Abstraction and Training of Stochastic Graph Transformation Systems

Mayur Bapodra and Reiko Heckel

Department of Computer Science, University of Leicester, UK
{mb294,reiko}@mcs.le.ac.uk

Abstract. Simulation of stochastic graph transformation systems (SGTS) allows us to analyse the model's behaviour. However, complexity of models limits our capability for analysis. In this paper, we aim to simplify models by abstraction while preserving relevant trends in their global behaviour. Based on a hierarchical graph model inspired by membrane systems, structural abstraction is achieved by "zooming out" of membranes, hiding their internal state. We use Bayesian networks representing dependencies on stochastic (input) parameters, as well as causal relationships between rules, for parameter learning and inference. We demonstrate and evaluate this process via two case studies, immunological response to a viral attack and reconfiguration in P2P networks.

Keywords: stochastic graph transformation, abstraction, Bayesian networks, membrane systems.

1 Introduction

Graph transformation systems (GTS) are a rule-based approach to modelling processes of structural change. Rules capture *local* behaviour in terms of preconditions and effects of atomic operations. In stochastic graph transformation systems (SGTS), each rule is assigned a probability distribution dictating the delay in its application, once enabled [10]. Such stochastic models allow us to observe emergent, *global* behaviour through simulation. For example, in a model of a peer-to-peer (P2P) network, we may specify operations of peers joining and leaving the network, making connections, etc. while being interested in a global property such as the probability of the overall network to be connected. Similarly for a study of viral attack and immunological response on cell tissue, the probability of tissue recovery or death will be of interest. This requires a detailed model of reactions such as virus multiplication, immune reaction, cell death and regeneration.

In any realistic scenario, such models will be large and complex. Detailed state representations lead to large state spaces with a high rate of low-level change and a large set of rules, creating scalability issues for analysis. Raising the level of abstraction, a model can be reduced to improve scalability, but for the price of potentially distorting analysis results. Related to the level of representation is the choice of delay distributions for rules. For example in an abstract version of

V. Cortellessa and D. Varró (Eds.): FASE 2013, LNCS 7793, pp. 312–326, 2013.

the immunological response model, a rule for a cell to be damaged beyond repair specifies an operation that takes several steps in the concrete version. The delay of the abstract rule should therefore correspond to the combined delays of the steps required at the concrete level.

In this paper, we address the interconnected problems of structural abstraction and the choice of delay distributions and their parameters. Structural abstraction is based on a hierarchical graph model inspired by membrane systems [20], where details of the lower level of the hierarchy can be hidden. As a result, the model becomes smaller in terms of the number of rules, the number of graph elements in each graph, and therefore the number of matches per rule. This increases the scalability of stochastic simulation, i.e., larger populations can be simulated over longer periods of simulated time.

The abstraction problem arises when comparing model with reality as well as between models at different levels of detail. We focus on the latter, which is easier to experiment with using stochastic simulation. Referring to the concrete and abstract models as $SGTS_1$ and $SGTS_2$, resp., the approach is based on three key ideas: Structural abstraction at the type level induces instance level projections of graphs and rules allowing us to relate concrete and abstract states and operations. Dynamic, quantitative analysis of conflicts and dependencies between operations allows us to discover cases where rules in $SGTS_2$ fail to reproduce the causal relationships between rules $SGTS_1$, or vice versa, potentially leading to the need to refine the abstract model. A Bayesian Network (BN) [21], constructed as a result of the dependency analysis of $SGTS_2$, allows us to infer stochastic parameters by training. In particular, training aims to match the throughputs (number of applications / time) of the rules of the two models. Our hypothesis is that by matching behaviour at this local level, we preserve the trends (if not absolute values) in the global properties of the models.

This leads to the following process, with steps being iterated as required.

1. Derive $SGTS_2$ as projection of $SGTS_1$ to a sub-type graph TG_2 of TG_1.
2. Simulating $SGTS_1$, perform dynamic dependency analysis over $SGTS_2$.
3. Define a Bayesian network representing the dependencies of $SGTS_2$.
4. Use parameter sweep of simulations of $SGTS_2$ to create training data for the BN. This process is known as *learning* in a BN.
5. Enter throughput data of $SGTS_1$ as evidence into the network and infer the stochastic parameters of $SGTS_2$ needed to replicate $SGTS_1$'s throughputs. This process is known as *inference* in a BN.
6. Test the parameters by running stochastic simulations of $SGTS_2$.

We demonstrate and evaluate this process via two case studies based on models of a P2P network and immunological response introduced in Sect. 3, full details of which are given in [4] and [3] respectively. Background on SGTS and BN is given in Sects. 2 and 4, resp. Sect. 4 also introduces the derivation of a BN from a SGTS and describes stochastic parameter training. Sect. 5 evaluates the results of the case studies, and Sect. 7 concludes the paper.

2 Stochastic Graph Transformation

A *typed graph transformation system* $\mathcal{G} = (TG, P, \pi)$ consists of a *type graph* TG, defining node/edge types and attributes, a set of *rule names* P and a function π defining for each name $p \in P$ a *rule* $\pi(p) = L \to R$ consisting of TG-typed graphs L, R whose intersection $L \cap R$ is called the interface of the rule. The left-hand side L represents the precondition and the right-hand side R the postcondition of the rule, whose applications transform *instance graphs*, also typed over TG. Rules can be equipped with negative application conditions (NACs) specifying forbidden context. Formally, the *application* $G \overset{p,m}{\Longrightarrow} H$ of rule p at a *match* $m : L \to G$ subject to NACs, is defined by the single-pushout (SPO) approach [16].

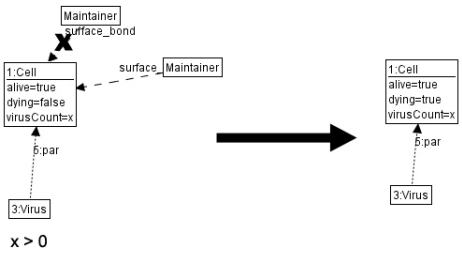

Fig. 1. Rule R2 for viral attack on last maintainer of a living cell (concrete model)

An example rule is shown in Fig. 1. It models the reaction in which a virus destroys the last maintainer in a cell (as indicated by the NAC), changing its status to dying. Fig. 2 shows the type graph for the viral attack model, defining a hierarchy of tissues, cells and organelles. Rules like those of Fig. 1 are defined over this type graph.

A *stochastic graph transformation system* $\mathcal{SG} = (TG, P, \pi, F)$ consists of a graph transformation system $\mathcal{G} = (TG, P, \pi)$ and a function $F : P \to [0,1]^{\mathbb{R}_+}$ associating to each rule name in P a probability distribution function $F(p) : \mathbb{R}_+ \to [0,1]$. Generalised SGTS, based on semi-Markov processes, allow the specification of arbitrary distributions of delays as opposed to just exponential distributions [14]. Exponential distributions model processes which depend on occurrences of random events, such as the collision between virus and *immuno* in a cell. They are characterised by a rate, i.e., the inverse of the average delay between two such events, if enabled. For operations with defined start and end points, such as the death of a cell once all maintainers are removed by viruses, normal (or lognormal) distributions are appropriate, given by mean and variance. Once started, there is an expected waiting time for the process to finish.

Given a start graph G_0, the behaviour of $\mathcal{SG}$ can be explored by simulation. For this purpose, the simulation tool GraSS [24] has been developed. The simulation works as follows. For a graph G, *events* $E(G)$ are pairs (p, m) of a rule p and an enabling match m. States (G, t, w) of the simulation are given by the current graph G, the simulation time t, and the *schedule* $w : E(G) \to \mathbb{R}_+$ mapping events

to their scheduled times. Initially, the current graph is the start graph $G = G_0$, the time is set to $t = 0$ and the scheduled time $w(p, m) = RN_{F(p)}$ for each enabled event (p, m) is selected randomly based on p's probability distribution. Then, for each simulation step

1. the first event $e = (p, m)$ is identified and rule p applied at match m to the current graph G producing the new current graph H via $G \overset{p,m}{\Longrightarrow} H$.
2. the simulation time is advanced to $t = w(e)$
3. the new schedule w', based on an updated set of enabled events $E(H)$, is defined by removing from the schedule all events in $E(G) \setminus E(H)$ and adding new events $(p', m') \in E(H) \setminus E(G)$ with time $w'(p', m') = RN_{F(p')}$ selected randomly based on p''s distribution.

The result is a *(simulation) run* $s = (G_0, t_0) \overset{p_1,m_1}{\Longrightarrow} \cdots \overset{p_n,m_n}{\Longrightarrow} (G_n, t_n)$, i.e., a transformation sequence where graphs labelled by time stamps $t_0, \ldots, t_n \in \mathbb{R}_+$ with $t_i < t_{i+1}$ for all $i \in \{0, \ldots, n-1\}$.

Graph transformation rules describe immunological response at a cellular level, as well as auxiliary operations such as virus replication, cell death and regeneration. Stochastic simulation allows us to determine the probability of tissue death once invaded by a set number of viruses. The average time taken for a tissue to come to either eliminate the viruses or suffering death can also be extracted. These are examples of global properties, as opposed to more local ones such as the throughput, i.e., the number of applications over time, of particular rules of the system.

3 Structural Abstraction

Since stochastic simulation is resource intensive, we aim to simplify hierarchical models, such as the one discussed above, by hiding details at the lowest level of the hierarchy. Formally, an abstraction relation $f : \mathcal{SG}_1 \to \mathcal{SG}_2$ between the concrete SGTS $\mathcal{SG}_1 = (TG_1, P_1, \pi_1, F_1)$ and the abstract SGTS $\mathcal{SG}_2 = (TG_2, P_2, \pi_2, F_2)$ is given by an inclusion of the type graphs $TG_2 \subseteq TG_1$ and a surjective mapping $f : P_1 \to P_2$ of rule names that is compatible with the projection of rules of $\mathcal{SG}_1$ to TG_2.

Fig. 2 shows the abstract type graph, forgetting about the contents of cells but retaining viruses, which can also exist in the tissue between cells. Attributes *alive, dying, virusCount* provide a boolean or numerical representation of information that is held in graphical form in the concrete model, but lost in the abstract one. These *aggregating attributes* are redundant in the concrete model, as expressed by invariants such as *The number of Virus nodes connected to a Cell node by par edges equals the value of the virusCount attribute of that Cell node.* Such invariants can be expressed in OCL or using a graphical constraint and either tested during the simulation or verified statically.

For an instance graph G of TG_1 such as in the top right of Fig. 2, $G|_{TG_2}$ denotes its projection to the abstract type graph via a pullback, as shown in the

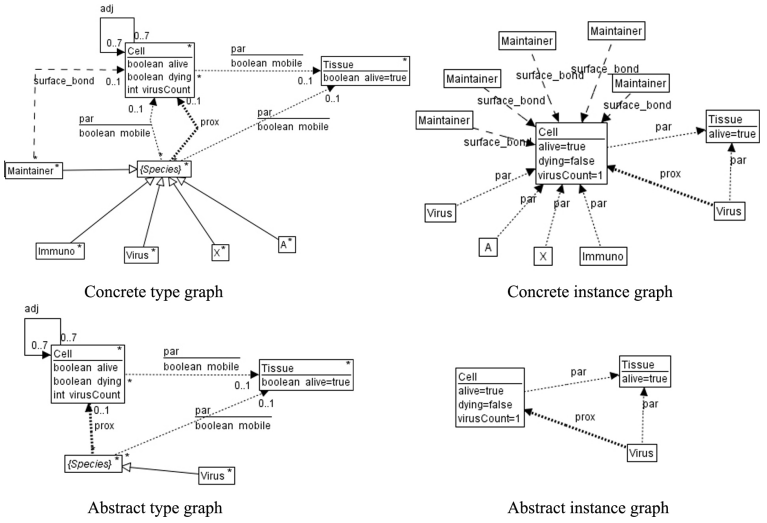

Fig. 2. Type and instance graphs for immunological response

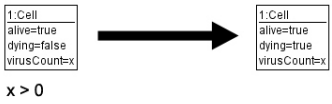

Fig. 3. Rule A_IR2 for viral attack on last maintainer of a living cell (abstract model)

bottom right. The contents of cells are hidden in the abstraction. In a similar way, rules in the concrete model are projected to the abstract type graph to create the set of abstract rules. The result is shown in Fig. 3 for the rule in Fig. 1, where the aggregating attributes retain aspects of the structural details (e.g., dying). Formally, for all $p \in P_1$, $\pi_2(f(p)) = \pi_1(p)|_{TG}$, i.e., the rule associated to concrete p equals that of abstract $f(p)$ upon projection to the abstract type graph [11]. If a concrete rule does not have any effect on cells or aggregating attributes (i.e., elements typed in TG_2), but is only concerned with manipulating elements lost in the abstraction (i.e., those typed in $TG_1 \setminus TG_2$), the result of the projection is a rule with no effect at all. We call such rules *idle*. Applications of idle rules as part of a sequence can be skipped because they do not change the graph. Thus, the abstract system may have less (non-idle) rules and abstract sequences may be shorter than their concrete counterparts. While idle rules can be disregarded, NACs in non-idle rules either have to be preserved entirely (no negative elements are lost under projection) or completely removed (at least one negative element is lost) by the projection [2].

Figure 3 shows the result of projecting the rule depicted in Figure 1, the virus destroying the last maintainer of a *Cell*, onto the abstract type graph. In this case, the rule is not idle and the single NAC is lost entirely. Abstraction leads to a smaller set of rules, and a smaller number of graph elements for each rule

application and graph, making simulation less resource-intensive (see Section 5). The full set of concrete and abstract rules can be found at [3].

Given a transformation sequence $s_1 = (G_0 \overset{p_1,m_1}{\Longrightarrow} \cdots \overset{p_n,m_n}{\Longrightarrow} G_n)$ in $\mathcal{SG}_1$, there exists a corresponding sequence $f * (s_1) = (G_0|_{TG_2} \overset{f(p_1),m_1|_{TG_2}}{\Longrightarrow} \cdots \overset{f(p_n),m_n|_{TG_2}}{\Longrightarrow} G_n|_{TG_2})$ in $\mathcal{SG}_2$. That means, under the assumptions above, mapping $f* : \mathcal{G}_1^* \to \mathcal{G}_2^*$ provides us with an abstraction not just of states and operations, but also of transformation sequences: Behaviour is preserved under abstraction [11].

We continue deriving the abstract model by defining its rules' probability distributions. After determining the type of distribution to be used, we create a Baysian network based on the causal relationships between rules, from which the distributions are inferred.

4 From SGTS to Bayesian Networks

In general, abstract rules may follow different types of distributions than their concrete counterparts. In particular, a rule representing an entire sequence of concrete steps could be normally distributed even if each concrete rule has an exponential distribution. Simulating the concrete model, we can detect matches for abstract rules and so measure the delays between the enabling and application of the rule. Plotting this data allows us to decide the shape of the distribution.

Having determined the types of the distributions, we use Bayesian networks to derive their parameters. The structure of the network is defined by the conflicts and dependencies of abstract rules. We say that rule p *enables* p' if there exists a sequence $G \overset{p,m}{\Longrightarrow} X \overset{p',m'}{\Longrightarrow} H$ such that the match m' for p' in X cannot be extended to G, i.e., the second step is dependent on the first. This is the case if p creates elements required for p's application, or deletes elements that violate a negative application condition of p'. Dually, given $H \overset{p,m}{\Longleftarrow} G \overset{p',m'}{\Longrightarrow} H'$, rule p *disables* p' if the match m' for p' is not preserved in H, i.e., the application of p is in conflict with the subsequent application of p'. This happens if p deletes elements needed for p's application or creates elements violating p's NAC.

While conflicts and dependencies can be analysed statically, like with many static methods this often results in an over approximation of the actual dependencies and conflicts occurring in simulation runs. A dynamic approach can take into account reachability, i.e., only report cases where the steps concerned are reachable from the start state. It can also provide statistics on *how many* matches for p' are generated or destroyed on average per application of p, thus allowing us to judge the significance of a dependency or conflict.

Since we are not able to simulate the abstract system before defining its stochastic parameters, we execute the concrete model and use our abstraction function $f : \mathcal{SG}_1^* \to \mathcal{SG}_2^*$ to derive abstract runs to record what we call *diagonal conflicts and dependencies* between concrete and abstract rules: According to our notion of abstraction, abstract rules $f(p)$ arise from concrete ones p that are not idle under projection to the abstract type graph TG_2. At the same time, each abstract rule, being typed over $TG_2 \subseteq TG_1$, is also implicitly typed over

TG_1. Therefore, it is possible to check for conflicts and dependencies between concrete and abstract steps while simulating the concrete system. Since overlaps of concrete and abstract rules can only be based on elements that are preserved in the abstraction, a dependency between a concrete rule p non-idle under projection and an abstract rule $f(p')$ is also a dependency between the abstract counterpart $f(p)$ and $f(p')$, and vice versa. The same is true for conflict. That means, diagonal conflicts and dependencies can be used in place of abstract ones.

The result of the analysis is recorded in the *dynamic incidence matrix* $DIM : P \times P_2 \to \mathbb{R}$ where $P \subseteq P_1$ is the subset of concrete rules with non-idle projections. For each rule $p \in P$ the matrix describes the average change in the number of matches for rules in P_1 caused by p's application. The runtime conflicts and dependencies recorded here provide the information needed to derive the structure of the BN.

A BN is an acyclic graph $G_B = (V, E, src, tg, \lambda)$. V is a set of vertices such that each $v \in V$ represents a variable (discrete or continuous). E is a set of directed edges such that each $e \in E$ has a source and target vertex $src(e)$ and $tg(e)$ in V, respectively. The function $\lambda : V \to [0,1]^{\mathbb{R}+}$ assigns to each vertex a probability distribution such that for all $v \in V$, $\lambda(v) = Q(v|V_{in}(v))$ where $V_{in}(v)$ is the set of nodes with edges towards v. Q represents a probability distribution for the value of v given the values of all variables on which v is conditionally dependent. The foundations of BNs in Bayes' Theorem are discussed in [12].

We use the BN's vertices to represent parameters of abstract rules' probability distributions as well as average rule throughputs. For exponential distributions, this parameter is the rate. For lognormal distributions, we represent the mean only. The variance is inherited from corresponding rules in the concrete model and confirmed via plotting the distributions of delays for abstract rules measured in concrete simulations. We therefore have a BN consisting of two distinct sets of nodes: input nodes representing stochastic parameters and output nodes representing the average throughput for each rule. For each rule an edge connects its parameter and throughput node, modelling the dependency of the throughput on the rule's distribution. In addition, conflicts and dependencies are represented as edges between throughput nodes. Because of the way data is propagated through the network by the training algorithm, the direction of edges between throughput nodes is irrelevant. We can therefore avoid cyclic networks, even if dependency and conflict relations are cyclic, by choosing a total order on rule names to direct edges in the network from "smaller" to "larger" names.

Definition 1 (BN of SGTS). *Assume $\mathcal{SG} = (TG, P, \pi, F)$ with arbitrary total order $<$ on P. The BN representing $\mathcal{SG}$ is a graph $B = (V, E, src, tg, \lambda)$ with functions*

- *$f_{in} : P \to V$ assigning to each rule a vertex representing its stochastic parameter (rate or mean for exponentially or lognormal distributions, respectively);*
- *$f_{out} : P \to V$ assigning to each rule a vertex representing its throughput;*

such that

- *for every $p \in P$, there is exactly one edge in B from $f_{out}(p)$ to $f_{in}(p)$;*

– *for all* $p_1, p_2 \in P$ *such that* p_1 *disables or enables* p_2*, there is exactly one edge in* B *between their respective output nodes, from* $f_{out}(p_1)$ *to* $f_{out}(p_2)$ *if* $p_1 < p_2$*, or from* $f_{out}(p_2)$ *to* $f_{out}(p_1)$ *if* $p_2 < p_1$*.*

To generate the *DIM*, while running the stochastic simulation of the concrete model we trace matches for abstract rules, their creation and destruction by concrete rule applications. The matrix records the *average* change in the number of matches over the entire simulation time. The results are given in Table 1, with a negative number indicating a net conflict between two rules (more matches being destroyed than created), and a positive entry a net dependency. The resulting BN is shown in Fig. 4.

Table 1. Dynamic incidence matrix for abstract rules over concrete model (aggregate of diagonal conflicts and dependencies)

	Applied Concrete Rule								
Abstract rule	**VC**	**IR**	**VA2**	**CD**	**TD**	**CR**	**VT**	**VM**	**SM**
A_VC	0	-0.408	-1.0	0	0	0	0.44	0.943	0
A_IR	0	-0.408	-1.0	0	0	0	0.44	0.943	0
A_VA2	0	-0.408	-1.0	0	0	0	0.44	0.943	0
A_CD	0	0	1.0	-1.0	0	0	0	0	0
A_TD	0	0	0	0.0347	-1.0	0	0	0	0
A_CR	0	0	0	0.544	0	-1.45	0	0	0
A_VT	0	0	3.8	-1.83	0	0.0744	-0.642	0	0
A_VM	0	0	-0.0548	0	0	0.0208	0	-1.0	0.00755
A_SM	0	0	0	5.82	-78.8	0	0	-4.70	-0.00708

While the structure of the net is fixed by the dependencies and conflicts, the net itself is not complete without a probability distribution λ at each node. Tools, such as Bayes Server [1], are able to learn these distributions if sufficient training data is available for every variable. The data must be varied enough so that the network can learn the effect of changes in values of one variable on another. Each row of training data includes stochastic parameters and resulting throughputs for all abstract rules, as measured by a batch of simulations.

The purpose of training is to match the throughputs measured for the abstract model with those of the concrete one. Once the probability distributions are learnt for every abstract variable in the network, we enter the known throughputs from the concrete model as evidence, i.e., fixing the values of the nodes representing outputs. The probability distributions at the remaining unknown variables, the stochastic parameters, will be perturbed as a result of this evidence, giving us their most likely values given the required throughput. This process is iterated, starting with a broad sweep of abstract stochastic parameters and refining their choices successively to sample the vicinity of parameters derived in the previous iteration. Thus, subsequent rounds of learning more and more precisely replicate the throughputs of the concrete model by the abstract one.

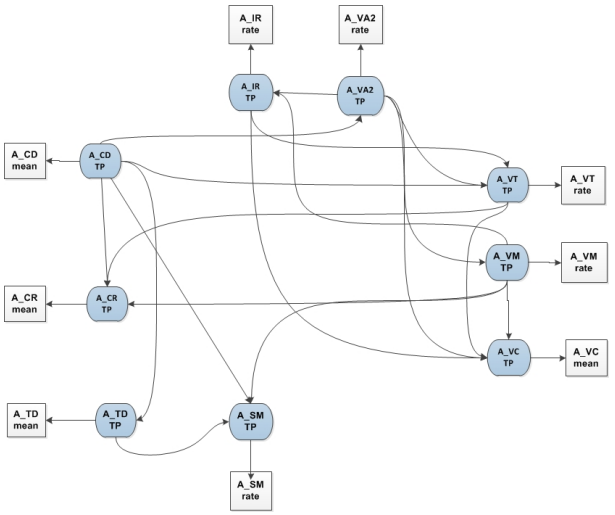

Fig. 4. BN representing dependencies and conflicts in the abstract model

In order to decide when to stop the process, we have to assess how closely concrete and abstract throughputs are matched with a given set of parameters. This is done by simulating the abstract model, measuring its throughputs, and calculating the distance between these and the concrete model's by the formula below as the product of the differences between average throughputs for each rule. If $\overline{y_c(p)}$ and $\overline{y_a(f(p))}$ are the average throughputs for non-idle concrete rule p and its abstract counterpart, respectively:

$$M = \prod_{p \in P} \frac{\overline{y_c(p)} - \overline{y_a(f(p))}}{\overline{y_c(p)}}$$

A value closer to zero indicates a better match between concrete and abstract throughputs. However, M also gives an intuitive judgement of distance. For example, for a set of 9 abstract rules as with our case study, a value of M at 10^{-9} indicates that on average the throughput for each rule is within 10% of that measured in the concrete model.

5 Evaluation

The BN in Fig. 4 was implemented in Bayes Server. A sample tissue consisting of 18 cells with 5 initial viruses was created as a start graph for simulation, using 12271 different sets of stochastic parameters with 50 simulation runs of each. The global behaviour variables measured were *total percentage of tissue deaths* (as opposed to complete virus eliminations), and *average simulated time* to complete tissue death or virus elimination. Once learning and inference were completed, in each iteration the extracted stochastic parameters were used to

run 6 stochastic simulation batches on the abstract model, each of 100 runs, to measure its throughputs and determine the distance with the concrete model. The abstract immunological response model underwent 3 iterations of parameter training before termination criteria were met. The resulting average throughputs of each iteration along with the associated distances M are shown in Table 2. The minimum value of M achieved was 2.71×10^{-8} for the second iteration.

Table 2. Throughputs (TP) in concrete and abstract immunological response models

	Concrete Model	Abstract Model Iteration 1	Abstract Model Iteration 2	Abstract Model Iteration 3
A_IR TP average	1.041	0.750	0.763	0.809
A_VA2 TP average	0.333	0.406	0.371	0.381
A_VM TP average	0.333	0.415	0.407	0.408
A_VC TP average	0.996	0.748	0.720	0.776
A_VT TP average	1.066	0.798	0.771	0.819
A_SM TP average	20.443	21.743	20.313	21.005
A_CD TP average	0.333	0.406	0.371	0.381
A_CR TP average	0.125	0.101	0.097	0.105
A_TD TP average	0.012	0.017	0.015	0.015
Distance Measure, M		1.15×10^{-6}	2.71×10^{-8}	7.60×10^{-8}

The throughputs show a good congruence between concrete and abstract model. Perturbation of any of the parameters in the abstract model increases the distance measure M, indicating that training was successful in finding a minimum distance of local behaviours. With the second iteration as the final parameter set for the abstract model, the global behaviours deviate significantly. For example, the average percentage of tissue death in the abstract model (Iteration 2) was measured as 42.7%, as opposed to 30.2% in the concrete model. This is not unexpected since not all rules in the concrete model are also present as abstract rules. Hence, the distributions of abstract rules have to account for delays of rules lost in the abstraction. For example, the reaction rule in Fig. 1 depends on rules, hidden in the abstract view, of creating the *immunos* from A resources, an auxiliary species in each cell. The delay of the immunological response reaction rule has to encompass this preliminary process. At the same time, immune reaction is in conflict with, for example, the multiplication of viruses. Increasing the delay will therefore change the balance of the corresponding race condition, thus limiting the ability of delay to compensate for rules missing in the abstract model.

It should be pointed out that the model could easily be trained to replicate global rather than local behaviour, but this would limit its use significantly, making it too specific on the particular property of interest. Alternatively, to ensure a closer match of the functional behaviour, aggregating attributes could be introduced to *Cell* as counters for all of the remaining types that are hidden. However, bringing the abstract model closer to the concrete one would of course counter our original objective of simplifying the model.

While global properties are not reproduced with their absolute values, trends and dependencies are preserved. Fig. 5 shows the result of a sensitivity analysis where we alter the delay of the Cell Regeneration rule to monitor its effect on the percentage of tissue death occurring before simulation time reaches 200 arbitrary units. We alter the mean parameter of the lognormal distributions proportionately. The graph shows a similar trend in both models, with an increase in tissue death, subject to fluctuations due to variance in simulation runs, followed by a decline as cell regeneration slows down. While the maxima do not agree, they occur at approximately similar points. This shows that while global properties may not be deduced from simulations of an abstract model, we can still infer patterns and trends of global behaviour. This in itself is a useful insight into a model.

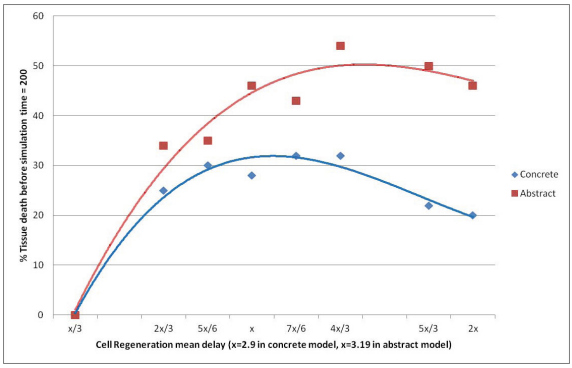

Fig. 5. Sensitivity of % Tissue Death on Cell Regeneration (concrete vs abstract)

Running a simulation of 100 runs on a 64bit Intel Core i5 2.53GHz CPU with 6GB of memory, using the 32bit Eclipse plug-in version of the software resulted in a 72% saving in runtime for the abstract model over the concrete one.

To evaluate the results on another example, a second case study was created, based on the model of a P2P VoIP network. Full details are given in [4], but the objective was for concrete and abstract models to be functionally bisimilar [13], no rules become idle upon abstraction and conflicts and dependencies are preserved and reflected. All distributions are exponential. However, the models still differ in their stochastic behaviour, due to different numbers of matches for corresponding rules at both levels caused by the projection. Parameter training took 3 iterations, the best results being produced during the second iteration at a distance M of 2.94×10^{-10}. The global behaviour matches more closely than in the immune response model: Percentage of disconnected super peer pairs is 11.4 in the concrete model vs. 12.8 in the abstract one, and that of connected clients only varies between 60.4 and 60.3.

Fig. 6 shows the dependency of the percentage of disconnected super pairs on the rate of the CreateShortcut rule, which creates redundant links to reduce the probability of loss of connectivity. Just as in the immune response case, there is

Table 3. Throughputs (TP) measured in concrete and abstract VoIP models

	Concrete Model	Abstract Model Iteration 1	Abstract Model Iteration 2	Abstract Model Iteration 3
A_NC TP average	1.230	1.080	1.197	1.198
A_LC TP average	1.250	1.154	1.251	1.228
A_NS TP average	0.902	0.897	0.821	0.874
A_PC TP average	0.016	0.020	0.030	0.024
A_TL TP average	0.729	0.763	0.796	0.782
A_TS TP average	0.909	0.907	0.841	0.888
A_TU TP average	0.472	0.285	0.358	0.379
A_CS TP average	0.623	0.963	0.780	0.876
Distance Measure, M		3.93×10^{-9}	2.94×10^{-10}	1.10×10^{-9}

a very good match of the trends in both models, even if absolute values are not exactly replicated throughout. Again, there is a significant gain in performance on abstraction, with a 66% reduction in runtime for 20 simulation runs.

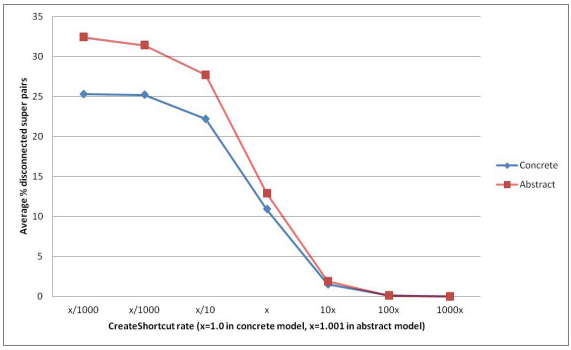

Fig. 6. Sensitivity of % Disconnected Super Pairs on CreateShortcut rate (concrete vs abstract)

Summarising, we were able to show that

- abstraction by projection, with aggregating attributes replacing some of the graphical structure lost, provides a simple and effective way of reducing the complexity of the model, increasing scalability;
- our approach to parameter training using Bayesian networks, defined on the basis of dynamic dependency and conflict analysis, allows us to find a good match of the local behaviours of concrete and abstract models as given by the throughputs of corresponding rules;
- absolute values of global properties are only replicated closely where the models are functionally (but not stochastically) bisimilar;
- trends in global properties, as expressed by their sensitivity with respect to the parameters of essential rules, are reproduced faithfully even if models are not bisimilar;

Before turning to related work, let us discuss possible threats to the validity of the evaluation. During training, variances of lognormal distributions for abstract rules are inherited from the corresponding concrete rules. For the Bayesian network, these variances are therefore not variables, but constants. Mean and rate parameters are inferred so as to replicate the concrete model's throughputs as closely as possible, given these variances. Fixing the variances obviously limits the flexibility of training and thus leads to larger deviations of abstract from concrete throughputs. However, while being one reason for not matching throughputs perfectly, this limitation of training does not affect the validity of the experiments, which are aimed at showing the match of global properties or trends.

The quality of the fit between concrete and abstract throughputs also depends on the distance measure used to control the iteration of training and simulation. The measure M introduced in Sect. 4 has been chosen for its ability to cope with smaller datasets, and its linear variation, over more elaborate notions such as Student's t-test and Mahalanobis distance [8]. Again, this choice could affect the quality of the match between concrete and abstract throughputs, but not the experiments themselves.

Obvious threats to the validity of the experiments are the selection of the experiments themselves as well as the limited number of simulation runs. As to the former, we have chosen case studies from two different domains, focussing on different global properties (probability of global outcomes vs. number of occurrences of certain patterns) and considering different degrees of abstraction. A larger number of simulation runs, especially for the sensitivity analysis, was beyond the available resources.

6 Related Work

Work on abstraction in graph transformation has followed a variety of motivations. In [19], it is a means to improve comprehensibility of complex GTS by hiding and retrieving substructures as required. To enable analysis of models, many approaches aim at reducing the state space or behaviour representation. [5] uses *neighbourhood abstraction* to group graph elements via an equivalence relation up to some radius defining a node's neighbourhood. This allows the level of precision to be adjusted if the current abstraction does not allow the verification of properties. [25] uses a similar approach, but abstracted nodes are characterised by satisfaction of temporal logic formulae representing some behavioural property of the concrete system. In [22], based on shape graphs introduced in [23], nodes are grouped by structural similarity with multiplicities to capture concrete representations of an abstract *shape*. Several states are therefore combined into a single structure. In counterexample-guided abstraction refinement based on unfoldings [15], the behaviour of a GTS is represented by a Petri graph representing an approximated unfolding. In all approaches, abstraction works at the level of the state space or unfolding, or requires a different notion of GTS at the abstract level. Our approach is based on abstraction of standard typed GTS. We simplify type graph, start graph and rules, but the graph transformation approach is unchanged. Analogous to the use of aggregating attributes to increase

precision, many approaches provide means to fine-tune the level of abstraction to the properties of interest. In our case these are based on preserving or reflecting essential conflicts and dependencies, rather than reachability.

While this work focuses on a non-deterministic, discrete, stochastic approach to modelling reactions, [7] is a similarly rule-based technique that aims to formulate a reduced, *deterministic* system of differential equations (ODEs) for a reaction system. This is achieved by framing the dynamics in terms of functionally distinct patterns known as fragments (rather than traditionally disjoint species), followed by methods derived from abstract interpretation to further reduce the number of ODEs. Also from a continuous, deterministic paradigm, [6] derives a *refinement* from a more abstract representation of a reaction system, both by replacing reactants with subtypes, or by adding possible reactivity. The resulting system preserves numerical properties (analogous to global trends in our work) without having to perform expensive model-fitting for each subsequent refinement.

With an aim related to our training of stochastic parameters, [17] presents an algorithm known as PEGG (Parameter Estimation for Graph Grammars) which can extract parameters for rules in context-free graph grammars from sequences of graphs resulting from the application of rules. This is useful for modelling based on observations of a system that is executable but with unknown parameters, where parameter estimation aims to determine these parameters. While conceptually close, the limitation to context-free graph grammars means that the approach is not applicable to general graph transformation systems.

7 Conclusion

We have demonstrated a methodology for abstraction of GTS and training their stochastic parameters. Evaluating the approach in two case studies we found that, while absolute values of global properties are not always preserved, the abstract model replicates faithfully trends and dependencies of the concrete one. In future work we plan to explore the relation between the number of matches of concrete and abstract rules and their stochastic parameters as well as the possibility of scaling systems by enlarging or reducing their start graphs.

References

1. Bayes server, intelligent systems specialists, `http://www.bayesserver.com/` (retrieved December 9, 2012)
2. Arijo, N., Heckel, R.: View-based modelling and state-space generation for graph transformation systems. ECEASST 47 (2012)
3. Bapodra, M., Heckel, R.: Case study - hypothetical immunological response rules (December 2012), `http://www.cs.le.ac.uk/people/mb294/docs/CaseStudyRules_v3.pdf`
4. Bapodra, M., Heckel, R.: Case study - VoIP rules (October 2012), `http://www.cs.le.ac.uk/people/mb294/docs/VoIPCaseStudyRules.pdf`
5. Bauer, J., Boneva, I., Kurbán, M., Rensink, A.: A modal-logic based graph abstraction. Graph Transformations, 321–335 (2008)

6. Iancu, B., Czeizler, E., Czeizler, E., Petre, I.: Quantitative refinement of reaction models. To appear: International Journal of Unconventional Computing
7. Danos, V., Feret, J., Fontana, W., Harmer, R., Krivine, J.: Abstracting the differential semantics of rule-based models: exact and automated model reduction. In: 2010 25th Annual IEEE Symposium on Logic in Computer Science (LICS), pp. 362–381. IEEE (2010)
8. De Maesschalck, R., Jouan-Rimbaud, D., Massart, D.: The Mahalanobis distance. Chemometrics and Intelligent Laboratory Systems 50(1), 1–18 (2000)
9. Drewes, F., Hoffmann, B., Plump, D.: Hierarchical graph transformation. In: Foundations of Software Science and Computation Structures, pp. 98–113. Springer (2000)
10. Heckel, R., Lajios, G., Menge, S.: Stochastic graph transformation systems. Fundamenta Informaticae 74(1), 63–84 (2006)
11. Heckel, R., Corradini, A., Ehrig, H., Löwe, M.: Horizontal and vertical structuring of typed graph transformation systems. Mathematical Structures in Computer Science 6(6), 613–648 (1996)
12. Heckerman, D., et al.: A tutorial on learning with Bayesian networks. Nato Asi Series D Behavioural And Social Sciences 89, 301–354 (1998)
13. Joyal, A., Nielson, M., Winskel, G.: Bisimulation and open maps. In: Proceedings of Eighth Annual IEEE Symposium on Logic in Computer Science, LICS 1993, pp. 418–427. IEEE (1996)
14. Khan, A., Torrini, P., Heckel, R.: Model-based simulation of VoIP network reconfigurations using graph transformation systems. Electronic Communications of the EASST 16 (2009)
15. König, B., Kozioura, V.: Counterexample-Guided Abstraction Refinement for the Analysis of Graph Transformation Systems. In: Hermanns, H., Palsberg, J. (eds.) TACAS 2006. LNCS, vol. 3920, pp. 197–211. Springer, Heidelberg (2006)
16. Löwe, M.: Algebraic approach to single-pushout graph transformation. Theor. Comput. Sci. 109, 181–224 (1993)
17. Oates, T., Doshi, S., Huang, F.: Estimating Maximum Likelihood Parameters for Stochastic Context-Free Graph Grammars. In: Horváth, T., Yamamoto, A. (eds.) ILP 2003. LNCS (LNAI), vol. 2835, pp. 281–298. Springer, Heidelberg (2003)
18. Palacz, W.: Algebraic hierarchical graph transformation. Journal of Computer and System Sciences 68(3), 497–520 (2004)
19. Parisi-Presicce, F., Piersanti, G.: Multilevel graph grammars. In: Graph-Theoretic Concepts in Computer Science, pp. 51–64. Springer (1995)
20. Păun, G.: Introduction to membrane computing. Applications of Membrane Computing, 1–42 (2006)
21. Pourret, O., Nam, P., Naïm, P., Marcot, B., et al.: Bayesian networks: a practical guide to applications, vol. 73. Wiley (2008)
22. Rensink, A., Distefano, D.: Abstract graph transformation. Electronic Notes in Theoretical Computer Science 157(1), 39–59 (2006)
23. Sagiv, M., Reps, T., Wilhelm, R.: Solving shape-analysis problems in languages with destructive updating. ACM Transactions on Programming Languages and Systems (TOPLAS) 20(1), 1–50 (1998)
24. Torrini, P., Heckel, R., Ráth, I.: Stochastic simulation of graph transformation systems. Fundamental Approaches to Software Engineering, 154–157 (2010)
25. Yamamoto, M., Tanabe, Y., Takahashi, K., Hagiya, M.: Abstraction of Graph Transformation Systems by Temporal Logic and Its Verification. In: Meyer, B., Woodcock, J. (eds.) VSTTE 2005. LNCS, vol. 4171, pp. 518–527. Springer, Heidelberg (2008)