Multiple Representations of Biological Processes

Carolyn Talcott[1] and David L. Dill[2]

[1] SRI International
clt@csl.sri.com
[2] Stanford University
dill@cs.stanford.edu

Abstract. This paper describes representations of biological processes based on Rewriting Logic and Petri net formalisms and mappings between these representations used in the Pathway Logic Assistant. The mappings are shown to preserve properties of interest. In addition a relevant subnet transformation is defined, that specializes a Petri net model to a specific query to reduce the number of transitions that must be considered when answering the query. The transformation is shown to preserve the query in the sense that no answers are lost.

Keywords: signal transduction, biological process, Pathway Logic, Rewriting Logic, Petri Net

1 Introduction

Pathway Logic [1–3] is an approach to modeling cellular processes based on formal methods. In particular, formal executable models of processes such as signal transduction, metabolic pathways, and immune system cell-cell signaling are developed using the rewriting logic language Maude [4, 5] and a variety of formal tools are used to query these models. An important objective of Pathway logic is to reflect the ways that biologists think about problems using informal models, and to provide bench biologists with tools for computing with and analyzing these models that are natural.

Using the reflective capabilities of Maude, several alternative representations are derived to support use of different tools for visualization and analysis. The Pathway Logic Assistant (PLA) manages these different representations and, using the IOP+IMaude framework [6], provides a user interface that supports visualization and interaction with the models, and access to tools such as the Pathalyzer for carrying out in silico experiments. In particular, PLA uses Petri nets which provide visual representations and algorithms for answering reachability queries interactively, displaying the results in a way that is natural for biologists.

Being able to use alternative representations with different expressive capabilities and tools for visualization and analysis is important both for managing complexity and being able to focus on different properties. In the presence of multiple representations it is crucial to be able move between representations in semantically meaningful ways, preserving relevant properties. In this paper we describe a class of rewriting logic models of biological processes and a mapping of these models to Petri net models. We show that this mapping preserves computations and satisfaction of temporal formulae. Finally, we describe transformations that specialize Petri net models to specific queries

by reducing the set of transitions that need to be considered, and show that transformations are safe in the sense that query results are not changed. Specializing simplifies a Petri net model and allows the user to focus attention on the part of the model relevant to the question under consideration.

Plan. The remainder of this paper is organized as follows. Related work is discussed in §2. To provide context and motivate the technical results, a brief overview of Pathway Logic and the ways that biologists can compute with and query Pathway Logic models is given in section 3. In section 4 the notion of occurrence-based rewrite theory is defined and mappings between such theories and Petri net models are defined and shown correct. The notion of subnet relevant for a particular query is introduced in section 5, and transformations for producing a safe approximation to the relevant subnet are defined. Section 6 concludes with a summary and discussion of future directions.

2 Related Work

Computational models of biological processes such as signal transduction fall into two main categories: differential equations to model kinetic aspects; and symbolic/logical formalisms to model structure, information flow, and properties of processes such as what events (interactions/reactions) are checkpoints for or consequences of other events.

Models of system kinetics based on differential equations use experimentally derived or inferred information about concentrations and rates to simulate changes in response to stimuli as a function of time [7–10]. Such models are crucial for rigorous understanding of, for example, the biochemistry of signal transduction. However, the creation of such models is impeded by the great difficulty of obtaining accurate intra-cellular rate and concentration information, and by the possibly stochastic nature of cellular scale populations of signaling molecules [11, 12]. Analysis of such models by numerical and probabilistic simulation techniques becomes intractable as the number of reactions to be considered grows [13]. Furthermore, for the present purpose the questions we want to ask of a model involve qualitative concepts such as causality and interference rather than detailed quantitative questions.

Symbolic/logical models allow one to represent partial information and to model and analyze systems at multiple levels of detail, depending on information available and questions to be studied. Such models are based on formalisms that provide language for representing system states and mechanisms of change such as reactions, and tools for analysis based on computational or logical inference. Symbolic models can be used for simulation of system behavior. In addition properties of processes can be stated in associated logical languages and checked using tools for formal analysis. A variety of formalisms have been used to develop symbolic models of biological systems, including Petri nets [14], the pi-calculus [15], statecharts [16], and rule-based systems such as rewriting logic [17]. Each of these formalisms was initially developed to model and analyze computer systems with multiple processes executing concurrently. Several tools for finding pathways in reaction and interaction network graphs have been developed. However as pointed out in [18], paths found in these graphs do have not much to do with biochemical pathways.

There are many variants of the Petri net formalism and a variety of languages and tools for specification and analysis of systems using Petri nets. Petri nets have a graphical representation that corresponds naturally to conventional representations of biochemical networks. They have been used to model metabolic pathways and simple genetic networks (e.g., see [19–26]). These studies have been largely concerned with dynamic or kinetic models of biochemistry. In [27] a more abstract and qualitative view is taken, mapping biochemical concepts such as stoichiometry, flux modes, and conservation relations to well-known Petri net theory concepts.

A pi-calculus model for the receptor tyrosine kinase/mitogen-activated protein kinase (RTK/-MAPK) signal transduction pathway is presented in [28]. BioSPI, a tool implementing a stochastic variant of the pi-calculus, has been used to simulate both the time and probability of biochemical reactions [29]. So far, symbolic/logical analysis tools have not be used to analyze BioSPI models.

In [30] a continuous stochastic logic and the probabilistic symbolic model checker, PRISM, is used to express and check a variety of temporal queries for both transient behaviors and steady state behaviors. Proteins modeled as synchronous concurrent processes, and concentrations are modeled by discrete, abstract quantities.

BioAmbients [31], an adaptation of the Ambients formalism for mobile computations has been developed to model dynamics of biological compartments. BioAmbient type models can be simulated using an extension of the BioSPI tool. A technique for analysis of control and information flow in programs has been applied to analysis of BioAmbient models [32]. This can be used, for example, to show that according to the model a given protein could never appear in a given compartment, or a given complex could never form.

Statecharts naturally express compartmentalization and hierarchical processes as well as flow of control among subprocesses. They have been used to model T-cell activation [33, 34]. Although Statecharts is a mature technology with a number of associated analysis and verification tools, it does not appear that these have been applied to the T-cell model. Live Sequence Charts [35] are an extension of the Message Sequence Charts modeling notation for system design. This approach has been used to model the process of cell fate acquisition during C.elegans vulval development [36].

P-systems is a multiset rewriting formalism that provides a built in notion of location. A continuous variant of P-systems is used in [37] to model intra-cellular signaling. Locations are used to represent compartmental structure of a cell. Abstract objects represent proteins and small molecules, with different objects used to represent different modifications / states of the same protein. The underlying relation between a protein and its modifications is not made explicit. A system state specifies the quantity of each object in each location. A rate function associates to each rule a function from system states to real numbers, representing the rate of the reaction in that state. This determines how a system state evolves over time. Such models can be used to predict concentration of objects, for example phosphorylated ERK, over time by a discrete step approximation method.

A simple formalism for representing interaction networks using an algebraic rule-based approach very similar to the Pathway Logic approach is presented in [38, 39]. The language has three interpretations: a qualitative binary interpretation much like the

Pathway Logic models; a quantitative interpretation in which concentrations and reaction rates are used; and a stochastic interpretation. Queries are expressed in a formal logic called Computation Tree Logic (CTL) and its extensions to model time and quantities. CTL queries can express reachability (find pathways having desired properties), stability, and periodicity. Techniques for learning new rules to achieve a desired system specification are described in [40].

BioSigNet (BSN) [41] is a system for representing and reasoning about signaling networks. A BSN knowledge base encodes knowledge about a signal network, including logical statements based on symbols termed fluents and actions. Fluents represent the various properties of the cell and its components while actions denote biological processes (e.g. biochemical reactions, protein interactions) or external interventions. The logical statements describe the impact of these actions on the fluents, how actions can be triggered or inhibited inside the cell. A BSN knowledge base is queried using a temporal logic language over propositions expressing presence or absence of particular fluents. Three classes of queries are identified: prediction (can a state be reached); explanation (find initial conditions that lead to a specified condition); and planning (determining when an action should occur in order to achieve a desired result). In [42] BSN is used to model the ERK signaling network.

Models that rely on quantitative information (BioSPI, PRISM, P-systems) are limited by the difficulty in obtaining the necessary rate data. Missing or inconsistent data (from experiments carried out under different conditions, and on different cell types) are likely to yield less reliable predictions. Models that abstract from quantitative details avoid this problem, but the abstractions may lead to prediction of unlikely behavior, or miss subtle interactions.

The Pathway Logic Assistant extends the basic representation and execution capability with the ability to support multiple representations, to use different formal tools to simplify and analyze the models, and to visualize models and query results. Other efforts to integrate tools for manipulating models include the Systems Biology Workbench [43] the Biospice Dashboard [44], IBM Discoverylink [45], and geneticXchange, Inc [46].

3 Pathway Logic

As mentioned above, Pathway Logic models of biological processes are developed using the Maude system [4, 5] a formal language and tool set based on rewriting logic. Rewriting logic [17] is a logical formalism that is based on two simple ideas: states of a system are represented as elements of an algebraic data type; and the behavior of a system is given by local transitions between states described by *rewrite rules*. The process of application of rewrite rules generates computations (also thought of as deductions). In the case of biological processes these correspond to paths. Using reflection, modules and computations are represented as terms of the Maude meta language. This makes it easy to compute with models and paths.

3.1 Pathway Logic Basics

Pathway Logic models are structured in four layers: (1) sorts and operations, (2) components, (3) rules, and (4) queries. The *sorts and operations* layer defines the main sorts, subsort relations, and operations for representing cell states. The sorts of entities include `Chemical`, `Protein`, `DNA`, `Complex`, and `Enclosure` (cells and other compartments). These are all subsorts of the sort, `Soup`, that represents 'liquid' mixtures, as multisets. The sort `Dish` is introduced to encapsulate a soup as a state to be observed. Post-translational protein modification is represented by terms of the form `[P - mods]` where `P` is a protein and `mods` is a set of modifications. Modifications can be abstract, just specifying being activated, bound, or phosphorylated, or more specific, such as, phosphorylation at a particular site. For example, the term `[Cas - act]` represents the activation of the protein `Cas`. A cell state is represented by a term of the form `{CM | cm { cyto }}` where `cm` stands for a soup of entities in or at the cell membrane and `cyto` stands for a soup of entities in the cytoplasm.

The *components* layer specifies particular entities (proteins, chemicals, DNA) and introduces additional sorts for grouping proteins in families. For example `ErbB1L` is declared to be a subsort of `Protein`. This is the sort of `ErbB1` ligands whose elements include the epidermal growth factor `EGF`. The *rules* layer contains rewrite rules specifying individual signal transduction steps representing processes such as activation, phosphorylation, complex formation, or translocation. The *queries* layer specifies initial states and properties of interest.

Below we give a brief overview of the representation in Maude of signal transduction processes, illustrated using a model of Rac1 activation. This model and several others are available as part of the Pathway Logic Demo available from the Pathway Logic web site `http://pl.csl.sri.com/` along with papers, tutorial material and download of the Pathway Logic Assistant tool.

3.2 Modeling Activation of Rac1 in Pathway Logic

Rac1 is a small signaling protein of the Ras superfamily. It functions as a protein switch that is "on" when it binds the nucleotide triphosphate GTP, and "off" when it binds the hydrolysis product GDP. The Pathway Logic model of Rac1 activation was curated using [47] and many other references (cited as metadata associated with individual rules). In the following we show an initial state for study of Rac1 activation and two example rules, and briefly sketch some of the ways one can compute with the model. The initial state (called `rac1demo`) is a dish `PD( ... )` with a single cell and two stimuli in the supernatant, `EGF` and `FN`, represented by the following term.

```
rac1demo = PD(FN EGF
{CM | EGFR Ia5Ib1 Src PIP2 [Actin - poly][HRas - GDP][Rac1 - GDP]
        {Crk2 Erk2 Mek1  PI3K Shp2 bRaf C3g Dock Sos1
         Cas E3b1 Elmo Eps8 Fak Gab1 Grb2 Vav2 }} )
```

The cell membrane (shown on the line beginning `CM`) has an EGF receptor (`EGFR`) and an integrin (`Ia5Ib1`) that binds to `FN`. The term `[Rac1 - GDP]` represents the Rac1

protein in its 'off' state. The cell cytoplasm (shown on the last two lines) contains additional proteins that participate in the signaling process.

One way to activate Rac1 begins with the activation of the EGFR receptor due to the presence of the EGF ligand. The following rule represents this signaling step.

```
rl[1.EGFR.is.act]:
   ?ErbB1L:ErbB1L {CM | cm  EGFR        {cyto }} =>
   ?ErbB1L:ErbB1L {CM | cm [EGFR - act] {cyto }} .
   *** ErbB1Ls are AR EGF TGFa Btc Epr HB-EGF
```

The term ?ErbB1L:ErbB1L is a variable ranging over the sort ErbB1L. The terms cm and cyto are variables standing for the remaining components in the membrane and cytoplasm, respectively. The rule matches a part of the rac1demo dish contents by binding the variable ?ErbB1L:ErbB1L to EGF, the variable cm to Ia5Ib1 ... [Rac1 - GDP] (every thing in the cell membrane except EGFR), and the variable cyto to the contents of the cytoplasm {Crk2 ... Vav2}. Applying the rule replaces EGFR by [EGFR - act] resulting in the dish

```
  PD(FN EGF
    {CM | [EGFR - act] Ia5Ib1 Src PIP2 [Actin - poly]
          [HRas - GDP][Rac1 - GDP]
        {Crk2 Erk2 Mek1  PI3K Shp2 bRaf C3g Dock Sos1
         Cas E3b1 Elmo Eps8 Fak Gab1 Grb2 Vav2}} )
```

The following is one of three rules characterizing conditions for the Rac1 switch to be turned on.

```
rl[256.Rac1.is.act-3]:
 {CM | cm [Cas - act][Crk2 - act][Dock - act] Elmo [Rac1 - GDP]
      {cyto }} =>
 {CM | cm [Cas - act][Crk2 - act][Dock - act] Elmo [Rac1 - GTP]
      {cyto }} .
```

This rule describes activation resulting from assembly of Elmo with activated Cas, Crk2, and Dock at the cell membrane. Executing the rule replaces [Rac1 - GDP] by [Rac1 - GTP], turning Rac1 on, and leaves the remaining components unchanged.

Maude provides several ways to compute with a model. One can rewrite an initial state such as rac1demo above, to see a possible final state, or search for all states satisfying some predicate. To find a path satisfying some (temporal logic) property the Maude model-checker can be used. The properties of interest for Pathway Logic are expressed in Maude as patterns matching states with specific proteins, possibly with modifications, occurring in particular compartments (called goals), or requiring that particular proteins do not appear (avoids).

Example 1: racAct3 Property. As an example, to find the path stimulated by FN alone, we define a property (called racAct3) that is satisfied when Rac1 is activated and the EGF stimulus is not used (EGFR is not activated), thus forcing the FN stimulus to be selected. The property racAct3 is axiomatized by assertions stating which dishes satisfy the property (the relation |=) using patterns such as the following.

```
ceq PD (out:Soup
        {CM | cm [Rac1 - GTP] {cyto}}) |= racAct3 = true .
    if not(cm has [EGFR - act])
```

The model-checker is asked to check the assertion that there is no computation satisfying this property and a path can be extracted from a counterexample if one is found.

3.3 The Pathway Logic Assistant

The textual representation of cell states and pathways quickly becomes difficult to use as the size of a model grows, and an intuitive graphical representation becomes increasingly important. In addition, it becomes important to take advantage of the simple structure of PL models when searching for paths and carrying out other analyses. A Pathway Logic model, such as the Rac1 model, meeting certain simple conditions, can be transformed into a Petri net model by specializing the rules to the model's initial state. Petri nets have a natural graphical representation and can be analyzed using special purpose analysis tools.

Our Petri net models are a special case of Place-Transition Nets given by a set of occurrences (places in Petri net terminology) and a set of transitions [48]. Occurrences can be thought of as atomic propositions asserting that a protein (in a given state) or other component occurs in a given compartment. A system state is a set of occurrences (called a marking in Petri net terminology), giving the propositions that are true. A transition is a pair of sets of occurrences. A transition can fire if the state contains the first set of occurrences. In which case the first set of occurrences is replaced by the second set. PL goal properties translate to Petri net properties expressed as occurrences that must be present (places to be marked) and avoids properties translate to occurrences that must not appear (places not to be marked) in a computation. Paths leading from an initial state to a state satisfying a set of goals can be represented compactly as a Petri net consisting of the transitions fired in the path, thus giving query results a natural graphical representation. Execution of the path net starting with the initial state, leads to a state satisfying the goals, and the net representation makes explicit the dependency relations between transitions: some can fire concurrently (order doesn't matter), and some require the output of other transitions to be enabled.

The Pathway Logic Assistant (PLA) manages the different model and computation representations and provides functions for moving from one representation to another, for answering user queries, displaying and browsing the results. The principle data structures are: PLMaude models, Petri net models, Petri subnets, PNMaude modules, computations (paths), and Petri graphs. Here we give an overview of the PLMaude and Petri net models and mappings between them, illustrated by the model of Rac1 activation. Details are given in the next section, where we define the notion of *occurrence-based rewrite theory* that abstracts the relevant features of PLMaude models, and specify the main properties required for mappings between and transformations of representations in order to preserve meaning. *Petri graphs* are used to represent Petri nets as data structures that have a natural visual representation. A Petri graph has two kinds of node: occurrence and rule. Edges connect nodes representing occurrences of a rule

premise (lhs) to the rule node and the rule node to the nodes representing occurrences of the rule conclusion (rhs).

PLMaude models are Maude modules, such as the modules specifying the model of Rac1 activation discussed in §3.2, having the four layer structure described in §3.1. A PLMaude model must also obey the *conservation law* that components can be modified, composed and decomposed, but are not created out of nothing. As discussed in §3.1-3.2, PLMaude states are represented as a mixture of cells and ligands where location of proteins and other chemicals is represented by the algebraic structure of a term. To make the Petri net structure explicit an alternative representation is defined using multisets of occurrences. An occurrence is a pair consisting of a protein, complex, or other chemical component and a location. The location uniquely identifies the position of the component within the algebraic term (modulo multiset equality). For example, the dish

```
PD(EGF {CM | EGFR {Erk2}})
```

is represented by the occurrences

```
< EGF,out > < EGFR,cm > < Erk2,cyto >
```

Although soups and occurrences are formalized as multisets, initial states contain only sets (no duplication) and we have required that PLMaude rules preserve this property.

A *Petri net model* is a pair (T, I) consisting of a set of transitions T, and an initial state I (a set of occurrences). Each transition consists of a rule identifier, a pair of occurrence sets (the pre-occurrences and the post-occurrences). The mapping of a PLMaude model, with a specified initial dish D, to a Petri net model first determines an upper approximation to the set of components that might occur in each dish location by a collecting operation. This is done by starting with D, and repeating the collection cycle until nothing new is collected. In the collection cycle, for each rule that can be applied to the current dish, the result of applying the rule is merged into the current dish (by adding any new components to each compartment). For example, applying the rule `[1.EGFR.is.act]` to the dish `rac1demo` in collecting mode would add `[EGFR -act]` to the membrane rather using it to replace `EGFR`. The set of transitions T is then the set of rule instances that apply to the collected dish, converted to occurrence pairs. For example the rule `[1.EGFR.is.act]` instantiated with `EGF` for `?ErbB1L:ErbB1L` is represented by the triple

```
(1.EGFR.is.act, < EGF, out > < EGFR, cm >,
                < EGF, out > < [EGFR - act], cm >)
```

The initial state I is the conversion of the dish D to the occurrence representation. Figure 1 shows the Petri net representation of the model of Rac1 activation produced by this mapping.

A *Petri subnet* is a tuple (T, I, G, A) consisting of a set of transitions, T, an initial marking, I a goal marking G, and an avoids set A. A Petri subnet specifies an analysis problem, namely finding a computation starting from the initial marking, and reaching a state with the goals marked, using the transitions in the given set, without ever marking

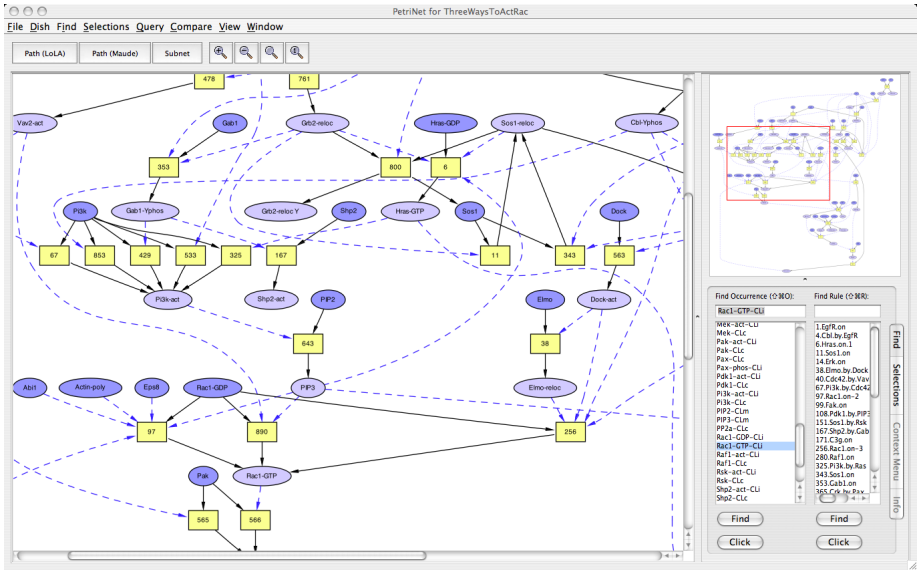
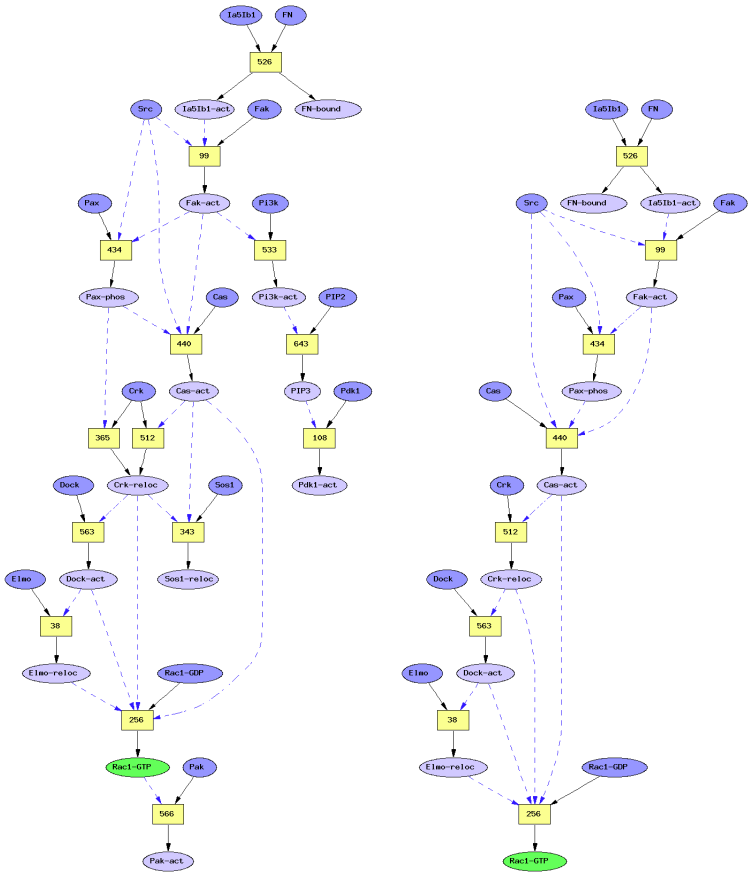


Fig. 1. Rac1 activation model as a Petri net. Ovals are occurrences, with initial occurrences darker. Rectangles are transitions. Two way dashed arrows indicate an occurrence that is both input and output. The full net is shown in the upper right thumbnail. A magnified view of the portion in the red rectangle is shown in the main view.

an avoid. Petri subnets are generated by a 'relevant subnet' computation that simplifies the specified analysis problem. Although a Petri subnet is a Petri net, it is only equivalent to the original net for a goal set that is a subset of G or an avoid set that is a superset of A. For example, for a goal that is not in G there may be a path in the original net, but not in the subnet, since transitions needed for this goal may have been discarded as not being relevant. Figure 2(a) shows the relevant subnet of the Rac1 activation model when the goal is activation of Rac1, avoiding activation of EGFR (Maude property `racAct3` in Example 1 of §3.2).

Computations are data structures used to represent system executions. We model a computation as a sequence of steps, each step being a triple consisting of a source state, a rule instance or transition that applies to that state, and a target state, the state resulting from the rule application. The target state of the ith step of a computation must be equal to the source state of the $i+1$st step. A compact representation of a computation is the Petri net consisting of the initial state together with the set of rule instances occurring as computation steps. We call this net a *path*. Figure 2(b) shows a path in the subnet of Figure 2(a). It can be executed as follows: If all of the ovals connected to a box by an incoming arrow (solid or dashed) are colored dark then color the outputs dark and make the inputs connected by solid arrows light color. Repeating this procedure, a state can be reached with Rac1 activated (Rac1-GTP colored dark).



(a) Subnet (b) Path

Fig. 2. FN stimulation of Rac1 activation.

3.4 Efficient Analysis of Petri nets

The path shown in Figure 2(b) was found by the LoLA (Low Level Analyzer) Petri net analysis tool [49, 50]. LoLA uses "stubborn set reduction", which is a technique that exploits the ease of determining the independence of certain transitions in the Petri nets. Specifically, a marking (state) m may have many possible successor markings, each resulting from a transition that is enabled in m. In many cases, transitions t_1 and t_2 are enabled in m because they are concurrent—either one can fire first, but they will both fire eventually. Nets with a lot of concurrency will also have a lot of states resulting from all the permutations of firing orders of concurrent transitions.

Stubborn set reduction prunes the state graph by recognizing concurrently enabled transitions and allowing them to be fired in a fixed order, instead of considering all the permutations. Subtle conditions must be met to ensure that a condition is reachable

in the reduced graph if and only if it is reachable in the original graph (the reader is referred to [51]). For highly concurrent graphs, stubborn set reduction can accelerate the solution of the reachability problem by many orders of magnitude.

For reachability queries on Pathway Logic nets, answering a reachability query that would have taken hours using a general purpose model-checking tool takes on the order of a second in LoLA—fast enough to permit interactive use. As an example, LoLA was run on 5 examples, with and without the stubborn set reduction option turned on. The examples were 5 queries on a single Petri net, each causing exploration of a different part of the network. The experiments were conducted on a an IBM ThinkPad X22 with an 800 MHz Pentium III CPU and 256 MB of RAM. All the examples completed within a fraction of a second with stubborn set reduction turned on. With stubborn set reduction turned off 4 of the examples completed within a fraction of a second. The 5th example was stopped after 28.5 minutes. At this point LoLA had generated 2,495,854 states, traversed 35,400,000 edges, and was using 500 MB of virtual memory, and all 256MB of physical memory.

In our experience, these results are typical. Without stubborn set reduction, LoLA either finishes quickly or runs for a very long time. With stubborn set reduction, it very reliably finishes in a fraction of a second on the many examples we have tried.

It is beyond the scope of this paper to compare different model-checking tools for Pathway Logic models. The interested reader can find such a comparison in [52]. We note that for the goals and avoids type queries, LoLA's performance is by far the best.

4 Relating PLMaude and Petri Nets

It is well known that Petri nets can be represented in rewriting logic [48]. The various forms that PLMaude models have taken as the modeling ideas have matured have led us to identify a special class of rewrite theories, called *occurrence-based* rewrite theories, that, restricted to terms reachable from a given initial term, have a natural representation as Petri nets. The idea is to build on the equivalence of the dish and occurrences representations of states and to identify the features of PLMaude models needed to ensure that the translation to the Petri net formalism preserves computations and goals-avoids properties. Furthermore, the resulting Petri net models can be transformed back into rewriting logic, again preserving computations and goals-avoids properties. In this section we define the mappings between PLMaude and Petri net models and the subnet reduction, and sketch proofs of correctness. These mappings are implemented in Maude and used in PLA.

4.1 Some rewriting logic notation

We first introduce some notation for talking about rewrite theories. A rewrite theory, $\mathcal{R}$, is a triple $((\Sigma, E), R)$ where (Σ, E) is an equational theory (for example, in order sorted logic) with sorts and operations given by Σ and equations E, and R is a set of rules of the form $(t_0 \Rightarrow t_1 \text{ if } c)$ where t_0, t_1 are terms, the rule premise and conclusion respectively, and c is a boolean term, the rule condition. Viewing PLMaude as a rewrite

theory, (Σ, E) is given by the first two layers (sorts and operations, components) and R is given by the rules layer.

A *context*, C, is a term with a single hole, denoted by $[\,]$, used to indicate the location of a rewrite application. $C[t]$ is the result of placing t in the hole of C.

A *substitution* σ is a finite mapping from variables to terms, preserving sort, and $\sigma(t)$ is the result of applying σ to the term t.

A *rule instance* is a triple $\rho = (r, C, \sigma)$ where r is a rule, C is context, and σ is a substitution. For a rule instance ρ as above we write $t \xrightarrow{\rho} t'$ if $t = C[\sigma(t_0)]$, $t' = C[\sigma(t_1)]$, and $\sigma(c)$ holds (rewrites to true). In this case we say that ρ is an application of r to t. We write $t \xrightarrow{r} t'$ if there is some $\rho = (r, C, \sigma)$ such that $t \xrightarrow{\rho} t'$. A computation over $\mathcal{R}$ is a sequence of rewrites of the form

$$\mathcal{R} \vdash s_0 \xrightarrow{\rho_1} s_1 \ldots \xrightarrow{\rho_k} s_k$$

with steps $s_{i-1} \xrightarrow{\rho_i} s_i$ for $1 \leq i \leq k$.

Note that rewriting is modulo E, that is the meaning of of the symbol '$=$' in the matching equations is defined by the equational theory E. The context makes explicit the location within a term where the rule applies. This is needed because when rewriting modulo equations the usual notion of path to a subterm of a syntax tree is not meaningful.

Example 2: Rewriting concepts. Consider the following:

- S_0 = `EGF {CM | EGFR {Mek1 [Mekk3 - act]}}`
- S_1 = `EGF {CM | EGFR {[Mek1 - act] [Mekk3 - act]}}`
- r_{mek} = `[Mekk3 - act]  Mek1 => [Mekk3 - act] [Mek1 - act]`
- C = `EGF {CM | EGFR {[]}}`

Then $\rho = (r_{mek}, C, \emptyset)$ is rule instance (with empty substitution, $\emptyset$) such that $S_0 \xrightarrow{\rho} S_1$. Note that S_0 can also be written `EGF {CM | {Mek1 [Mekk3 - act]} EGFR}`. Syntactically the subterm that matches the rule right hand side is at a different position in this case, but modulo associativity and commutativity the two ways of writing the term have the same meaning. The corresponding context `EGF {CM | {[]} EGFR}` is also equivalent to C, thus giving a representation of position that is independent to the representation of equivalence class.

4.2 Occurrence-based rewrite theories

There are five conditions to be met for a rewrite theory to be an occurrence-based rewrite theory, two conditions on the representation of state (**SC1** and **SC2**) and two conditions on rules (**RC1**, **RC2**) and one condition on the interaction of states and rules **SRC**).

In the following assume we are given a rewrite theory, $\mathcal{R}$, with distinguished sort $\mathbf{S}$ of elements representing system states to be analyzed.

SC1. The first condition is that $\mathbf{S}$ is generated from a base sort, by constructors such as the PLMaude enclosure constructors, in such a way that one only needs to know the 'location' of the base subterms to determine an element of $\mathbf{S}$. More precisely, we require that there be a base sort $\mathbf{B}$, a sort $\mathbf{L}$, of locations, and a sort $\mathbf{O}$ of occurrences, where elements of $\mathbf{O}$ have the form $<b, l>$ for base element b and location l, and two functions

$$s2o(\_) : \mathbf{S} \to \mathbf{P}_\omega[\mathbf{O}] \text{ and } o2s(\_) : \mathbf{P}_\omega[\mathbf{O}] \xrightarrow{\mathrm{p}} \mathbf{S}$$

such that $o2s(s2o(s)) = s$ where $\xrightarrow{\mathrm{p}}$ denotes partial functions and $\mathbf{P}_\omega[\mathbf{O}]$ denotes finite sets from $\mathbf{O}$.

SC2. We extend $s2o(\_)$ to contexts and terms with variables, by treating holes and variables as basic terms, and we require a function $cloc(C)$ that gives the location of the hole in a context. We also relativize the map from states to occurrences so that $s2o(t, l)$ gives the occurrences for t in a context with hole location l. Thus

$$s2o(C[t]) = O \cup (s2o(t, l)) \text{ where } l = cloc(C), s2o(C) = O \cup <[\,], l>.$$

The **SC2** requirement is that if $s2o(s_0) = O \cup s2o(\sigma(t_0), l)$ then we can find C such that $cloc(C) = l$ and $s_0 = C[\sigma(t_0)]$.

Example 3: Checking SC1, SC2 for PLMaude. In PLMaude, the base sort is called `Thing`, which has subsorts `Protein` and `Chemical` amongst others. Each membrane enclosed compartment has two associated locations, the membrane and the interior. For example, a cell has locations `cm` and `cyto`, and things not inside a cell have location `out`. Continuing the notation of Example 2 from §4.1, `EGF` has location `out` and `EGFR` has location `cm` and we have

- $s2o(S_0)$ =
  `< EGF,out > < EGFR,cm > < Mek1,cyto > < [Mekk3 - act],cyto >`
- $cloc(C)$ = `cyto`
- $s2o(S_2, cloc(C))$ = `< Mek1,cyto > < [Mekk3 - act],cyto >`

where S_2 is the left-hand side of r_{mek}. From the discussion in the previous sections, it is easy to see that PLMaude modules satisfy conditions SC1 and SC2.

SRC. We require that there is an associative and commutative operation on states $merge : \mathbf{S} \to \mathbf{S}$ such that rewriting is preserved by merging. Specifically, if $s_0 \xrightarrow{(r,C,\sigma)} s_0'$, $s_1 = merge(s_0, s')$ and $s_1' = merge(s_0', s')$ then $s_1 \xrightarrow{(r,C',\sigma')} s_1'$ for some C', σ', such that $\sigma/B = \sigma'/B$ and $cloc(C) = cloc(C')$ where σ/B is the restriction of σ to basic variables. Using associativity and commutativity we extend the $merge$ operation to sets: $merge(s, S)$ is the result of merging elements of S into s in some order.

Example 4: Merging. Continuing with the notation of example 2 we have

- $merge(S_0, S_1)$ = `EGF {CM | EGFR{Mek1 [Mek1 - act] [Mekk3 - act]}}`

RC1. We require that the variables appearing in rule terms either have basic sorts, or 'mixture' sorts (for example finite sets). This allows us to convert a rule application instance (r, C, σ) into a pair of occurrence sets that represent the actual change described by the rule. The mixture variables stand for the remaining basic terms and substructure at each location of interest that are not changed by the rule. Furthermore, we assume that the variables occurring in the rule condition have basic sorts.

Definition: Collection. Now we define a (partial) function that iteratively merges the reachable states into one state $\hat{s}$ in which each location contains all basic elements that could appear at that location in a reachable state. Given $s \in \mathbf{S}$ define $\hat{s}$ by

$$\hat{s} = s_k \text{ if } s_k = s_{k+1} \quad \text{where } s_0 = s \text{ and } s_{i+1} = merge(s_i, \{s' \mid (\exists \rho)(s_i \xrightarrow{\rho} s')\})$$

RC2. The final condition for $\mathcal{R}$ to be occurrence-based (relative to a choice of initial states) is that for any initial state s collection terminates, i.e. there is some k such that $s_k = s_{k+1}$.

4.3 Mapping occurrence-based rewrite theories to Petri nets

To define the mapping we need some Petri net notation. A transition τ over an occurrence set $\mathbf{O}$ is a pair $(O_i, O_o) \in \mathbf{P}_\omega[\mathbf{O}] \times \mathbf{P}_\omega[\mathbf{O}]$ (for simplicity we omit the transition labels). We define the pre- and post-occurrences of a transition as follows:

$$pre(O_i, O_o) = O_i \qquad post(O_i, O_o) = O_o.$$

The input and output occurrences are the pre- and post-occurrences with the shared occurrences removed.

$$in(O_i, O_o) = O_i - O_o \qquad out(O_i, O_o) = O_o - O_i.$$

Note that

$$in(O_i, O_o) \cap out(O_i, O_o) = \emptyset$$
$$(pre(O_i, O_o) - in(O_i, O_o)) = (post(O_i, O_o) - out(O_i, O_o)) = (pre(O_i, O_o) \cap post(O_i, O_o))$$

A Petri net model over occurrences $\mathbf{O}$ is a pair (T, I) where T is a set of transitions and $I \in \mathbf{P}_\omega[\mathbf{O}]$ is the initial state/marking. A computation over T is a sequence

$$T \vdash O_0 \xrightarrow{\tau_1} O_1 \ldots \xrightarrow{\tau_k} O_k$$

such that $pre(\tau_{i+1}) \subseteq O_i$ and $O_{i+1} = (O_i - in(\tau_{i+1})) \cup out(\tau_{i+1})$.

Definition: Rule2Transition. We extend the occurrence mapping to map rule instances to transitions.

$$s2o((t_0, t_1, c), C, \sigma) = (s2o(t_0, C, \sigma), s2o(t_1, C, \sigma))$$

where

$$s2o(t_0, C, \sigma) = s2o((\sigma/B)(t_0), cloc(C))^{\dagger}.$$

Where the † means to drop variable occurrences $<V, l>$ for mixture variables V. Note that if (r, C, σ) and (r, C', σ') are as in **RC2** then $s2o(r, C, \sigma) = s2o(r, C', \sigma')$. (See below for examples of $s2o(\_)$ applied to rules.)

Definition: OccB2Petri. Assume $\mathcal{R}$ is occurrence-based, with state sort $\mathbf{S}$, and s is an initial state. The Petri net model, $\mathcal{P}(\mathcal{R}, s)$, associated with $\mathcal{R}$ and s, has occurrences $s2o(\hat{s})$ (the result of collection), initial state $s2o(s)$, and a transition for each rule instance that applies to $\hat{s}$.

$$\mathcal{P}(\mathcal{R}, s) = (T_s, s2o(s)) \quad \text{where } T_s = \{s2o((r, C, \sigma)) \mid (\exists s')(\hat{s} \xrightarrow{(r,C,\sigma)} s')\}$$

Example 5: A Tiny model and its Petri net representation. We now introduce a tiny hypothetical model to illustrate the transformation to a Petri net in some detail. The resulting Petri net, TinyPN, will also be used in §5 to illustrate the transformations defined there. The tiny model defines two subsorts of `Protein`, `AP` and `BP`. There are six basic proteins: `A0,A1` of sort `AP`, `B0` of sort `BP`, and `E0,E1,C` of sort `Protein`.

```
sorts AP BP  . subsort AP BP < Protein .
ops A0 A1 : AP .  op B : BP .  ops E0 E1 C : Protein .
```

There are four rules. Rule `T0` expresses activation of `A0` by `E0` and recruitment of `A0` to the cell membrane. As in §3.2 `cm` and `cyto` are variables needed in order for the rule to apply to any cell with the specified components. Rule `T1` expresses activation and recruitment of any protein of sort `AP` by `E1`. Thus a variable `?A:AP` of sort `AP` is used in the rule. Rule `T2` says that any activated protein of sort `AP` can activate any protein of sort `BP`. Rule `T3` says that activated `A0` can activate protein `C`.

```
rl[T0]: {CM | cm E0 {cyto A0}}  =>
       {CM | cm E0 [A0 - act] {cyto}}

rl[T1]: {CM | cm E1 {cyto ?A:AP}}  =>
       {CM | cm E1 [?A:AP - act] {cyto}}

rl[T2]: {CM | cm [?A:AP - act] ?B:BP {cyto}} =>
       {CM | cm [?A:AP - act] [?B:BP - act] {cyto}}

rl[T3]: {CM | cm [A0 - act]  C {cyto}} =>
       {CM | cm [A0 - act] [C - act] {cyto}}
```

The initial state of the tiny model is given by the term `tinyDish`. It contains `B,C,E0,E1` at the cell membrane and `A0,A1` in the cytoplasm.

```
tinyDish = PD({CM | B C E0 E1 {A0 A1}}) .
```

The collection starting with the dish `tinyDish` using the rules `T0,T1,T2` yields the dish `tinyDish*`. Rule `T0` adds `[A0 - act]` to the membrane compartment, rule `T1` instantiated with `?A:AP` as `A1` adds `[A1 - act]` to the membrane compartment. The instantiation of rule `T1` with `?A:AP` as `A0` adds nothing new. Rule `T2` instantiated with `?B:BP` as `B0` adds `[B0 - act]` to the membrane compartment. For the dish `tinyDish` there is only one instantiation of rule `T2`. Rule `T3` adds `[C - act]` to the membrane compartment.

```
tinyDish* = PD({CM | [A0 - act] [A1 - act] [B - act] B
                     C [C -act] E0 E1
                     {A0 A1}}) .
```

The Petri net TinyPN has transitions `tinyT`, obtained by applying $s2o(\,)$ to rule instances for `tinyDish*`, and occurrences `tinyI`, the result of $s2o($`tinyDish`$)$. Here we added labels to the transitions for convenient reference. If there is more than one rule instance, the transition labels are indexed by instance numbers, just to keep labels unique.

```
TinyPN = (tinyT,tinyI)
tinyI = < B,cm >< C,cm >< E0,cm >< E1,cm >< A0,cyto >< A1,cyto >
tinyT =
   (T0,   < E0,cm> < A0,cyto > => < E0,cm > < [A0 - act],cm >)
   (T1.0, < E1,cm> < A0,cyto > => < E1,cm > < [A0 - act],cm >)
   (T1.1, < E1,cm> < A1,cyto > => < E1,cm > < [A1 - act],cm >)
   (T2.0, < [A0 - act],cm >< B,cm > =>
          < [A0 - act],cm >< [B - act],cm >
   (T2.1, < [A1 - act],cm >< B,cm >  =>
          < [A1 - act],cm >< [B - act],cm >
   (T3,   < [A0 - act],cm >< C,cm > =>
          < [A0 - act],cm >< [C - act],cm >
```

For example there are two instances of rule `T1`, $\rho_0 = ((t_0, t_1), C, \sigma_0)$ and $\rho_1 = ((t_0, t_1), C, \sigma_1)$ where

- t_0 = {`CM | cm E1 {cyto ?A:AP}`}
- t_1 = `\{CM | cm E1 [?A:AP - act] \{cyto\}\}`
- C = `PD([])`
- σ_0/B binds `?A:AP` to `A0`
- σ_1/B binds `?A:AP` to `A1`

Using the rule for transforming instances to transitions we obtain the transition labeled `T1.0` as follows.

$$
\begin{aligned}
&s2o((t_0, t_1), C, \sigma_0)\\
&= (s2o((\sigma_0/B)(t_0), cloc(C)), s2o((\sigma_0/B)(t_1), cloc(C)))\\
&= (s2o(\{\texttt{CM | cm E1\{cyto A0\}}\}, \texttt{out}),\\
&\quad\ s2o(\{\texttt{CM | cm E1[A0 - act]\{cyto\}}\}, \texttt{out}))\\
&= (\texttt{< E1, cm >< A0,cyto >,< E1,cm >< [A0 - act],cm >})
\end{aligned}
$$

Theorem: OccB2Petri. For $\mathcal{R}$ an occurrence-based rewrite theory and s an initial state, the mapping to Petri nets preserves computations. Specifically, if $(T_s, s2o(s)) = \mathcal{P}(\mathcal{R}, s)$, then

$$\mathcal{R} \vdash s = s_0 \xrightarrow{\rho_1} s_1 \ldots \xrightarrow{\rho_k} s_k \Leftrightarrow T_s \vdash s2o(s_0) \xrightarrow{s2o(\rho_1)} s2o(s_1) \ldots \xrightarrow{s2o(\rho_k)} s2o(s_k).$$

Proof Sketch. By induction on the computation length k. If $s_i \xrightarrow{\rho_{i+1}} s_{i+1}$ then $s2o(\rho_{i+1}) \in T_s$ by SRC. Let $\rho_{i+1} = (r, C, \sigma)$ with $r = (t_0, t_1, c)$ and $l = cloc(C)$. Then $s_i = C[\sigma(t_0)]$, $s_{i+1} = C[\sigma(t_1)]$, and for some occurrence set O $s2o(s_i) = s2o(\sigma(t_0), l) \cup O$ and $s2o(s_{i+1}) = s2o(\sigma(t_1), l) \cup O$. Thus $s2o(s_i) \xrightarrow{s2o(\rho_{i+1})} s2o(s_{i+1})$. Conversely, let $O_i \xrightarrow{\tau_{i+1}} O_{i+1}$, and by induction $O_i = s2o(s_i)$ for some reachable s_i. Also $\tau = (O_0, O_1) = s2o(r, C, \sigma)$ where (r, C, σ) applies to $\hat{s}$. We can find O' such that $O_i = O' \cup O_0$ and $O_{i+1} = O' \cup O_1$. By SC3 we can find C', σ' such that $s_i = C'[\sigma'(t_0)]$, and $s_i \xrightarrow{(r, C', \sigma')} s_{i+1}$ where $s2o(s_{i+1}) = O_{i+1}$.

Counterexample. To see that requirement (**SRC**) that merging preserves rewrites is needed, consider the following rule variants in the Pathway Logic language:

```
[r1]: {CM | Ras {cyto Rac}} => {CM | Ras [Rac - act]{cyto}}
[r2]: {CM | cm Ras {cyto Rac}} => {CM | cm Ras [Rac - act]{cyto}}
```

where `cyto` and `cm` are variables standing for any other components located in the cytoplasm or cell membrane respectively. Consider the state `{CM | Ras Grb2 {Src Rac}}` which can be obtained from `{CM | Ras {Src Rac}}` by a merge. The rule `r2` applies but `r1` does not, although `r1` applies to the 'before merge' state. Both rules transform to the same Petri net transition:

```
< Ras,CM > < Rac,Cyto > => < Ras,CM > < [Rac - act],CM >
```

which indeed applies to the corresponding occurrence state

```
< Ras,cm > < Grb2,cm ><  Rac,cyto > < Src,cyto >
```

Definition: Petri2RWL. The conversion of an occurrence Petri net to a rewrite theory is simple. If $(T_s, s2o(s)) = \mathcal{P}(\mathcal{R}, s)$, then $PS(\mathcal{R}, s)$ is the rewrite theory with the equational part of $\mathcal{R}$ extended with the definition of occurrences, and rules

$$\{O_i \Rightarrow O_o \mid (O_i, O_o) \in T_s\}$$

Theorem: Petri2RWL. The mapping PS preserves computations.

$$PS(\mathcal{R}, s) \vdash s2o(s) \xrightarrow{\tau_1} \ldots \xrightarrow{\tau_k} O_k \Leftrightarrow T_s \vdash s2o(s) \xrightarrow{\tau_1} \ldots \xrightarrow{\tau_k} O_k$$

5 Relating and transforming queries

5.1 Preservation of properties

The temporal logic used by the Maude model checker, LTL, is based on atomic propositions that can be defined by boolean functions in Maude. In the case of an occurrence-based rewrite theory, we restrict attention to propositions that are positive (goals) and negative (avoids) occurrence tests – basic component b occurs (or does not occur) at location l. These propositions translate to simple membership tests $<b, l> \in s2o(s)$ in the corresponding Petri net model. For example, the property `racAct3` presented in Example 1 of §3.2 contains one positive occurrence test (for the presence of `< [Rac1 - GTP], cm >`) and one negative occurrence test (for the absence of `< [EGFR - act], cm >`).

Let LTLO be the Maude LTL language with propositional part restricted to occurrence propositions. Let ψ be an LTLO formula expressed in the PLMaude language and let $s2o(\psi)$ be the same property expressed in terms of occurrence membership, lifting $s2o(\_)$ homomorphically (on syntax) to LTLO formulas.

Theorem: LTLO. Given an occurrence-based rewrite theory $\mathcal{R}$ and initial state s, let π be a computation of $\mathcal{R}, s$, π' be the corresponding computation of $\mathcal{P}(\mathcal{R}, s)$, and π'' be the corresponding computation of $PS(\mathcal{R}, s)$. Then for any LTLO formula ψ

$$\pi \models \psi \;\Leftrightarrow\; \pi' \models s2o(\psi) \;\Leftrightarrow\; \pi'' \models s2o(\psi)$$

and thus

$$(\mathcal{R}, s) \models \psi \;\Leftrightarrow\; \mathcal{P}(\mathcal{R}, s) \models s2o(\psi) \;\Leftrightarrow\; (PS(\mathcal{R}, s), s) \models s2o(\psi)$$

This is a consequence of the isomorphism of computations and the preservation of satisfaction of occurrence properties by the occurrence translations.

Note that the **LTLO** theorem implies that counterexamples are also preserved. This is important, since queries asking to find a computation having certain properties are answered by asking a model-checker to find a counterexample to the assertion that no such computation exists.

5.2 Relevant Subnets for Goals-Avoids Queries

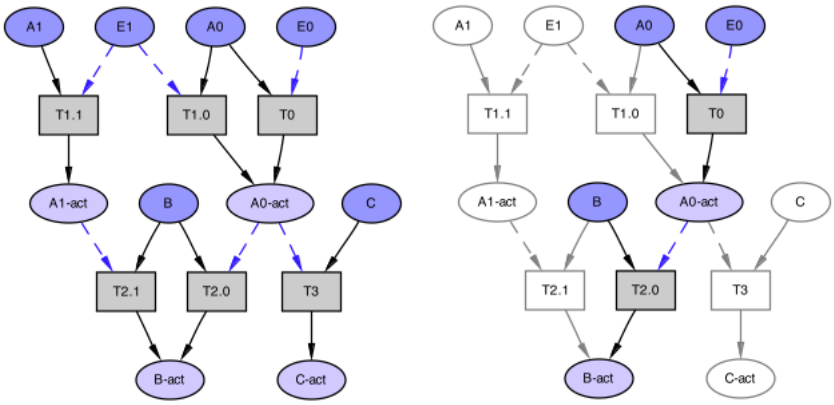
As indicated in § 3, we are especially interested in answering queries of the form "given initial state I, find a path that satisfies (G, A)" where (G, A) is a basic goals-avoids property with goals G and avoids A. We interpret this as meaning find a path (that is, a computation) starting with the initial state I, that reaches a state satisfying goals G, and such that no state in the path contains any occurrence of A. Without loss, we further require the path to be minimal, in the sense of not using irrelevant transitions. That is, if any transition is removed from the set generating the path, the remaining transitions do not generate a path satisfying the goals. Ideally we would like to focus attention on the subnet of transitions of a Petri net model that might appear in any minimal path satisfying that property. We call these the *truly relevant* transitions. This is of interest

both to help the biologist focus on a smaller set of transitions and to reduce the search space to be considered by an analysis tool.

Finding just the truly relevant transitions means finding exactly the minimal paths satisfying a given property, the problem we are trying to simplify. Thus we will look for a safe approximation, called the *relevant* transitions, that is a superset of truly relevant transitions set. Clearly, transitions that mention an occurrence to be avoided can be eliminated, as can transitions that do not contribute to reaching some goal, or transitions whose pre-set will not be a subset of a reachable state. In the following we define three transformations that formalize these intuitions. The first transformation removes transitions that mention an occurrence to be avoided. The second transformation is a backwards collection of transitions that contribute to reaching a goal, either because the post-occurrences contain a goal, or recursively contain a pre-occurrence of some contributing transition. The third transformation is a forward collection of transitions applicable to reachable states. Then given a Petri net (T, I), and a goals-avoids property (G, A), T is transformed/reduced to the corresponding set of relevant transitions $relTrans(T, I, G, A)$ by the following process:

$$\begin{array}{ccccccc} A & & G & & I & & \\ \downarrow & \text{avoids} & \downarrow & \text{backwards} & \downarrow & \text{forwards} & \\ [T] & \longrightarrow & [T/A] & \longrightarrow & [(T/A)^b_G] & \longrightarrow & [((T/A)^b_G)^f_I] \\ & \text{elimination} & & \text{collection} & & \text{collection} & \end{array}$$

We will show that any minimal path from initial state I meeting a goals-avoids property (G, A) using transitions in T, in fact uses only transitions in $relTrans(T, I, G, A)$, thus it is a safe approximation.



(a) TinyPN (b) Relevant Subnet

Fig. 3. Tiny Petri net (a) and a relevant subnet (b). Dark ovals represent initial state. In (b) the subnet is colored and the original net context is white.

Figure 3 show the Petri net `tinyPN` = (`tinyT`, `tinyI`) from Example 5 of §4.3. In the graphical form we use simple names to label (and refer to) occurrences, leaving the location part implicit. This will be used to illustrate the concepts and transformations discussed below.

Definition: Minimal Paths. Let I (initial state), G (goals), A (avoids) be occurrence sets such that $(I \cup G) \cap A = \emptyset$. The set $P(T, I, G, A)$ is the set of Petri net computations, π, that start from the initial state I, and reach a state containing all occurrences in G, using transitions in T without ever marking A.

$$\pi = O_0 \xrightarrow{\tau_1} \ldots \xrightarrow{\tau_k} O_k \in P(T, I, G, A) \Leftrightarrow O_0 = I \wedge G \subseteq O_k \wedge \bigwedge_{0 \le i \le k} O_i \cap A = \emptyset$$

π is minimal if there is no computation π' in $P(T, I, G, A)$ that uses a proper subset of the transitions used in π, and we let $mP(T, I, G, A)$ be the set of computations of $P(T, I, G, A)$ that are minimal.

Example 6: Minimal and non Minimal Paths. In `tinyPN` (Figure 3) the transitions `T1.1`, `T1.0` form a path to the goal `A0-act`, but it is not minimal as `T1.1` can be removed since `T1.0` alone is a path.

Lemma: Path Monotonicity. The set of minimal paths monotonically increases with increasing initial state and decreasing goals and avoids. Specifically, if $T \subseteq T'$, $I \subseteq I'$, $G' \subseteq G$, $A' \subseteq A$, then

$$P(T, I, G, A) \subseteq P(T', I', G', A')$$

and

$$mP(T, I, G, A) \subseteq mP(T', I', G', A')$$

Definition: Removing Avoids. Assume given T and A as above. The result of removing rules that mention an element of A is defined by

$$T/A = \{\tau \in T \mid (pre(\tau) \cup post(\tau)) \cap A = \emptyset\}$$

Example 7. Removing Avoids. Taking A to be `A1*`, removing the avoids from the transitions of `tinyPN` means removing `T1.1` and `T2.1`, leaving `T0`, `T1.0`, `T2.0`, and `T3`, that is

$$\texttt{tinyT}/A = \{\texttt{T0}, \texttt{T1.0}, \texttt{T2.0}, \texttt{T3}\}.$$

Lemma: Removing Avoids is Safe. If $\pi \in mP(T, I, G, A)$, then $\pi \in mP(T/A, I, G, A)$.
Proof. Since by definition no transition in $T - T/A$ could be used in π.

Definition: Backward collection. Assume given T, G as above. The backward collection T^b_G of T relative to G is defined by

$$T^b_G = \bigcup_{j \in \mathbf{Nat}} T^b_j \quad \text{where}$$

$$G_0 = G \qquad G_{j+1} = G_j \cup \bigcup_{\tau \in T^b_j} pre(\tau)$$

$$T^b_j = \{\tau \in T \mid out(\tau) \cap G_j \neq \emptyset\}$$

Note that for some n, $G_j = G_{j+1}$ for $j > n$ since T is finite and thus only finitely many increments can be made.

Example 8. Backwards Collection. Backwards collection of `tinyT` for goal `B-act`, $\texttt{tinyT}^b_{\texttt{B-act}}$, is `tinyT` minus `T3`. The can be seen from the following steps in the collection:

$G_0 = \texttt{B-act}$

$\texttt{tinyT}^b_0 = \{\texttt{T2.0}, \texttt{T2.1}\}$

$G_1 = \{\texttt{B-act}, \texttt{B}, \texttt{A0-act}, \texttt{A1-act}\}$

$\texttt{tinyT}^b_0 = \texttt{tinyT}^b_0 \cup \{\texttt{T0}, \texttt{T1.0}, \texttt{T1.1}\}$

As another example, for goals `A0-act` and `A1-act` we have

$$\texttt{tinyT}^b_{\{\texttt{A0-act},\texttt{A1-act}\}} = \{\texttt{T0}, \texttt{T1.0}, \texttt{T1.1}\}$$

Lemma: Backward Monotonicity. Backwards collection is monotonic in transitions and goals. That is, if $T \subseteq T'$ and $G \subseteq G'$, then $T^b_G \subseteq (T')^b_{G'}$.

The lemma **Backwards 1** captures the essence of the reason that a transition that appears in some minimal path for a set of goals is one produced by backwards collection.

Lemma: Backwards 1. If $O \xrightarrow{\tau_1} O_1 \xrightarrow{\tau_2} O_2$ and $pre(\tau_2) \cap out(\tau_1) = \emptyset$ then we can find O'_2 such that $O \xrightarrow{\tau_2} O'_2$. Furthermore, for any occurrence set G^*, if $out(\tau_1) \cap G^* = \emptyset$, then $G^* \cap O_2 \subseteq G^* \cap O'_2$.

Proof. With the assumptions of the lemma, $pre(\tau_2) \subseteq O$, letting $O'_2 = (O - in(\tau_2)) \cup out(\tau_2)$ we have, by definition of transition, the desired transition. Also by definition of transition, $O_2 = ((O - in(\tau_1) \cup out(\tau_1)) - in(\tau_2)) \cup out(\tau_2)$. Assuming $out(\tau_1) \cap G^* = \emptyset$ we have $G^* \cap O_2 = G^* \cap ((O - in(\tau_1) - in(\tau_2)) \cup out(\tau_2)) \subseteq G^* \cap O'_2$.

The lemma **Backwards 2** identifies conditions under which a sequence of transitions can be restarted at a new state. For backwards collection, the state of interest is one resulting from deleting an irrelevant transition, such as τ_1 in **Backwards 1**.

Lemma: Backwards 2. If $O \cap G \subseteq O' \cap G$, $pre(\tau) \subseteq G$ and $O \xrightarrow{\tau} O_1$, then we can find O'_1 such that $O_1 \cap G \subseteq O'_1 \cap G$ and $O' \xrightarrow{\tau} O'_1$.
Proof. By the assumptions, $pre(\tau) \subseteq O'$, so letting $O'_1 = (O' - in(\tau)) \cup out(\tau)$ we have the desired transition. Since $O_1 = (O - in(\tau)) \cup out(\tau)$, if $g \in O_1$ either $g \in out(\tau)$ or $g \in O - in(\tau) \subseteq O' - in(\tau)$. Thus $g \in O'_1$.

Theorem: Backward safety. If $\pi \in mP(T, I, G, A)$, then $\pi \in mP(T^b_G, I, G, A)$.
Proof Sketch. Let $\pi = I \xrightarrow{\tau_1} O_1 \ldots \xrightarrow{\tau_k} O_k \in mP(T, I, G, A)$. We show that $\tau_j \in T^b_G$ for $1 \leq j \leq k$. Suppose not. Let G^* be the union of the G_js in the definition of T^b_G, and let j be the largest number such that $\tau_j \notin T^b_G$. Thus $out(\tau_j) \cap G^* = \emptyset$. We construct $\pi' \in P(T, I, G, A)$ using fewer transitions, contradicting minimality of π. If $j = k$ then $G \subseteq O_{k-1}$ and π' is the first $k-1$ transitions of π. If $j < k$ then let $O'_j = O_{j-1}$, and $O'_{i+1} = (O'_i - in(\tau_{i+1})) \cup out(\tau_{i+1})$ for $j \leq i < k$. By maximality of j, $\tau_{i+1} \in T^b_G$ and thus $pre(\tau_{i+1}) \subseteq G^*$ for $j \leq i < k$. We claim that $O_{i+1} \cap G^* \subseteq O'_{i+1} \cap G^*$ and $O'_i \xrightarrow{\tau_{i+1}} O'_{i+1}$ for $j \leq i < k$. For $i = j$ this follows by backwards lemma 1 and for $i > j$ it follows by backwards lemma 2. Thus taking $\pi' = I \xrightarrow{\tau_1} O_1 \ldots \xrightarrow{\tau_{j-1}} O'_j \xrightarrow{\tau_{j+1}} \ldots \xrightarrow{\tau_k} O'_k$ we are done.

Definition: Forward collection. The forward collection T^f_I of T relative to I is defined by

$$T^f_I = \bigcup_{j \in \mathbf{Nat}} T^f_j \qquad I^f = \bigcup_{j \in \mathbf{Nat}} I_j \quad \text{where}$$

$$I_0 = I \qquad I_{j+1} = \bigcup_{\tau \in T^f_j} post(\tau)$$

$$T^f_j = \{\tau \in T \mid pre(\tau) \subseteq I_j\}$$

Again, we have that for some n, $I_j = I_n$ for $j \geq n$.

Example 9. Forward Collection. Suppose we remove `E1` from the inital state, call this I_1, then forward collection of `tinyT` from I_1 omits `T1.0`, `T1.1` and `T2.1`. Thus $\texttt{tinyT}^f_{I_1} = \{\texttt{T0}, \texttt{T2.0}, \texttt{T3}\}$.

Lemma: Forward Monotonicity. If $T \subseteq T'$ and $I \subseteq I'$, then $T^f_I \subseteq (T')^f_{I'}$

Theorem: Forward safety. If $\pi \in mP(T, I, G, A)$, then $\pi \in mP(T^f_I, I, G, A)$.
Proof. This is because for each transition τ_{j+1} in π, using the notation of the definition, $pre(\tau_{j+1}) \subseteq I_j$, and thus $\tau_{j+1} \in T^f_j$ for $0 \leq j < k$.

Definition: Relevant Subnet. The subnet of transitions from T relevant to initial state I, goals G, and avoids A is defined by

$$relTrans(T, I, G, A) = ((T/A)^b_G)^f_I.$$

Corollary: Relevant Subnet. If $\pi \in mP(T, I, G, A)$ is non-empty, then

$$\pi \in mP(relTrans(T, I, G, A), I, G, A)$$

Thus search for such paths can be carried out in the relevant subnet $relTrans(T, I, G, A)$. Note that if $G \not\subseteq I^f$ then $P(T, I, G, A)$ is empty.

Example 10. Relevant subnets. The relevant subnet of `tinyPN` for goals `B-act`, avoids `A1-act` and initial state $I_1 = \texttt{tinyI} - \texttt{E1}$

$$relTrans(\texttt{tinyT}, I_1, \texttt{B-act}, \texttt{A1-act}) = \{\texttt{T0}, \texttt{T2.0}\}$$

is shown in figure 3(b).

6 Summary and Future Work

The main contributions of the paper are: a definition of mappings between rewriting logic and petri net representations of biological processes (and similar concurrent processes) that satisfy certain conditions; proof that these mappings preserve properties of interest; and definition of a relevant subnet transformation that reduces the number of transitions that must be considered in search for a path satisfying a goals-avoids property.

As context we presented an overview of Pathway Logic illustrated with a model of Rac1 activation as Maude rules and the representation as a Petri net. We also discuss the advantages of analyses based on Petri nets.

As models grow in size, we expect to need to explore alternative path finding algorithms. Possibilities include employing more highly tuned model checkers, discovering new simplification and abstraction transformations, and developing constraint solving approaches to take advantage of the rapid advances being made in this area. Another big challenge is refining PLMaude models to incorporate semi-quantitative information about expression levels and relative preference for competing reactions, and to be able compute with and visualize the refined models in ways that are meaningful to working biologists.

Acknowledgments. The authors would like to thank the Pathway Logic team and the anonymous reviewers for helpful criticisms. The work was partially funded by NIH/NIGMS grant GM68146.

References

1. Eker, S., Knapp, M., Laderoute, K., Lincoln, P., Meseguer, J., Sonmez, K.: Pathway Logic: Symbolic analysis of biological signaling. In: Proceedings of the Pacific Symposium on Biocomputing. (2002) 400–412
2. Eker, S., Knapp, M., Laderoute, K., Lincoln, P., Talcott, C.: Pathway Logic: Executable models of biological networks. In: Fourth International Workshop on Rewriting Logic and Its Applications (WRLA'2002), Pisa, Italy, September 19 — 21, 2002. Volume 71 of Electronic Notes in Theoretical Computer Science., Elsevier (2002) `http://www.elsevier.nl/locate/entcs/volume71.html`.

3. Talcott, C., Eker, S., Knapp, M., Lincoln, P., Laderoute, K.: Pathway logic modeling of protein functional domains in signal transduction. In: Proceedings of the Pacific Symposium on Biocomputing. (2004)
4. Clavel, M., Durán, F., Eker, S., Lincoln, P., Marti-Oliet, N., Meseguer, J., Talcott, C.: Maude 2.0 Manual (2003) `http://maude.cs.uiuc.edu`.
5. Clavel, M., Durán, F., Eker, S., Lincoln, P., Martí-Oliet, N., Meseguer, J., Talcott, C.L.: The Maude 2.0 system. In Nieuwenhuis, R., ed.: Rewriting Techniques and Applications (RTA 2003). Volume 2706 of Lecture Notes in Computer Science., Springer-Verlag (2003) 76–87
6. Mason, I.A., Talcott, C.L.: IOP: The InterOperability Platform & IMaude: An interactive extension of Maude. In: Fifth International Workshop on Rewriting Logic and Its Applications (WRLA'2004). Electronic Notes in Theoretical Computer Science, Elsevier (2004)
7. Kohn, K.W.: Functional capabilities of molecular network components controlling the mammalian g1/s cell cycle phase transition. Oncogene **16** (1998) 1065–1075
8. Weng, G., Bhalla, U.S., Iyengar, R.: Complexity in biological signaling systems. Science **284** (1999) 92–96
9. Shvartsman, S.Y., Hagan, M.P., Yacoub, A., Dent, P., Wiley, H., Lauffenburger, D.: Autocrine loops with positive feedback enable context-dependent cell signaling. Am J Physiol Cell Physiol **282** (2002) C545–559
10. Smolen, P., Baxter, D.A., Byrne, J.: Mathematical modeling of gene networks. Neuron **26** (2000) 567–580
11. Lok, L.: Software for signaling networks, electronic and cellular. Science STKE **PE11** (2002)
12. Rao, C.V., Wolf, D.M., Arkin, A.P.: Control, exploitation and tolerance of intracellular noise. Nature **420** (2002) 231–237
13. Endy, D., Brent, R.: Modelling cellular behaviour. Nature **409** (2001) 391–395
14. Peterson, J.L.: Petri Nets: Properties, analysis, and applications. Prentice-Hall (1981)
15. Milner, R.: A Calculus of Communicating Systems. Springer Verlag (1980)
16. Harel, D.: Statecharts: A visual formalism for complex systems. Science of Computer Programming **8** (1987) 231–274
17. Meseguer, J.: Conditional Rewriting Logic as a unified model of concurrency. Theoretical Computer Science **96**(1) (1992) 73–155
18. Croes, D., Couche, F., van Helden, J., Wodak, S.: Path finding in metabolic networks: measuring functional distances between enzymes. In: Open Days in Biology, Computer Science and Mathematics JOBIM 2004. (2004)
19. Hofestädt, R.: A Petri net application to model metabolic processes. Syst. Anal. Mod. Simul. **16** (1994) 113–122
20. Reddy, V.N., Liebmann, M.N., Mavrovouniotis, M.L.: Qualitative analysis of biochemical reaction systems. Comput. Biol. Med. **26** (1996) 9–24
21. Goss, P.J., Peccoud, J.: Quantitative modeling of stochastic systems in molecular biology using stochastic Petri nets. Proc. Natl. Acad. Sci. U. S. A. **95** (1998) 6750–6755
22. Küffner, R., Zimmer, R., Lengauer, T.: Pathway analysis in metabolic databases via differential metabolic display (DMD). Bioinformatics **16** (2000) 825–836
23. Matsuno, H., Doi, A., Nagasaki, M., Miyano, S.: Hybrid Petri net representation of gene regulatory network. In: Pacific Symposium on Biocomputing. Volume 5. (2000) 341–352
24. Oliveira, J.S., Bailey, C.G., Jones-Oliveira, J.B., Dixon, D.A., Gull, D.W., Chandler, M.L.: An algebraic-combinatorial model for the identification and mapping of biochemical pathways. Bull. Math. Biol. **63** (2001) 1163–1196
25. Genrich, H., Küffner, R., Voss, K.: Executable Petri net models for the analysis of metabolic pathways. Int. J. STTT **3** (2001)

26. Oliveira, J.S., Bailey, C.G., Jones-Oliveira, J.B., Dixon, D.A., Gull, D.W., Chandler, M.L.: A computational model for the identification of biochemical pathways in the Krebs cycle. J. Computational Biology **10** (2003) 57–82
27. Zevedei-Oancea, I., Schuster, S.: Topological analysis of metabolic networks based on Petri net theory. In Silico Biology **3**(0029) (2003)
28. Regev, A., Silverman, W., Shapiro, E.: Representation and simulation of biochemical processes using the pi-calculus process algebra. In Altman, R.B., Dunker, A.K., Hunter, L., Klein, T.E., eds.: Pacific Symposium on Biocomputing. Volume 6., World Scientific Press (2001) 459–470
29. Priami, C., Regev, A., Shapiro, E., Silverman, W.: Application of a stochastic name-passing calculus to representation and simulation of molecular processes. Information Processing Letters (2001) in press.
30. Calder, M., Vyshemirsky, V., Gilbert, D., Orton, R.: Analysis of signalling pathways using the PRISM model checker. In Plotkin, G., ed.: Proceedings of the Third International Conference on Computational Methods in System Biology (CMSB 2005). (2005)
31. Regev, A., Panina, E., Silverman, W., Cardelli, L., Shaprio, E.: Bioambients: An abstraction for biological compartments (2003) to appear TCS.
32. Nielson, F., Nielson, H.R., Priami, C., Rosa, D.: Control flow analysis for bioambients. In: BioConcur. (2003)
33. Kam, N., Cohen, I., Harel, D.: The immune system as a reactive system: Modeling t cell activation with statecharts. In: Proc. Visual Languages and Formal Methods (VLFM'01). (2001) 15–22
34. Efroni, S., Harel, D., Cohen, I.: Towards rigorous comprehension of biological complexity: Modeling, execution and visualization of thymic t-cell maturation. Genome Research (2003) Special issue on Systems Biology, in press.
35. Damm, W., Harel, D.: Breathing life into message sequence charts. Formal Methods in System Design **19**(1) (2001)
36. Kam, N., Harel, D., Kugler, H., Marelly, R., Pnueli, A., Hubbard, J., Stern, M.: Formal modeling of C.elegans development: A scenario-based approach. In: First International Workshop on Computational Methods in Systems Biology. Volume 2602 of Lecture Notes in Computer Science., Springer (2003) 4–20
37. Prez-Jimnez, M., Romero-Campero, F.: Modelling EGFR signalling cascade using continuous membrane systems. In Plotkin, G., ed.: Proceedings of the Third International Conference on Computational Methods in System Biology (CMSB 2005). (2005)
38. Fages, F., Soliman, S., Chabrier-Rivier, N.: Modelling and querying interaction networks in the biochemical abstract machine BIOCHAM. Journal of Biological Physics and Chemistry **4**(2) (2004) 64–73
39. Chabrier-Rivier, N., Chiaverini, M., Danos, V., Fages, F., Schächter, V.: Modeling and querying biomolecular interaction networks. Theoretical Computer Science **351**(1) (2004) 24–44
40. Calzone, L., Chabrier-Rivier, N., Fages, F., Gentils, L., Soliman, S.: Machine learning biomolecular interactions from temporal logic properties. In Plotkin, G., ed.: Proceedings of the Third International Conference on Computational Methods in System Biology (CMSB 2005). (2005)
41. Baral, C., Chancellor, K., Tran, N., Tran, N., Joy, A., Berens, M.: A knowledge based approach for representing and reasoning about signaling networks. Bioinformatics **20** (2004) i15–i22
42. Shankland, C., Tran, N., Baral, C., Kolch, W.: Reasoning about the ERK signal transduction pathway using BioSigNet-RR. In Plotkin, G., ed.: Proceedings of the Third International Conference on Computational Methods in System Biology (CMSB 2005). (2005)

43. Hucka, M., Finney, A., Sauro, H., Bolouri, H., Doyle, J., Kitano, H.: The ERATO systems biology workbench: Enabling interaction and exchange between software tools for computational biology. In: Proceedings of the Pacific Symposium on Biocomputing. (2002)
44. : BioSpice (2004) https://community.biospice.org/.
45. : IBM Discoverylink (2004) http://publib-b.boulder.ibm.com/Redbooks.nsf/0/3c7a635147cf20d785256a540064e287?OpenDocument&Highlight=0,DiscoveryLink.
46. : geneticXchange (2004) http://midas-10.cs.ndsu.nodak.edu/bio/ppts/chung.pdf.
47. Schmidt, A., Hall, A.: Guanine nucleotide exchange factors for rho gtpases: turning on the switch. Genes Dev. **16** (2002) 1587–15609
48. Stehr, M.O.: A rewriting semantics for algebraic nets. In Girault, C., Valk, R., eds.: Petri Nets for System Engineering – A Guide to Modelling, Verification, and Applications. Springer-Verlag (2000)
49. Schmidt, K.: LoLA: A Low Level Analyser. In Nielsen, M., Simpson, D., eds.: Application and Theory of Petri Nets, 21st International Conference (ICATPN 2000). Volume 1825 of Lecture Notes in Computer Science., Springer (2000) 465–474
50. : LoLA: Low Level Petri net Analyzer (2004) http://www.informatik.hu-berlin.de/~kschmidt/lola.html.
51. Schmidt, K.: Stubborn sets for standard properties. In: International Conference on Application and Theory of Petri nets. Volume 1639 of Lecture Notes in Computer Science., Springer (1999) 46–65
52. Porter, D.: An Interpreter for JLambda (2004) http://mcs.une.edu.au/~iop/Data/Papers/.