transferred from state 1 to state 2 when interacting with protein C has to be provided. Rather, we perform actions on proteins and let the protein itself *decide* whether its new state might trigger further actions. Thereby, we allow each protein to reach states, which are not explicitly encoded in its state list and are able to unravel novel pathways intrinsic to the analyzed protein interaction network.

Comparing States. A fundamental concept of the presented approach is the ability to compare different states. This is, for example, of importance for deciding, whether the current state of a protein *matches* a state which is connected to a new action. Obviously, the most straightforward approach would be to check for identity within all features of the state (localisation, interactions and modifications). This would lead to the same results as explicitly coding state transitions and would not allow to detect states not encoded within the system. To allow for a more explorative searching, we have defined a *subsumption* relationship for states.

```
subsumes_localisation(S, G) :-
   SLoc := S@location,
   GLoc := G@location,
   localisation:specifies(SLoc, GLoc).

subsumes_interactions(S, G) :-
   identical := G/interactions@mode,
   !,
   get_interactions(S, SInt),
   get_interactions(G, GInt),
   subset(GInt, SInt),
   subset(SInt, GInt).

subsumes_modifications(S, G) :-
   identical := G/modifications@mode,
   !,
   get_modifications(S, SMod),
   get_modifications(G, GMod),
   subset(GMod, SMod), subset(SMod, GMod).
```

In the case of localisations, the subsuming state needs to have a localisation which is either identical to or more specific than the general state. For example, a state with a cytoplasmic localisation will subsume a state with a intracellular localisation. Here, the general biological knowledge about the compartments of the cell is substantial. For the interactions and modifications, the general state has to be a subset of the subsuming state. To allow more restrictive comparisons, both interaction and modification can be set to identical, which enforces identity between the general and the subsuming state. These rules allow to decide whether a protein is in a state, which can trigger a specified action.

Search Algorithm and Implementation. Having defined a subsumption relationship between states and actions associated with given states, which are called *soups* below, the system can now be used to search through protein state space. To start the search process, an action as well as the starting states of all proteins are needed. In the first step, the system tries to perform the action, taking into account the biological background given above. If the action can be performed, at least one protein will be in a novel state. The system now compares the current state of all proteins with all action states of the protein. If no current state is found which subsumes an action state (`soup_node_subsumes/2`), then the system tries to perform the action. If successful, a new round of search is initiated (`find_next_soup/4`). In each step, applications of the general biological concepts assure, that only correct states are reached. For example, a protein A might be in a state triggering an action to bind to protein B. This action will fail, if protein B in its current state is located in a different cellular compartment than protein A. The search process iterates until no protein is in a state triggering a new action. Here, the first solution of the search process has been found.

```
protein_state_search(S, _, S).
protein_state_search(S1, Actions_1, S3) :-
   M := S1@id, not(soup_node_subsumes(_, M)),
   find_next_soup(S1, Actions_1, S2, Actions),
   find_candidate_actions(S2, Actions, Cs),
   add_candidate_actions(Cs, Actions, Actions_2).
   protein_state_search(S2, Actions_2, S3).

find_next_soup(S1, As1, S2, As2) :-
   A := As1/action::[@id=Id, /status@state=todo],
   [Type, SubType] := A-[@type, @subtype],
   once(apply(Type:SubType, [S1, A, S])),
   As2 := As1 * [
      /action::[@id=Id]/status@state:done ],
   remember_search_edge(S1, S, S2).
```

Natural numbers are assigned as identificators to Soups, and the edges between two soups are asserted in the PROLOG database:

```
remember_search_edge(S1, S, S2) :-
   M := S1@id,
   soup_to_number_(S, N),
   S2 := S * [@id:N],
   assert(soup_edge(M, N)),
   !.
```

The system finds all solutions using PROLOG's *backtracking* mechanism. In each intermediate step, the system checks, whether another action than the one already performed might be triggered. If so, another search process is started, leading to further solutions of the system. Thus, a *depth first search* through the space of all possible protein states is performed.

Within the search process, care is taken to detect possible cycles which would lead to infinite searches. Therefore, the search is stopped if an action already performed is triggered.

```
find_candidate_actions(Soup, Actions, Candidates) :-
   findall( Action,
      ( Protein := Soup/proteins/protein,
        check_action(Protein, Action),
        \+ performed(Action, Soup),
        \+ action_already_found(Actions, Action) ),
      Candidates_2 ),
   sort(Candidates_2, Candidates).

check_action(Protein, Action) :-
   C_State := Protein/current_state/state,
   A_State := Protein/action_states/state,
   protein_state_subsumes(C_State, A_State),
   Action := A_State/actions/action.

performed(Action, Soup) :-
   Test_Action := Soup/pathway/step/action,
   action:identical(Action, Test_Action).

action_already_found(Actions, Action) :-
   Action_2 := Action * [@id:_],
   Action_2 := Actions/action.
```

Interface / Visualization. Already in comparatively small biological systems, many end states can be reached. Obviously, any two of these end states can be identical, as each can be reached by different combinations of actions. To allow for the evaluation of the results and decrease the redundancy, we have developed a visualization in Swi–Prolog [Wielemaker, 2009], cf. Figures 2 and 3. Here, each combination of protein states, which can be reached is represented by a circle. If two states are identical, then the nodes representing these states are combined leading to a collapse of the original search tree.

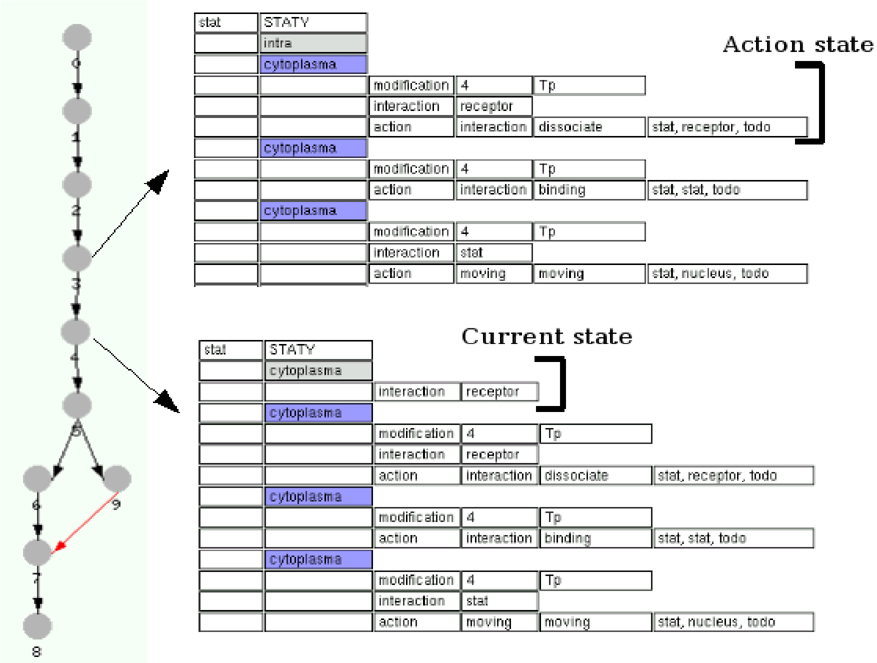
For each node, not only the state of all proteins, but also all pathways leading to the node are given. As the representation is based on the same Xml structure as the input, it might be used as input for other programs, e.g. visualizing the pathways in a more detailed way.

4 Applications

To show, that the proposed system is indeed able to represent signaling pathways, we evaluate two test cases, the Jak–Stat and the MAP kinase pathway. In addition to finding novel states, it was a fundamental goal of the presented system to allow for *qualitative simulations* of modifications of the signal transduction pathways; here, the implemented general biological concepts are of major importance.

4.1 Jak–Stat Pathway

The Jak–Stat signalling pathway provides one of the fastest routes from a cell surface receptor to the nucleus [O'Shea *et al.*, 2002]. Still, all major principles of signalling pathways, dimerization, phosphorylation and translocation are implemented within this short pathway. The involved proteins are the receptor, a ligand, the Jak protein kinases and the Stats. For each of the proteins we have defined a starting state as well as states which lead to further actions. An example is the rule that a phosphorylated Stat can homodimerize. The state searching procedure was started with an interaction of the ligand with the receptor.



Action state

stat	STATY				
	intra				
	cytoplasma				
		modification	4	Tp	
		interaction	receptor		
		action	interaction	dissociate	stat, receptor, todo
	cytoplasma				
		modification	4	Tp	
		action	interaction	binding	stat, stat, todo
	cytoplasma				
		modification	4	Tp	
		interaction	stat		
		action	moving	moving	stat, nucleus, todo

Current state

stat	STATY				
	cytoplasma				
		interaction	receptor		
	cytoplasma				
		modification	4	Tp	
		interaction	receptor		
		action	interaction	dissociate	stat, receptor, todo
	cytoplasma				
		modification	4	Tp	
		action	interaction	binding	stat, stat, todo
	cytoplasma				
		modification	4	Tp	
		interaction	stat		
		action	moving	moving	stat, nucleus, todo

Fig. 2. Representation of the Jak–Stat Pathway

Figure 2 shows all soups (combinations of proteins states) which have been reached from this initial condition. Most interestingly, there are two ways how node 7 can be reached. Inspection of this node reveals, that the difference between the two pathways is the order in which the dissociation of Stat from the receptor and the dimerization of two Stats are handled.

- With the knowledge given to system, two pathways have been found leading to node 7. Whereas in the first (via node 6), the phosphorylated Stat dissociates from the receptor before dimerization with another phosphorylated Stat, the pathway via node 9 reverses these steps.
- The detailed information underlying nodes 3 and 4 is selected and shown in an HTML table. The annotation is automatically improved. After binding to the receptor, the localisation of Stat is specified from *intracellular* to *cytoplasmic* (shown in grey).

It is an important feature of our system, that states which are not explicitly given in the *knowledge base* can be generated. This happens for example at node 9, which describes a protein complex of the ligand, a dimerized receptor, Jak and Stat. Additionally, Stat itself is a dimer. The existence of such a protein complex was not given within any state described in the knowledge base.

In addition to the generation of novel states, the system can also help in refining current knowledge. At the beginning of the search process, Stats location is annotated as *intracellular*. Within node 4, Stat binds to the receptor, which is annotated as *transmembrane* protein. As in the current implementation transmembrane proteins consist of an extracellular, a transmembrane and a cytoplasmic region, the system deduces, that Stat has to be cytoplasmic, if it binds as an intracellular protein to a transmembrane protein.

This example illustrates, that information about important states is not only sufficient to reconstruct pathways, also novel states as well as improvements of current knowledge can be automatically generated.

4.2 MAP Kinase Pathway

The MAPK pathway represents one of the oldest signalling cascades conserved between yeast and human [Bardwell, 2005,Wang & Dohlman, 2004]. Its core is a cascade of phosphorylation of protein kinases.

In total, 167 possible soups (combinations of protein states) were found. As the integration of these states and the pathways leading to them shows (Figure 3), there are many different pathways leading to identical combinations of protein states. Directed by this analysis, experimental research might unravel which of these pathways are actually implemented by the yeast cell.

5 Discussion

Unraveling the molecular details of signal transduction processes is an important step towards understanding the regulation of cellular networks. Although it is the ultimate

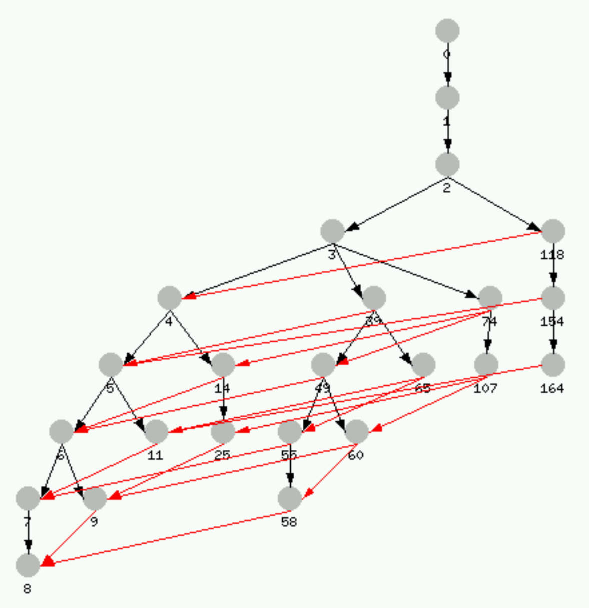


Fig. 3. Representation of the Yeast Mating Pheromone Response Pathway

goal of systems biology to quantitatively model the whole cell, especially in signal transduction we are still far from the enumeration of all involved molecular reactions. The identification of these reaction as well as the delineation of possible pathways is therefore a crucial prerequisite.

We have described a PROLOG system which, given a defined set of proteins and their functions, searches for all possible states encoded within a system and thereby finds all possible pathways using backtracking. The fundamental concept of the approach is the state of a protein, which is defined by its location, modifications and interactions. The whole system consists of three components: first an XML encoded state based representation of protein function. Second, a rule based representation of general biological concepts and third, a search algorithm implemented in PROLOG. The resulting system is not only able to search through protein state space, but furthermore to simulate the outcome of manipulations to the proteins. Declarative logic programming in PROLOG greatly simplified the system despite the *complex logic* behind the search algorithm and the complexity of the underlying *data structures*.

Contrasting other approaches [Saunders *et al.*, 2008,Zheng *et al.*, 2008], we do not encode any explicit state transitions. Instead, the protein *knows* about states which trigger further actions. If the protein is in a state that subsumes one of these states, an action is started, which might transform other proteins, leading to further actions. Thus, proteins can reach states, which have not been given to the system beforehand. These novel deduced states might be a good starting point for further experimental exploration of signal transduction networks.

A critical point within the approach is the collection of the functions of the involved proteins. Although manual *knowledge acquisition* will assure the highest quality, it is comparably slow. One might overcome this hindrance by adding automated function prediction for the involved proteins based for example on sequence analysis.

So far, the system has been tested only on smaller signal transduction networks. One possible challenge might be the increasing computational resources needed for larger system. To date, we rather see the collection of biological knowledge in the level of detail needed for the system as the limitation. More on the users point of view, the huge amount of detail might become hard to digest in the current representation. Thus, a more *graphical*, possibly *animated interface*, which might even allow to interfere with the system, would be the long term goal.

References

[Alemendros–Jiménez *et al.*, 2007] Almendros–Jiménez, J.M., Becerra–Terón, A., Enciso–Baños, F.J. (2007) Integrating XQUERY and Logic Programming. *Proc. 17th International Conference on Applications of Declarative Programming and Knowledge Management (INAP 2007) and 21st Workshop on (Constraint) Logic Programming (WLP 2007),* 136–147.

[Baral, 2004] Baral, C., Chancellor, K., Tran, N., Tran, N.L., Joy, A., Berens, M. (2004) A knowledge based approach for representing and reasoning about signaling networks, *Bioinformatics, 20* (Suppl. 1), i15–i22.

[Bardwell, 2005] Bardwell, L. (2005) A walk–through of the yeast mating pheromone response pathway. *Peptides, 26* (2), 339–350.

[Bratko, 2001] Bratko, I. (2001) PROLOG– Programming for Artificial Intelligence, 3rd Edition, Addison–Wesley, 2001.

[Clocksin & Mellish, 2003] Clocksin, W. F., Mellish, C. S. (2003) Programming in PROLOG. 5th Edition, Springer, 2003.

[Consortium, 2008] Consortium, G. O. (2008) The Gene Ontology project in 2008. *Nucleic Acids Res., 36*, D440–444.

[Duan *et al.*, 2002] Duan, X. J., Xenarios, I. & Eisenberg, D. (2002) Describing biological protein interactions in terms of protein states and state transitions: the LiveDIP database. *Mol Cell Proteomics, 1* (2), 104–116.

[Fages *et al.*, 2004] Fages, F., Soliman, S., Chabrier, N. (2004) Modelling and querying interaction networks in the biochemical abstract machine BIOCHAM, *Journal of Biological Physics and Chemistry, 4*, 64–73.

[Grafahrend–Belau *et al.*, 2008] Grafahrend–Belau, E., Schreiber, F., Heiner, M., Sackmann, A., Junker, B. H., Grunwald, S., Speer, A., Winder, K. & Koch, I. (2008) Modularization of biochemical networks based on classification of Petri net t–invariants. *BMC Bioinformatics, 9*, 90.

[Hermjakob *et al.*, 2004] Hermjakob, H., Montecchi–Palazzi, L., Bader, G., Wojcik, J., Salwinski, L., Ceol, A., Moore, S., Orchard, S., Sarkans, U., von Mering, C., Roechert, B., Poux, S., Jung, E., Mersch, H., Kersey, P., Lappe, M., Li, Y., Zeng, R., Rana, D., Nikolski, M., Husi, H., Brun, C., Shanker, K., Grant, S. G. N., Sander, C., Bork, P., Zhu, W., Pandey, A., Brazma, A., Jacq, B., Vidal, M., Sherman, D., Legrain, P., Cesareni, G., Xenarios, I., Eisenberg, D., Steipe, B., Hogue, C. & Apweiler, R. (2004) The HUPO PSI's molecular interaction format – a community standard for the representation of protein interaction data. *Nat. Biotechnology, 22* (2), 177–183.

[Hucka *et al.*, 2003] Hucka, M., Finney, A., Sauro, H., Bolouri, H., Doyle, J., Kitano, H., Arkin, A., Bornstein, B., Bray, D., Cornish–Bowden, A., Cuellar, A., Dronov, S., Gilles, E., Ginkel, M., Gor, V., Goryanin, I., Hedley, W., Hodgman, T., Hofmeyr, J.–H., Hunter, P., Juty, N., Kasberger, J., Kremling, A., Kummer, U., NovÃÍre, N. L., Loew, L., Lucio, D., Mendes, P., Minch, E., Mjolsness, E., Nakayama, Y., Nelson, M., Nielsen, P., Sakurada, T., Schaff, J., Shapiro, B., Shimizu, T., Spence, H., Stelling, J., Takahashi, K., Tomita, M., Wagner, J., Wang, J. & Forum, S. (2003) The systems biology markup language (SBML): a medium for representation and exchange of biochemical network models. *Bioinformatics, 19* (4), 524–531.

[Karp, 2001] Karp, P. (2001) Pathway databases: a case study in computational symbolic theories. *Science, 293* (5537), 2040–2044.

[Keseler *et al.*, 2009] Keseler, I. M., Bonavides–Martínez, C., Collado–Vides, J., Gama–Castro, S., Gunsalus, R. P., Johnson, D. A., Krummenacker, M., Nolan, L. M., Paley, S., Paulsen, I. T., Peralta–Gil, M., Santos–Zavaleta, A., Shearer, A. G. & Karp, P. D. (2009) EcoCyc: a comprehensive view of Escherichia coli biology. *Nucleic Acids Research, 37*, D464–470.

[O'Shea *et al.*, 2002] O'Shea, J. J., Gadina, M. & Schreiber, R. D. (2002) Cytokine signaling in 2002: new surprises in the Jak/Stat pathway. *Cell, 109 Suppl.*, S121–131.

[Ratsch *et al.*, 2003] Ratsch, E., Schultz, J., Saric, J., Lavin, P., Wittig, U., Reyle, U. & Rojas, I. (2003) Developing a protein–interactions ontology. *Comparative and Functional Genomics, 4*, 85–89.

[Regev *et al.*, 2001] Regev, A., Silverman, W. & Shapiro, E. (2001) Representation and simulation of biochemical processes using the π–calculus process algebra. In *Pacific Symposium on Biocomputing*, 459–470.

[Rzhetsky *et al.*, 2000] Rzhetsky, A., Koike, T., Kalachikov, S., Gomez, S., Krauthammer, M., Kaplan, S., Kra, P., Russo, J. & Friedman, C. (2000) A knowledge model for analysis and simulation of regulatory networks. *Bioinformatics, 16* (12), 1120–1128.

[Saunders *et al.*, 2008] Saunders, B., Lyon, S., Day, M., Riley, B., Chenette, E., Subramaniam, S. & Vadivelu, I. (2008) The Molecule Pages database. *Nucleic Acids Research, 36*, D700–706.

[Schacherer *et al.*, 2001] Schacherer, F., Choi, C., Götze, U., Krull, M., Pistor, S. & Wingender, E. (2001) The TRANSPATH signal transduction database: a knowledge base on signal transduction networks. *Bioinformatics, 17* (11), 1053–1057.

[Seipel, 2002] Seipel, D. (2002) Processing XML–documents in PROLOG. In *Proc. 17th Workshop on Logic Programming (WLP 2002).*

[Spellman *et al.*, 2002] Spellman, P. T., Miller, M., Stewart, J., Troup, C., Sarkans, U., Chervitz, S., Bernhart, D., Sherlock, G., Ball, C., Lepage, M., Swiatek, M., Marks, W., Goncalves, J., Markel, S., Iordan, D., Shojatalab, M., Pizarro, A., White, J., Hubley, R., Deutsch, E., Senger, M., Aronow, B. J., Robinson, A., Bassett, D., Stoeckert, C. J. & Brazma, A. (2002) Design and implementation of microarray gene expression markup language (MAGE–ML). *Genome Biology, 3* (9), RESEARCH0046.

[Wang & Dohlman, 2004] Wang, Y. & Dohlman, H. G. (2004) Pheromone signaling mechanisms in yeast: a prototypical sex machine. *Science, 306* (5701), 1508–1509.

[Wielemaker, 2009] Wielemaker, J. (2009) SWI–PROLOG 5.0 Reference Manual and
Wielemaker, J., Anjewierden, A. (2009) Programming in XPCE/PROLOG,
`http://www.swi-prolog.org/`

[Zheng *et al.*, 2008] Zheng, S., Sheng, J., Wang, C., Wang, X., Yu, Y., Li, Y., Michie, A., Dai, J., Zhong, Y., Hao, P., Liu, L. & Li, Y. (2008) MPSQ: a web tool for protein–state searching. *Bioinformatics, 24*, 2412–2413.