Fundamenta Informaticae 72 (2006) 1–22
IOS Press

Stochastic Graph Transformation Systems

Reiko Heckel[*†]
Department of Computer Science
University of Leicester, United Kingdom
reiko@mcs.le.ac.uk

Georgios Lajios[‡] **and Sebastian Menge**
Software-Technology
University of Dortmund, Germany

Abstract. In distributed and mobile systems with volatile bandwidth and fragile connectivity, non-functional aspects like performance and reliability become more and more important. To formalize, measure, and predict such properties, *stochastic methods* are required. At the same time such systems are characterized by a high degree of architectural reconfiguration. Viewing the architecture of a distributed system as a graph, this is naturally modeled by *graph transformations*.

To combine these two views, in this paper we introduce stochastic graph transformation systems associating with each rule its application rate—the rate of the exponential distribution governing the delay of its application.

Beside the basic definitions and a motivating example, we derive continuous time Markov chains, establish a link with continuous stochastic logic, deal with the problem of abstraction through graph isomorphisms, and discuss the analysis of properties by means of an experimental tool chain.

1. Introduction

Non-functional requirements, concerning the quality or resources of a system, are often difficult to capture, measure, and predict. At the same time they are usually critical for success. Many

[*]European Community's Human Potential Programme under contract HPRN-CT-2002-00275, [SegraVis]
[†]Address for correspondence: Department of Computer Science, University of Leicester, United Kingdom
[‡]Research funded in part by Deutsche Forschungsgemeinschaft, grant DO 263/8-1 [Algebraische Eigenschaften stochastischer Relationen]

failures of software engineering projects have been attributed to a lack of understanding of non-functional aspects in the early stages of development [17].
With the success of Internet and mobile technology, properties like the reliability of connections, available bandwidth and computing resources become an even greater concern. Since individual occurrences of failures are generally unpredictable, stochastic concepts are required to formalize such properties. Many specification formalisms provide corresponding extensions, including stochastic transition systems (or Markov chains [2, 30]), stochastic Petri nets [1, 5, 27, 29] or process algebras [6, 10]. Most of these formalisms specialize on describing behavior in terms of orderings of events, neglecting aspects like data transformations and changes to software architecture or network topology.
A noticeable exception is the π-calculus [26] which allows communication of channel names between interacting processes. It is thus possible to describe changes of data structures or network topologies. The stochastic π-calculus [32], extending the original by assigning rates to the communication actions of a process, allows to address non-functional aspects. However, while the π-calculus is an adequate semantic framework for programming, it is too low-level for expressing requirements in the early stages of a project. Here communication between developers and clients requires a direct, diagrammatic description of *what* changes are required, instead of a detailed description of *how* they are achieved.
A more visual and more abstract formalism of similar expressiveness are graph transformation systems. In this approach, only pre- and post-conditions of an operation (*what* it should achieve) are specified as graphical patterns while the underlying mechanisms for achieving these changes are ignored. In order to account for the non-functional aspects, we introduced *stochastic* graph transformation systems [20]. Associating an exponentially distributed application delay with each rule, we derived continuous time Markov chains (CTMCs), the standard model for stochastic analysis. The exclusive use of exponential distributions imposes a certain restriction on the possible applications, but it opens the door to Model Checking tools. We are thus able to establish a link to continuous stochastic logic (CSL) to express and verify properties like the probability of being connected within 20 seconds after start-up, the long-term probability for connectedness, etc. A way to model more general distributions on the base of exponential distributions would be to introduce virtual states, a procedure known from Queuing Theory as the Cox method [23, 4.7]. This paper provides an elaborated account of the concepts and theory presented in [20], improved in several respects.

- Instead of plain typed graphs, the present paper uses *attributed* typed graph transformations systems [15]. Beside the obvious advantage of using abstract data types to deal with information that is not structural in nature, it also provides a way of identifying graph elements by means of name attributes across transformation sequences, as well as a notion of variables and assignments used for expressing properties of graphs in logic formulae.

  These features are not easily obtained otherwise, since graph transformation steps are only defined up to isomorphism (renaming). In [20], dedicated sets of identifiers and parameter lists with partial assignment functions were employed for this purpose.

- We deal explicitly with abstraction, i.e., conceptually and in the tools we use, concrete graphs are understood as representatives of isomorphism classes of graphs. We are able

to keep track of the identity of "interesting" nodes by means of dedicated attributes (see above).

Stochastic bisimulation is the standard equivalence relation on states of CTMCs used by for model checkers for state space reduction because bisimilar states are undistinguishable by CSL formulae. We show that certain graph isomorphisms are compatible with bisimilarity, thus justifying the use of isomorphism classes as states.

The paper is structured as follows. After discussing related work and outlining our approach, in Sect. 3 we present the basic definitions of typed attributed graph transformation systems along with a simple example of a wireless network. Sect. 4 is devoted to the definition of Markov Chains and CSL. In Sect. 5 we make use of stochastic concepts by associating rates of exponential probability distributions with the rules of a graph transformation system, describing the Markov chain generated from it, investigating the relation of bisimilarity and graph isomorphisms. Sect. 7 concludes the paper with a discussion of tools and relevant theoretical problems.

2. Related Work

The presented approach inherits from two lines of research: *stochastic modeling and analysis* and *graph transformation for mobility*. We discuss both of them in turn.

Stochastic modeling and analysis. The underlying model for stochastic analysis is provided by *Markov chains*, i.e., transition systems labeled with probability distributions on transitions [2, 30, 4]. *Stochastic Petri nets* provide a convenient method of describing Markov chains. The reachability graph of the net provides the underlying transition system, and its state transitions are decorated with the probabilities of the net transitions from which they are generated [5]. A similar idea lies behind *stochastic process algebras* where process algebras like CCS [26] or the π-calculus are used to describe the transition system [6].
Generalizing from these examples, the idea of stochastic modeling can be phrased as follows. A state-based formalism is used to specify the desired behavior. From suitable annotations in the specification, probability distributions for the transitions of the generated transition systems are derived which provide the input to stochastic analysis techniques.
Our approach follows the same strategy, replacing Petri nets or process algebra with graph transformation systems. In this way, we obtain a high-level formalism in which both functional and non-functional aspects of mobile systems can be adequately specified.

Graph transformation for mobility. Graph transformation systems have been used for describing the semantics of languages for mobility, like the Ambient calculus [16] as well as corresponding extensions of the UML [3]. However, we will be more interested in direct applications to the modeling of mobile and distributed systems.
Generally, we may distinguish approaches which take a strictly local perspective, modeling a distributed system from the point of view of a single node [22, 9], from those taking a global point of view, [41, 18]. In the latter case, each rule specifies the preconditions and effects of a potentially complex protocol with multiple participants, rather than a single operation as in the former case.

Our approach follows the second, global style of specification, with the running example derived from [18]. However, we think that the same combination of graph transformation with stochastic concepts could be applied to the more local style of specification.

Stochastic Context-Free Graph Grammars While our approach clearly targets system modelling and analysis, the use of stochastic grammars in other fields is definitely worth mentioning. In bioinformatics and speech recognition, for example, different kinds of stochastic grammars have been known for a long time [7, 33]. Recently, interest in modelling biochemical processes by graph transformation has grown [35] again. Given the structures of a set of glucans as training data, [39] shows how inferred rules of a stochastic graph grammar can be used to predict the structure of glucan given a set of unknown glucoprotein.
Oates et al. present in [31] an algorithm to estimate the maximum likelihood parameters for stochastic context-free graph grammars, adapting the Inside-Outside algorithm of [25].
Mosbah did early work on probabilistic graph grammars, using them to generate random graphs. He is able to derive the statistical behavior of the grammar and to find the "right" production probabilities so that the generated graphs fit some conditions (graph properties)[28].

3. Typed Attributed Graph Transformation

In this section, we provide the basic definitions of graph transformation systems based on the so-called algebraic or double-pushout (DPO) approach to the transformation of typed attributed graphs [13, 8, 15]. Fundamental notions of algebraic specification [12], like signature, algebra and homomorphism, term algebra, etc. are assumed.
We use directed and attributed graphs, in which both the vertices and the edges can carry attribute values of an arbitrary Σ-algebra. The elements of the algebra are a special kind of vertices in the graph and the attribution is given by special edges mapping vertices and edges to the data. First we need

Definition 3.1. (E-graph)
An *E-graph* $G = (V_1, V_2, E_i, s_i, t_i), i = 1, 2, 3$ consists of

- V_1, the set of graph nodes and V_2, the set of data nodes,
- E_1, the set of graph edges, E_2, the set of node attribute edges and E_3, the set of edge attribute edges

and source and target functions

- $s_1 : E_1 \to V_1$ and $t_1 : E_1 \to V_1$,
- $s_2 : E_2 \to V_1$ and $t_2 : E_2 \to V_2$ and
- $s_3 : E_3 \to E_1$ and $t_3 : E_3 \to V_2$.

An *E-graph morphism* $f : G_1 \to G_2$ is a tuple $(f_{V_1}, f_{V_2}, f_{E_1}, f_{E_2}, f_{E_3})$ with $f_{V_i} : G_{1,V_i} \to G_{2,V_i}$ and $f_{E_j} : G_{1,E_j} \to G_{2,E_j}$ for $i = 1, 2$ and $j = 1, 2, 3$ such that f commutes with all source and target functions.

Thus, an E-graph is the algebraic formulation of the kind of graphs described above. The edges E_1 specify the graph structure on V_1, whereas E_2 and E_3 specify the attribution of vertices in V_1 and edges in E_1. Attributed graphs then just use the elements of a Σ-algebra as V_2.

Definition 3.2. (attributed graph)
Let $\Sigma = (S_\Sigma, OP_\Sigma)$ be a data signature with attribute value sorts $S' \subseteq S_\Sigma$. An *attributed graph* consists of an E-graph together with a Σ-algebra D, such that $\dot{\bigcup}_{s \in S'} D_s = G_{V_2}$. An *attributed graph morphism* is a pair $f = (f_G, f_A)$ with an E-graph morphism f_G and a Σ-algebra homomorphism f_A such that (1) is a pullback for all $s \in S'$.

$$\begin{array}{ccc} D_{1,s} & \xrightarrow{f_{A,s}} & D_{2.s} \\ \subseteq\downarrow & (1) & \downarrow\subseteq \\ G_{1,V_2} & \xrightarrow[f_{G,V_2}]{} & G_{2,V_2} \end{array}$$

Together with the corresponding morphisms one can construct categories of both E-graphs and attributed graphs, opening the door to the theory of adhesive high level replacement (HLR). The requirement of (1) being a pullback is needed to ensure that data and graph elements, as well as data elements of different sort, are kept entirely separate, that is, f_{G,V_2} preserves and reflects the different subsets. It is also needed to show that the category of attributed graphs is adhesive HLR [15, 14].
In the sequel, we assume a distinguished sort $ID \in S'$ of *identity attributes* which we need to keep track of individual nodes.

Definition 3.3. (typed attributed graphs)
We fix the *attributed type graph* $ATG = (TG, Z)$ where Z is the final Σ-algebra. A typed attributed graph (or instance graph) (AG, t) over ATG is an attributed graph AG together with an attributed graph morphism $t : AG \to ATG$. A typed attributed graph morphism is an attributed graph morphism $f : (AG_1, t_1) \to (AG_2, t_2)$ such that $t_2 \circ f = t_1$. We will omit the typing morphism t if the situation allows doing so.

With just one element for each sort, Z provides a representation of the signature Σ as an algebra. Finality means that every Σ-algebra has a unique homomorphism into Z, by which every data element is mapped to the element of Z representing its sort.
Typed attributed graphs provide a simple data model quite similar to basic class and object diagrams of UML. We use this well-known notation in the example below.

Example 3.1. (mobile systems: types and configurations)
As a running example, a mobile system is modeled, consisting of cells and devices, as well as the spatial and connectivity relations between them.
Fig. 1 shows the attributed type graph ATG consisting of $V_1 = \{Device, Cell\}$, $V_2 = \{Bool, ID\}$ in UML notation. Note that we assume an additional attribute type ID to model identities of vertices. The edge types on V_1 are $E_1 = \{connection, located, neighbor\}$ and we have vertex attributes $E_2 = \{ok, id\}$ of type $Bool, ID$ respectively (the ID-attribute is not shown explicitly in the type graph). Edge attributes are not used in this example, so $E_3 = \emptyset$.

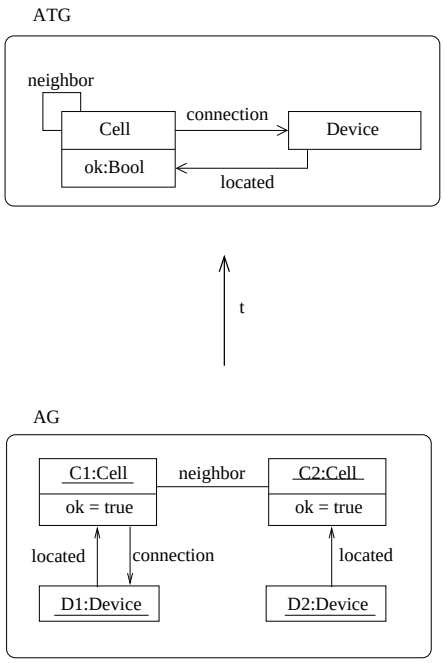


Figure 1. Type graph ATG and sample instance graph $\langle AG, t\rangle$

Using the type graph ATG we can model a nomadic wireless network like a mobile phone network or wireless LAN. Cells are linked by a geographic neighborhood relation. To communicate with mobile devices within the cell, connections are established. The base stations of a cell can be broken, indicated by the Boolean attribute ok with value $ok = false$, otherwise $ok = true$.
In the lower of Fig. 1 an ATG-typed instance graph is depicted modelling two devices located in adjacent cells. As usual, an undirected edge is used to model two directed edges in opposite directions. The left of them has established a connection. The typing morphism t should be obvious. Using UML-like notation, we write *C1:Cell* for a node of type *Cell* having $C1$ as ID-attribute. We will also use the notation *C1.ok* for the ok-attribute of cell $C1$.

After having defined the objects and morphisms of the category we are working in, we are now ready to formulate graph transformation.

Definition 3.4. (typed attributed rule spans)
Given an attributed type graph ATG, an *ATG-typed (attributed) rule span* is an injective span of ATG-typed attributed graph morphisms $L \xleftarrow{l} K \xrightarrow{r} R$ such that the data algebra is shared between the three graphs ($L_A = K_A = R_A$) and preserved by the morphisms of the span ($l_A = r_A = id_A$).
For an attribute edge type $attr$, rule span $L \xleftarrow{l} K \xrightarrow{r} R$ is called *$attr$-preserving* if the following conditions hold

- For every node $x \in K_{V_1}$ and edge $e : attr \in L_{E_2}$ with $s_2^L(e) = l(x)$, there is an edge $e' : attr \in K_{E_2}$ such that $l(e') = e$.

- For all nodes $x \in L_{V_1}, x' \in R_{V_1}$ and edges $e : attr \in L_{E_2}, e' : attr \in R_{E_2}$: $s_2^L(e) = x, s_2^R(e') = x'$ and $t_2^L(e) = t_2^R(e')$ implies the existence of $x'' \in K_{V_1}$ such that $l(x'') = x$ and $r(x'') = x'$.

The requirement to preserve the data algebra in rules and transformations is motivated by pre-defined data types, like integers or booleans, which are not changed in the course of a computation. In the context of this paper, the condition also ensures that the set of identities for vertices and edges is preserved, thus providing a constant name space for nodes and edges. Whenever a distinguished set ID of identity attributes is used, we shall implicitly assume that rule spans are $attr$-preserving for all $attr \in ID$. We always require the ID-attributes in a typed attributed graph AG to be *unique*, i.e. $t_2^{AG}(e) = t_2^{AG}(e')$ of sort ID implies $e = e'$ for all $e, e' \in E_2$. The first attribute condition assures that certain attributes are not deleted without the graph elements to which they are attached. This is particularly interesting for attributes serving as identifiers for these elements. The second condition requires that attribute values are not reused, i.e., when a vertex occurs in both L and R, it is assigned to the same attribute.

Definition 3.5. (attributed graph transformation system)
An *attributed graph transformation system* $\mathcal{G} = \langle \Sigma, ATG, P \rangle$ consists of

- a data signature $\Sigma = \langle S_\Sigma, OP_\Sigma \rangle$
- an attributed type graph ATG over Σ
- a set P of ATG-typed rule spans $p(x_1, \ldots, x_n) = (L \xleftarrow{l} K \xrightarrow{r} R)$ rule span attributed over $T_\Sigma(x_1, \ldots, x_n)$, the Σ-term algebra with variables $x_1, \ldots, x_n$.

Attributing rules with the Σ-term algebra enables us to use the variables in X as "formal parameters", to be replaced by "actual parameters" (nodes of the state graphs) as part of transition labels. This provides us with a notion of object variables that we will use in the stochastic logic introduced in section 4.
Assume an attributed graph transformation system $\mathcal{G} = \langle \Sigma, ATG, P \rangle$ and let $p(x_1, \ldots, x_n) = (L \xleftarrow{l} K \xrightarrow{r} R)$ in P. An *attributed graph transformation step* $G \overset{p(a_1,\ldots,a_n)}{\Longrightarrow} H$ is given by a *double-pushout (DPO) diagram* as shown below, where

- (1), (2) are pushouts
- the bottom is a rule span attributed over a Σ-algebra A
- $a_i = o_A(x_i)$ for all $i = 1 \ldots n$ where $o_A : T_\Sigma(x_1, \ldots, x_n) \to A$ is the Σ-homomorphism representing the shared data type mapping of o_L, o_K, o_R, i.e. $o_A := (o_L)_A = (o_K)_A = (o_R)_A$.

$$\begin{array}{ccccc} L & \xleftarrow{l} & K & \xrightarrow{r} & R \\ {\scriptstyle o_L}\downarrow & (1) & \downarrow{\scriptstyle o_K} & (2) & \downarrow{\scriptstyle o_R} \\ G & \xleftarrow[g]{} & D & \xrightarrow[h]{} & H \end{array}$$

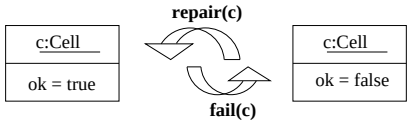


Figure 2. Failure and repair rules

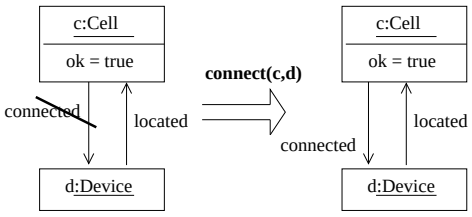


Figure 3. Establishing a connection

Thus the Σ-homomorphism o_A maps the formal parameters of the rule to concrete values in A. This can be thought of as the instantiation of a rule scheme. Homomorphism o_A is uniquely determined by $a_1, \ldots, a_n$ because the term algebra is initial, so the notation $p(a_1 \cdots a_n)$ is well-defined, meaning the use of rule p together with morphism o_A. To ease the notation we write $p(\overline{a})$ for $p(a_1 \cdots a_n)$. We call the graph transformation step $G \overset{p(a_1,\ldots,a_n)}{\Longrightarrow} H$ *attr-preserving*, if its underlying rule span is *attr*-preserving. For more details on DPO graph transformation (e.g. the existence of the pushout complement D) we refer to [36].

Example 3.2. (rules and transformations)
Graph transformation rules model the failure and recovery of components, their movement, the establishment and loss of connections, etc. Failure and repair of stations are expressed by the rules in Fig. 2.
If a device is located in a working cell and not already connected (a negative application condition, shown as a crossed out edge [19]), a connection may be established as shown in Fig. 3.
Devices may move into and out of cells, thus loosing and regaining the signal as specified in Fig. 4.
Another negative application condition ensures that there is at most one signal edge between a station and a device. Devices should move between cells covered by neighboring stations more often than moving out of range entirely, as shown in Fig. 5. When the sending station of a cell fails, its connections break down too as shown in Fig. 6.

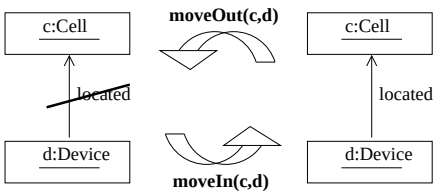


Figure 4. Movement of devices into and out of a cell

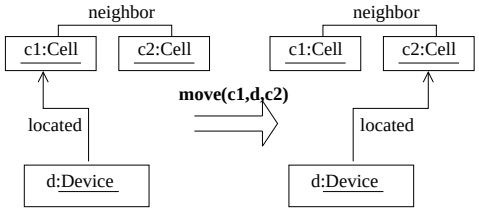


Figure 5. Movement of devices between cells

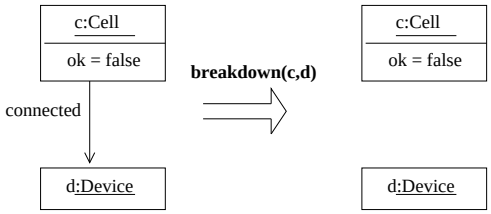


Figure 6. Breakdown of connection if cell fails

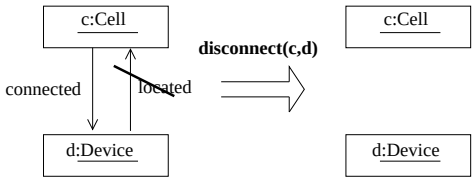


Figure 7. Establishing a connection and loosing it due to loss of a signal

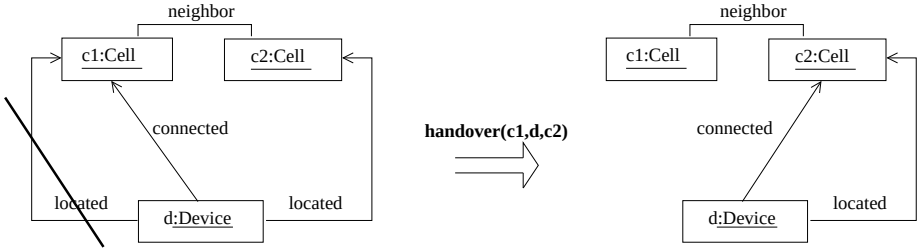


Figure 8. Handover between two stations

If a device moves out of a cell while staying connected, there are two possibilities: The connection is simply lost as well, cf. Fig. 7. Or, in order to ensure continuous connectivity, a handover is performed as shown in Fig. 8: If a device has just moved out of a cell, but is still located in another adjacent cell, the connection may be handed over to the second station.
Note the use of lowercase letters for the variables in the rules, e.g. d_1, c_2, in contrast to the use of capitals in the instance graph of figure 1.

Definition 3.6. (transformation sequence)
A transformation sequence in $\mathcal{G}$

$$G_0 \overset{p_1(\overline{a}_1)}{\Longrightarrow} G_1 \overset{p_2(\overline{a_2})}{\Longrightarrow} \cdots \overset{p_n(\overline{a_n})}{\Longrightarrow} G_n$$

is a sequence of consecutive transformation steps using the rules in P, briefly denoted by $G_0 \Longrightarrow^*_{\mathcal{G}} G_k$. We call $G_0 \Longrightarrow^*_{\mathcal{G}} G_k$ *attr-preserving*, if every span $G_i \overset{p_1(\overline{a}_1)}{\Longrightarrow} G_{i+1}$ $(i = 0, \ldots, n-1)$ is *attr-preserving*. System $\mathcal{G}$ is called *attr-preserving* with respect to an attribute type *attr* if all transformation sequences $G_0 \Longrightarrow^*_{\mathcal{G}} G_k$ are *attr*-preserving.

It is thus not possible to delete a node with *attr* value v in one transformation step and then re-introduce a new node with the same *attr* value in a later step, or to delete *attr* values before the graph elements they are attached to. We will require this property for all ID-attributes in order to identify individual nodes through transformation sequences.

Example 3.3. (transformation sequence)
Fig. 9 shows a sample transformation sequence. Device $D1$ moves into cell $C1$ and establishes a connection. In the next step, the station in $C1$ fails. Before the connection of $D1$ can breakdown, the device moves into the adjacent cell $C2$ and handover occurs. Our sample system is ID-preserving, but of course not $Bool$-preserving, as the status of a station can change from $ok = true$ to $ok = false$ and vice versa.

Now we are ready to define the labeled transition system induced by a graph transformation system with the state space built from isomorphism classes of graphs, and transitions labeled by rule names with actual parameters.

Definition 3.7. (induced labeled transition system)
Let $\mathcal{G} = \langle \Sigma, ATG, P \rangle$ be a graph transformation system and G_0 an attributed graph typed over ATG. The *labeled transition system induced by* $\mathcal{G}$ *and* G_0 is given by $LTS(\mathcal{G}, G_0) = \langle L, S, T \rangle$, where

- S is the set of all isomorphism classes of graphs reachable from G_0, i.e. $S = \{\, [G_n] \mid G_0 \Longrightarrow^*_{\mathcal{G}} G_n \}$,
- $L = \{p(a_1, \ldots, a_n) \mid p(x_1, \ldots, x_n) : (L \xleftarrow{l} K \xrightarrow{r} R) \in P \wedge a_1, \ldots, a_n \in (G_0)_A\}$ is the set of rule names, with formal parameters replaced by actual ones from the data algebra of G_0
- $T \subseteq S \times L \times S$ is the transformation relation, where $\langle G, p(a_1, \ldots, a_n), H \rangle \in T$ iff there is a transformation step $G \overset{p(a_1,\ldots,a_n)}{\Longrightarrow} H$.

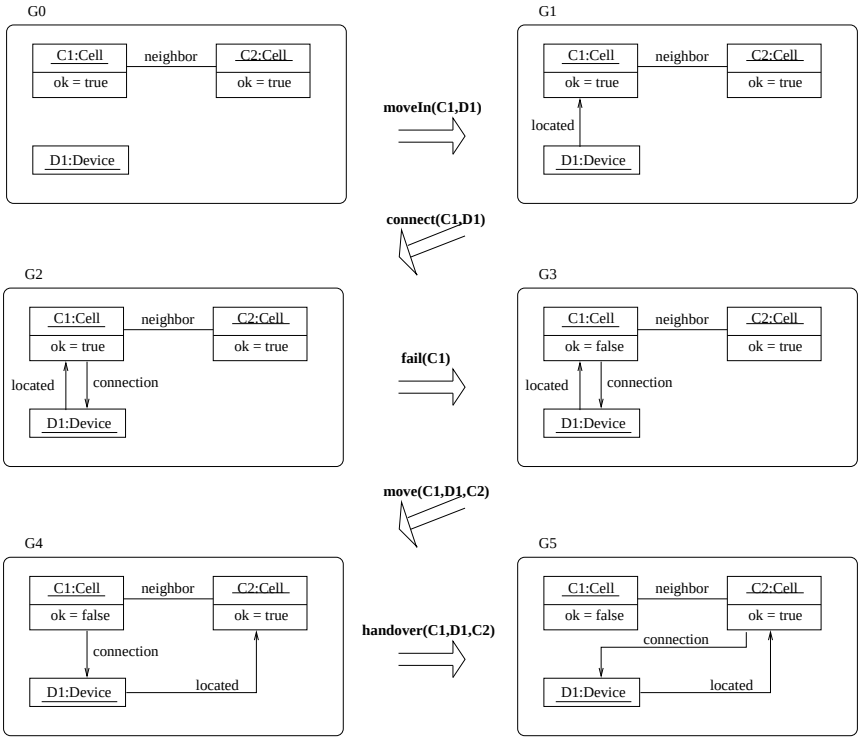
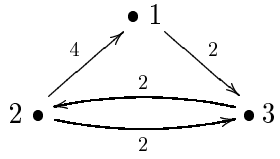


Figure 9. A transformation sequence



$$\begin{pmatrix} -2 & 0 & 2 \\ 4 & -6 & 2 \\ 0 & 2 & -2 \end{pmatrix}$$

Figure 10. Example CTMC with Q-matrix

4. Markov Chains and Stochastic Logic

After having described how to model mobility with graph transformation systems, and how based on this model non-functional requirements can be expressed, in this section we define Continuous Time Markov Chains (CTMCs) and explain how they can be analyzed. Furthermore, Continuous Stochastic Logic **CSL** is introduced to make assertions on CTMCs.

Markov Chains First we provide some basic notions adopting the Q-matrix, a kind of "incidence matrix" of the Markov Chain, as elementary notion (cf. [30]).

Definition 4.1. (Q-matrix)
Let S be a countable set. A Q-matrix on S is a real-valued matrix $Q = Q(s,s')_{s,s' \in S}$ satisfying the following conditions:

(i) $0 \leq -Q(s,s) < \infty$ for all $s \in S$,

(ii) $Q(s,s') \geq 0$ for all $s \neq s'$,

(iii) $\sum_{s' \in S} Q(s,s') = 0$ for all $s \in S$.

The Q-matrix is also called *transition rate matrix*. Some authors (e.g. [2]) use a more general notion of Q-matrix and call the matrices defined above *stable* and *conservative*. The most important notion is captured in the following

Definition 4.2. (CTMC)
A (homogeneous) Continuous Time Markov Chain is a pair $\langle S, Q \rangle$ where S is a countable set of states and Q is a Q-matrix on S.

If $s \neq s'$ and $Q(s,s') > 0$, then there is a transition from s to s'. The transition delay is exponentially distributed with rate $Q(s,s')$. Consequently, the probability that, being in s, the transition $s \to s'$ can be triggered within a time interval of length t is $1 - e^{-Q(s,s')t}$. The *total exit rate* $Q(s) = -Q(s,s)$ specifies the rate of leaving a state s to any other state. If the set $\{s' \mid Q(s,s') > 0\}$ is not a singleton, then there is a competition between the transitions originating in s. The probability that transition $s \to s'$ wins the 'race' is [2, §1.2].

$$\frac{Q(s,s')}{Q(s)}.$$

The transition probability matrix $P(t) = (P_{ss'}(t))_{s,s' \in S}$ describes the dynamic behavior of a CTMC. It is the minimal non-negative solution of the equation

$$P'(t) = QP(t), \quad P(0) = I.$$

Minimality means that for any other non-negative solution $\tilde{P}$, the inequality $\tilde{P}(t) \leq P(t)$ holds for all $t \geq 0$. The (s, s')-indexed entry of $P(t)$ specifies the probability that the system is in state s' after time t if it is in state s at present. Given an initial distribution $\pi(0)$, the *transient solution* $\pi(t) = (\pi_s(t))_{s \in S}$ is then

$$\pi(t) = \pi(0)P(t).$$

In the finite case, $P(t)$ can be computed by the matrix exponential function, $P(t) = e^{Qt}$, but the numerical behavior of the matrix exponential series is rather unsatisfactory [40]. Apart from the transient solution, which specifies the behavior as time evolves, the *steady state* or *invariant distribution* is of great interest.

Definition 4.3. (invariant distribution)
A map $\pi : S \to [0, 1]$ is an invariant distribution if

$$\pi Q = 0$$

$$\sum_{s \in S} \pi_s = 1.$$

In the example system shown in figure 10, the vector $\pi = \left(\frac{1}{3}, \frac{1}{6}, \frac{1}{2}\right)$ is an invariant distribution and the transient solution is given by

$$P(t) = \begin{pmatrix} -\frac{1}{3}e^{-6t} + e^{-4t} + \frac{1}{3} & \frac{1}{6} - \frac{1}{2}e^{-4t} + \frac{1}{3}e^{-6t} & \frac{1}{2} - \frac{1}{2}e^{-4t} \\ e^{-4t} - \frac{4}{3}e^{-6t} + \frac{1}{3} & \frac{4}{3}e^{-6t} + \frac{1}{6} - \frac{1}{2}e^{-4t} & \frac{1}{2} - \frac{1}{2}e^{-4t} \\ \frac{1}{3} - e^{-4t} + \frac{2}{3}e^{-6t} & \frac{1}{2}e^{-4t} - \frac{2}{3}e^{-6t} + \frac{1}{6} & \frac{1}{2} + \frac{1}{2}e^{-4t} \end{pmatrix}$$

Note that each row of $P(t)$ converges to π as $t \to \infty$, so the invariant distribution is always approached in the long term and is independent of the initial distribution, a property which can only be assured under certain reachability conditions. We say that a state s' *can be reached* from s, and write $s \rightharpoonup s'$, if there are states $s = s_0, \ldots, s_n = s'$, such that $Q(s_0, s_1) \cdot Q(s_1, s_2) \cdot \ldots \cdot Q(s_{n-1}, s_n) > 0$. If $s \rightharpoonup s'$ and $s' \rightharpoonup s$, then we say that s and s' *communicate*, and write $s \rightleftharpoons s'$.

Definition 4.4. (irreducible Q-matrix)
A Q-matrix Q is irreducible if $s \rightleftharpoons s'$ for all $s, s' \in S$.

Non-irreducible matrices can be partitioned into their irreducible components for analysis. We will therefore consider only irreducible matrices. It can be shown that the first condition of Definition 4.3 is equivalent to $\pi P = \pi$ if Q is finite and irreducible. The entries of the transition probability matrix then converge to the steady state,

$$\lim_{t \to \infty} P_{ss'}(t) = \pi_{s'}.$$

In the infinite case, stronger assumptions are necessary (positive recurrence, [30], Th. 3.5.5 and 3.6.2).

A Stochastic Temporal Logic. We use extended Continuous Stochastic Logic **CSL** as presented in [4] to describe properties of CTMCs. Suppose that a labelling function $L : S \to 2^{AP}$ is given, associating to every state s the set of atomic propositions $L(s) \subseteq AP$ that are valid in s. The syntax of **CSL** is:

$$\Phi ::= tt \mid a \mid \neg\Phi \mid \Phi_1 \wedge \Phi_2 \mid \mathcal{S}_{\lhd p}(\Phi) \mid \mathcal{P}_{\lhd p}(\Phi_1 \mathcal{U}^I \Phi_2)$$

where $\lhd \in \{\leq, \geq\}$, $p \in [0,1]$, $a \in AP$ and $I \subseteq \mathbf{R}$ is an interval. The other boolean connectives are defined as usual, i.e., $ff = \neg tt$, $\Phi \vee \Psi = \neg(\neg\Phi \wedge \neg\Psi)$ and $\Phi \to \Psi = \neg\Phi \wedge \Psi$. The steady-state operator $\mathcal{S}_{\lhd p}(\Phi)$ asserts that the steady-state probability of the formula Φ meets the bound $\lhd p$. The operator $\mathcal{P}_{\lhd p}(\Phi_1 \mathcal{U}^I \Phi_2)$ asserts that the probability measure of the paths satisfying $\Phi_1 \mathcal{U}^I \Phi_2$ meets the bound $\lhd p$.[1]

A *path* through a CTMC M is an alternating sequence $\sigma = s_0 t_0 s_1 t_1 \cdots$ with $Q(s_i, s_{i+1}) > 0$. The time stamps t_i denote the sojourn time in state s_i. For $t \in \mathbb{R}_{\geq 0}$ and i the smallest index with $t \leq \sum_{j=0}^{i} t_j$, let $\sigma@t = s_i$, the state of σ at time t. Let $Path^M$ be the set of all paths through M. The exponentially distributed application delays, together with an initial distribution, induce a probability measure Pr_s on sets of paths that start in s, for a formal definition see [4]. Then, $Prob^M(s, \phi_1 \mathcal{U}^I \phi_2)$ is defined as $Pr_s\{\sigma \in Path^M \mid \exists t \in I\ \sigma@t \models \Phi_2 \ \wedge\ \forall u \in [0, t[\ \sigma@u \models \Phi_1\}$, so $\Phi_1 \mathcal{U}^I \Phi_2$ asserts that Φ_2 will be satisfied at some time instant in the interval I and that at all preceding time instants Φ_1 holds. We define the semantics of CSL as follows:

$$\begin{array}{llll} s \models tt & \Leftrightarrow & s \in S & s \models \phi_1 \wedge \phi_2 \Leftrightarrow s \models \phi_1 \text{ and } s \models \phi_2 \\ s \models a & \Leftrightarrow & a \in L(s) & s \models \mathcal{S}_{\lhd p}(\phi) \Leftrightarrow \sum_{s \models \phi} \pi_s \lhd p \\ s \models \neg\phi & \Leftrightarrow & s \not\models \phi & s \models \mathcal{P}_{\lhd p}(\phi_1 \mathcal{U}^I \phi_2) \Leftrightarrow Prob^M(s, \phi_1 \mathcal{U}^I \phi_2) \lhd p \end{array}$$

CSL Example. Consider the CTMS of Fig. 10 with initial distribution $\pi(0) = (1, 0, 0)$. Define an atomic proposition a to be true in states 1 and 2, i.e. $L(1) = L(2) = \{a\}$, $L(3) = \emptyset$. Then the formula $\mathcal{S}_{\geq 0.5}(a)$ is true, because in the steady-state, a is fulfilled with probability 0.5, whereas $\mathcal{P}_{\geq 0.9}(a\, \mathcal{U}^{[0,1]}\, \neg a)$ is false, as the probability is only 0.86.

Model Checking huge CTMCs is a challenge which can be mastered by reducing them to smaller systems with the help of *stochastic bisimulations*.

Definition 4.5. Let $F \subseteq AP$ be a subset of atomic propositions. A stochastic *F-bisimulation* on $\langle S, Q, L\rangle$ is an equivalence R on S such that whenever $(s, s') \in R$ then $L(s) \cap F = L(s') \cap F$ and $\sum_{t \in C} Q(s,t) = \sum_{t \in C} Q(s',t)$ for every equivalence class $C \in S/R$. States s and s' are *F-bisimilar* if there exists an F-bisimulation R that contains (s, s').

F-bisimilar states satisfy the same **CSL**-formulae containing only atomic propositions from F and we can consider a quotient system $(S/R, Q_R, L_R)$ with $Q_R([s],[s']) = \sum_{(t,s) \in R} Q(s,t)$ and $L_R([s]) = L(s) \cap F$. There is an invariance result for this construction [4]:

Proposition 4.1. Let R be an stochastic bisimulation on $\langle S, Q, L\rangle$ and s a state in S. Then for all **CSL**-formulae ϕ containing only atomic propositions from F:

$$s \models_{\langle S,Q,L\rangle} \phi \quad \text{iff} \quad [s] \models_{\langle S/R, Q_R, L_R\rangle} \phi.$$

[1] The other path and state operators can be derived. Details are given in [4].

5. Stochastic Graph Transformation Systems

A stochastic graph transformation system associates with each rule name a positive real number representing the rate of the exponentially distributed delay of its application.

Definition 5.1. (stochastic GTS)
A stochastic graph transformation system $\mathcal{SG} = \langle \Sigma, ATG, P, \rho \rangle$ consists of a graph transformation system $\langle \Sigma, ATG, P \rangle$ and a function $\rho : P \to \mathbb{R}^+$ associating with every rule its application rate $\rho(p)$.

Example 5.1. (mobile system: application rates)
The application rates for the rules of our mobility example, which are needed to define the Q-matrix, are shown in the following table.

rule name p	rate $\rho(p)$
repair	500
fail	1
moveIn	1
moveOut	1
move	100
connect	10000
disconnect	10000
breakdown	100000
handOver	100000

For convenience, we only use integer values for rates. We assume that the average time needed to repair a broken host is optimistically much smaller than the average time until the next failure (the expected value of the application delay is just the inverse of the rate). If one likes to interpret the proposed values with the unit *per day*, this means that in the mean term, an error occurs once a day and can be repaired in approximately three minutes.
Moving in and out are assumed to happen equally often, but much more scarcely than moving into another cell. When a station fails, the connection is lost almost immediately, so the rate for rule breakdown is very high. Connecting, and disconnecting when the signal is lost, are assumed to happen equally fast. Finally, handover has to be possible within a few seconds to guarantee stability of connection.

Exponential distributions are single-parameter distributions that have a wide range of applications in analyzing the reliability and availability of electronic systems. Modeling a component's reliability with an exponential distribution presupposes that the failure rate is constant, which is generally true for electronic components during the main portion of their useful life. This means that the life of a component is independent of its current age (the *memoryless property*).
While reliability is one of the standard examples for exponential distributions, user mobility and connection duration can – at least in a coarse model – also be described as exponentially

distributed, see [11, 44]. Of course, more detailed and realistic models are not confined to this approach but use other stochastic techniques, too, in order to take into account aspects like speed or direction [45].

From Stochastic GTS to Markov Chains. We now show how a stochastic graph transformation system gives rise to a Markov Chain, so that the analysis techniques described in Sect. 4 can be applied. First, we need an important notion.

Definition 5.2. (finitely-branching)
Let $LTS = \langle L, S, \Rightarrow \rangle$ be a labeled transition system. Let $R(s,s') := \{l \mid \exists s' : \ s \stackrel{l}{\Longrightarrow} s'\}$ be the set of all transition labellings between s and s'. LTS is called *finitely-branching* iff $R(s,S)$ is finite for all $s \in S$.

We are now ready for the main result:

Proposition 5.1. (and Definition: induced Markov chain)
Let $\mathcal{SG} = \langle \Sigma, ATG, P, \rho \rangle$ be a stochastic graph transformation system with start graph G_0 and let the induced labeled transition system $LTS(\mathcal{G}, G_0) = \langle L, S, \Rightarrow \rangle$ be finitely-branching. Assume for all $s \in S$ that $\rho(p) = 0$ if $p \in R(s,s)$. We set[2]

$$Q(s,s') = \begin{cases} \sum\limits_{s \stackrel{p(\overline{a})}{\Longrightarrow} s'} \rho(p) & , \text{ for } \quad s \neq s' \\ -\sum\limits_{t \neq s} Q(s,t) & , \text{ for } \quad s = s'. \end{cases}$$

Then $\langle S, Q \rangle$ is a Continuous Time Markov chain, the *induced Markov chain of* $\mathcal{SG}$.

Proof:
We have to show that Q is a Q-matrix on the set of all graphs reachable from the start graph G_0 by applying the transformation rules, and so $\langle S, Q \rangle$ is a CTMC. The finite-branching property is crucial to ensure that condition (i) of Def. 4.1 is fulfilled, because the sum $\sum_{p(\overline{a}) \in R(s,s')} \rho(p)$ is finite as is its indexing set.
It also implies that $R(s,s')$ is finite for all $s \neq s'$. So compliance with (ii) is secured by Def. 5.1 as all $\rho(p)$ have to be finite, and part (iii) of Def. 4.1 is fulfilled trivially because of the definition of $Q(s,s)$.
The assumption that $\rho(p) = 0$ if $p \in R(s,s)$ for all $s \in S$ is not needed formally, but as we ignore loops, we require them to have infinite delay, so that they will never be applied. □

Note that we sum over all transformations between two isomorphism classes of states. All multiple rules p, p' linking two states have to be considered for the Q-matrix.
The initial distribution $\pi(0)$ is given by $\pi_s(0) = 1$ for $s = [G_0]$ and $\pi_s(0) = 0$ else. As discussed above, for assuring existence of a unique steady state solution it is beneficial if the Markov Chain is finite and irreducible, i.e., every state is reachable from every other state in the system. This property, which can be checked on the graph transition system, is typical for non-deterministic

[2] We use the convention $\sum\limits_{\emptyset} = 0$ for the empty sum.

models like the one given by our running example. Indeed, this model does not specify an individual application with determined behavior, but rather a whole class of mobile systems with similar structure and behavior, comparable to an architectural style [18]. For the case of infinite systems, analysis is possible under certain conditions (positive recurrence).

Stochastic Logic for induced Markov Chains. In order to use **CSL** for analyzing stochastic graph transformation systems, we have to define the set of atomic propositions AP as well as the labeling function L.

Definition 5.3. (stochastic logic over graph transformation systems)
Assume a stochastic graph transformation system $\mathcal{SG} = \langle TG, P, \pi, \rho \rangle$, a set of variables X, and an initial graph G_0. We define

$$AP = \{p(\bar{a}) \mid p : (L \leftarrow K \rightarrow R) \in P, \bar{a} = (a_1, ..., a_n)\}$$

as the set of all instantiated rules, and the labeling of states

$$L(s) = \{p(\bar{a}) \in AP \mid \exists t : s \stackrel{p(\bar{a})}{\Longrightarrow} t\}$$

to be the set of all instantiated rules on transitions outgoing from a state.

Thus, we can reason about the applicability of instantiated rules to attributed graphs, with the special case of property rules whose left- and right hand sides are copies of a pattern defining a structural graph property. The transition rates of property rules are set to 0, so that they do not affect the Q-matrix.
This enables us to answer the following questions.

- In the long run, is station z broken at most 1% of the time: $\mathcal{S}_{<0.01}\ z.ok = false$?
- Is the overall connectivity of device x greater than 5 %: $\mathcal{S}_{>0.5}\ con(x)$?
- Is the probability of device y being connected before t days from now at least 0.9: $\mathcal{P}_{\geq 0.9}\ (true\ \mathcal{U}^{[0,t]}\ con(y))$?

Bisimulation for induced CTMCs Let's assume we are only interested in properties concerning a certain subset of nodes with identities in $ID' \subseteq ID$, say, only the cells $C1$ and $C2$. We can thus forget about the attributes in $ID \backslash ID'$ and delete the corresponding attribute edges in a graph $G = (V_1, V_2, E_i, s_i, t_i)_{i=1,2,3}$, setting

$$E_2' = E_2 \backslash \{e \mid t_2(e) \text{ is of type } ID \backslash ID'\},$$

which gives us a reduced graph $G|_{ID'}$ (the other vertex and edge sets remain unchanged). Let G and H be graphs such that the reduced graphs $G|_{ID'}$ and $H|_{ID'}$ are isomorphic via an isomorphism ϕ, cf. Fig. 11. As all instantiated rules derive from (abstract) rules, the transitions out of G and H are instantiations of the same rules, and therefore have the same rates. For instance, while G allows a rule application $moveOut(C1, D1)$, H allows the production $moveOut(C1, D2)$. We thus have to adapt the instantiations of rule $moveOut$ to show that there are similar production steps in H. This means that states $[G]$ and $[H]$ are F-bisimilar, if F is the set of all attributes which are not of type $ID \backslash ID'$.

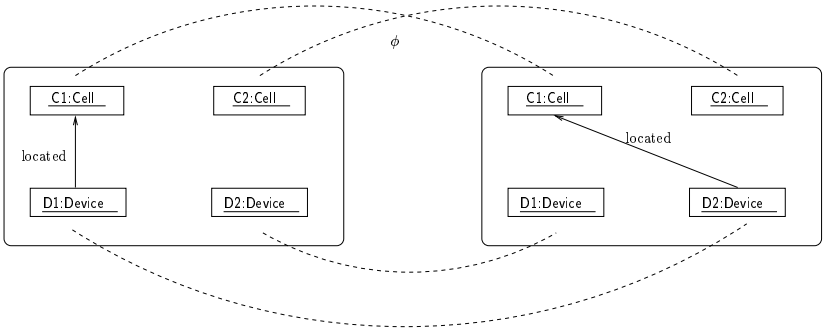


Figure 11. Two isomorphic reductions for $ID' = \{C1, C2, D1, D2\}$.

Proposition 5.2. Let $\langle S, Q, L\rangle$ be an induced CTMC, $ID' \subseteq ID$ a subset of identity attributes and F the set of all attributes which are not of sort $ID \backslash ID'$. The relation R on S comprising all pairs $\langle [G], [H]\rangle$ with $G|_{ID'} \cong H|_{ID'}$ is an F-bisimulation.

Proof:
Let G, H be two graphs such that $G|_{ID'} \cong H|_{ID'}$ via an isomorphism ϕ. It is clear that $L([G]) \cap F = L([H]) \cap F$, because G and H agree with respect to all non-$ID \backslash ID'$-attributes, i.e. those relevant for atomic propositions in F.
To establish the summation condition, we show that for every rule application $\langle [G], p(a_1, \ldots, a_n), [G']\rangle \in T$, there is a similar transition $\langle [H],\, p(\tilde{a}_1, \ldots, \tilde{a}_n),\, [H']\rangle \in T$ such that $G'|_{ID'} \cong H'|_{ID'}$. We thus have to find an appropriate instantiation $\tilde{a}_1, \ldots, \tilde{a}_n$ for rule p. For values a_i concerning attributes of sort other than $ID \backslash ID'$, we set $\tilde{a}_i = a_i$.
The a_i of sort $ID \backslash ID'$, e.g. those we like to ignore, have to be adapted: Let a_i be such an attribute with node v be associated to it in G – as it is an identity attribute, there can not be more than one such node in each graph. Now set $\tilde{a}_i$ to be the attribute value of $\phi(v)$. Then, we have $\langle [H],\, p(\tilde{a}_1, \ldots, \tilde{a}_n),\, [H']\rangle \in T$, where H' is the graph resulting from the application of $p(\tilde{a}_1, \ldots, \tilde{a}_n)$ to H. The reductions of G' and H' are isomorphic, $G'|_{ID'} \cong H'|_{ID'}$. Altogether, we see that $[G]$ and $[H]$ are F-bisimilar.

□

We can thus, by a suitable adaptation of the attributation, apply the same rules to two states which differ only with respect to the attributes in $ID \backslash ID'$. This means that we can reduce the CTMC dramatically for model checking purposes. There is also a way to obtain the quotient system immediately: Use as states isomorphism classes of graphs disregarding the attributes of no interest, but take multiple transitions for the same rule application if there are differently attributed instantiations.
If the initial graph is finite, the same holds for all graphs in the system if transformations preserve finiteness. In this case, the finite-branching condition is ensured if every finite graph has only a finite number of applicable rules. This is trivial in the case of a finite set of rules in the system. However, there may by occasions where the set of rules is infinite, e.g., when rules with path expressions are regarded as a rule schema expanding to countably infinite sets of rules. In this

case, the condition may be violated if the given graph contains a cycle and thus an infinite number of paths.

6. Tool Support

Our interpretation of stochastic logic, although semantically satisfactory, has the disadvantage of being "non-standard": The construction of states with assignments is not supported by any graph transformation tool. However, even when analyzing the small example in this paper, we have found that tool support is indispensable in a stochastic setting, much more so than for the analysis of standard transition systems.

A possible way out that worked well in the example is the encoding of assignments into the graphical structure. The idea is to introduce a vertex for each logical variable together with an edge pointing to the referenced graph element. Extending the rules in a similar way, we obtain a slightly awkward, but semantically equivalent encoding of the transition system.

We have implemented this idea in the GROOVE tool [34] which allows the simulation of graph transformation systems and exports the labeled transition system generated from it. While GROOVE can in theory generate arbitrary large state spaces, we were only able to generate state spaces to the order of 10^7 due to hardware and time limitations. To our best knowledge, GROOVE is the only available tool with a strong focus on state space generation including import/export functionality.

Given this system and providing in a separate file the rule application rates, we generated the Markov Chain in various formats for analysis in numerical tools like MAPLE or probabilistic model checkers like PRISM [24] or $E \models MC^2$ [21]. Extracting loops via property rules from the transition system we also generate the labeling of states by atomic propositions. Thus we can verify CSL formulae with instantiated graph patterns as atomic propositions against CTMCs generated from stochastic graph transformation systems.

It is ongoing work to port our tool chain in a usable modelling and analysis tool. While PRISM has exciting features like calculating exact probabilities for CSL formulae or performing test runs, it has no well defined API to use from other programs. Since our focus is not to perform elaborated stochastic analyses, we consider using $E \models MC^2$ which is more suitable as a model-checking *component*. Integrating $E \models MC^2$ with GROOVE we will obtain a prototype for stochastic graph transformation.

7. Conclusion

We have proposed an approach for analyzing graph transformation systems with stochastic methods. We have shown that under certain conditions, a stochastic graph transformation system induces a Continuous Time Markov Chain. This opens the door to a wide range of applications in modeling, and to tools and numerical methods for analysis.

We have constructed an experimental tool chain consisting of GROOVE and PRISM. Another approach to generate the input to the model checker is followed by [43, 37]. In that work, a transition system specification is generated directly from the given graph grammar. This approach could allow us to benefit from built-in optimizations or on-the-fly techniques, because

the construction of the transition system is done inside the model checker. It also allows the use of different modeling tools, like AGG or PROGRES [42, 38], which support attributed graph transformation systems. These options are currently under investigation.
Another line of research is the generalization of the basic theory of graph transformation systems concerning independence and critical pairs, concurrency, and synchronization to the stochastic case. In particular, the last issue could be relevant for a compositional translation of graph transformation systems into transition system specifications which are composed of modules combined by synchronous parallel composition.

Acknowledgement. The authors wish to thank Arend Rensink for his help with the GROOVE tool and enlightening discussions on graph transformation and temporal logic.

References

[1] M. Ajmone-Marsan, G. Balbo, G. Conte, S. Donatelli, and G. Franceschinis. *Modelling with Generalized Stochastic Petri Nets.* Wiley Series in Parallel Computing. John Wiley and Sons, 1995.

[2] William G. Anderson. *Continuous-Time Markov Chains.* Springer, 1991.

[3] L. Andrade, P. Baldan, and H. Baumeister. AGILE: Software architecture for mobility. In *Recent Trends in Algebraic Develeopment, 16th Intl. Workshop (WADT 2002)*, volume 2755 of *Lecture Notes in Computer Science*, Frauenchiemsee, 2003. Springer-Verlag.

[4] Christel Baier, Boudewijn R. Haverkort, Holger Hermanns, and Joost-Pieter Katoen. Model checking continuous-time markov chains by transient analysis. In *Computer Aided Verification*, pages 358–372, 2000.

[5] Falko Bause and Pieter S. Kritzinger. *Stochastic Petri Nets.* Vieweg Verlag, 2nd edition, 2002.

[6] Ed Brinksma and Holger Hermanns. Process algebra and markov chains. In J.-P. Katoen E. Brinksma, H. Hermanns, editor, *FMPA 2000*, number 2090 in Lecture Notes in Computer Science, pages 183–231. Springer, 2001.

[7] G.A. Churchill. Stochastic models for heterogeneous DNA sequences. *Bull. Math. Biol.*, 51(1):79–94, 1989.

[8] A. Corradini, U. Montanari, and F. Rossi. Graph processes. *Fundamenta Informaticae*, 26(3,4):241–266, 1996.

[9] A. Corradini, L. Ribeiro, and F.L. Dotti. A graph transformation view on the specification of applications using mobile code. In G. Taentzer L. Baresi, M. Pezze and C. Zaroliagis, editors, *Proceedings 2nd Int. Workshop on Graph Transformation and Visual Modeling Technique (GT-VMT 01)*, volume 50.3 of *Elect. Notes in Th. Comput. Sci.* Elsevier Science, 2001.

[10] P.R. D'Argenio. *Algebras and Automata for Timed and Stochastic Systems.* IPA Dissertation Series 1999-10, CTIT PhD-Thesis Series 99-25, University of Twente, November 1999.

[11] J. Diederich, L. Wolf, and M. Zitterbart. A mobile differentiated services QoS model. In *Proceedings of the 3rd Workshop on Applications and Services in Wireless Networks*, 2003.

[12] H. Ehrig and B. Mahr. *Fundamentals of algebraic specifications 1: Equations and initial semantics*, volume 6 of *EACTS Monographs on Theoretical Computer Science.* Springer, Berlin, 1985.

[13] H. Ehrig, M. Pfender, and H.J. Schneider. Graph grammars: an algebraic approach. In *14th Annual IEEE Symposium on Switching and Automata Theory*, pages 167–180. IEEE, 1973.

[14] Hartmut Ehrig, Annegret Habel, Julia Padberg, and Ulrike Prange. Adhesive high-level replacement categories and systems. In Hartmut Ehrig, editor, *Proceedings of the Second International Conference on Graph Transformations*, volume 3256 of *Lecture Notes in Computer Science*, pages 144–160. Springer, 2004.

[15] Hartmut Ehrig, Ulrike Prange, and Gabriele Taentzer. Fundamental theory for typed attributed graph transformation. In Hartmut Ehrig, editor, *Proceedings of the Second International Conference on Graph Transformations*, volume 3256 of *Lecture Notes in Computer Science*, pages 161–177, 2004.

[16] F. Gadducci and U. Montanari. A concurrent graph semantics for mobile ambients. In St. Brooks and M. Mislove, editors, *Proc. Mathematical Foundations of Programming Semantics, Aarhus*, volume 45 of *Elect. Notes in Th. Comput. Sci.* Elsevier Science, 2001.

[17] T. Gilb. *Principles of Software Engineering Management.* Addison-Wesley, 1988.

[18] P. Guo and R. Heckel. Conceptual modelling of styles for mobile systems: A layered approach based on graph transformation. In *Proc. IFIP TC8 Working Conference on Mobile Information Systems, Oslo, Norway*, pages 65–78, 2004.

[19] A. Habel, R. Heckel, and G. Taentzer. Graph grammars with negative application conditions. *Fundamenta Informaticae*, 26(3,4):287 – 313, 1996.

[20] Reiko Heckel, Georgios Lajios, and Sebastian Menge. Stochastic graph transformation. In *Proceedings of the 2nd International Conference on Graphtransformation, ICGT '04*, Lecture Notes in Computer Science, 2004.

[21] H. Hermanns, J. P. Katoen, J. Meyer-Kayser, and M. Siegle. A tool for model checking markov chains. *Software Tools for Technology Transfer*, 2002.

[22] D. Hirsch and U. Montanari. Consistent transformations for software architecture styles of distributed systems. In G. Stefanescu, editor, *Workshop on Distributed Systems*, volume 28 of *Electronic Notes in TCS*, 1999.

[23] Leonard Kleinrock. *Queueing Systems, Volume 1: Theory.* Wiley-Interscince, 1975.

[24] M. Kwiatkowska, G. Norman, and D. Parker. PRISM: Probabilistic symbolic model checker. In T. Field, P. Harrison, J. Bradley, and U. Harder, editors, *Proc. 12th Int. Conf. on Modelling Techniques and Tools for Computer Performance Evaluation (TOOLS'02)*, volume 2324 of *Lecture Notes in Computer Science*, pages 200–204. Springer, April 2002.

[25] K. Lari and S.J. Young. The estimation of stochastic context-free grammars using the inside-outside algorithm. *Computer Speech and Language*, 4:35–56, 1990.

[26] R. Milner. *Communication and Concurrency.* Prentice-Hall, 1989.

[27] M. K. Molloy. *On the Integration of Delay and Throughput Measures in Distributed Processing Models.* PhD thesis, University of California, 1981.

[28] M. Mosbah. Probabilistic graph grammars. *Fundamenta Informaticae*, vol. 26:pp. 341–360, 1996.

[29] S. Natkin. *Les Réseaux de Petri Stochastiques et leur Application ê l'Evaluation des Systčmes Informatiques.* PhD thesis, CNAM Paris, 1980.

[30] James R. Norris. *Markov Chains.* Cambridge University Press, 1997.

[31] Tim Oates, Shailesh Doshi, and Fang Huang. Estimating maximum likelihood parameters for stochastic context-free graph grammars. In *Inductive Logic Programming: 13th International Conference, ILP 2003*, pages 281–298, 2003.

[32] Corrado Priami. Stochastic pi-calculus. *Computer Journal*, 38(7):578–589, 1995.

[33] Lawrence R. Rabiner. *A tutorial on hidden Markov models and selected applications in speech recognition*, pages 267–296. Morgan Kaufmann Publishers Inc., 1990.

[34] A. Rensink. The GROOVE simulator: A tool for state space generation. In J.L. Pfaltz, M. Nagl, and B. Böhlen, editors, *Applications of Graph Transformation with Industrial Relevance Proc. 2nd Intl. Workshop AGTIVE'03, Charlottesville, USA, 2003*, volume 3062 of *Lecture Notes in Computer Science*. Springer-Verlag, Berlin, 2004.

[35] F. Rossell"o and G. Valiente. Graph transformation in molecular biology. In H.-J. Kreowski, U. Montanari, F. Orejas, G. Rozenberg, and G. Taentzer, editors, *Formal Methods in Software and System Modeling*, volume 3393 of *Lecture Notes in Computer Science*, pages 116–133. Springer, 2005.

[36] G. Rozenberg, editor. *Handbook on Graph Grammars: Foundations*, volume 1. World Scientific, Singapore, 1997.

[37] Ákos Schmidt and Dániel Varró. CheckVML: A tool for model checking visual modeling languages. In Perdita Stevens, Jon Whittle, and Grady Booch, editors, *UML 2003 - The Unified Modeling Language. Model Languages and Applications. 6th International Conference, San Francisco, CA, USA, October 2003, Proceedings*, volume 2863 of *Lecture Notes in Computer Science*, pages 92–95. Springer, 2003.

[38] Andy Schürr. PROGRES: a VHL-language based on graph grammars. In *Proc. 4th Int. Workshop on Graph-Grammars and Their Application to Computer Science*, number 532 in Lecture Notes in Computer Science, pages 641–659. Springer-Verlag, 1991.

[39] Baozhen Shan. Stochastic context-free graph grammars for glycoprotein modelling. In *CIAA*, pages 247–258, 2004.

[40] W. Stewart. *Introduction to the Numerical Solution of Markov Chains*. Princeton University Press, 1994.

[41] G. Taentzer. Hierarchically distributed graph transformation. In *5th Int. Workshop on Graph Grammars and their Application to Computer Science, Williamsburg '94, LNCS 1073*, pages 304 – 320, 1996.

[42] Gabriele Taentzer. AGG: A graph transformation environment for modeling and validation of software. In John L. Pfaltz, Manfred Nagl, and Boris Böhlen, editors, *Applications of Graph Transformations with Industrial Relevance, Second International Workshop*, number 3062 in Lecture Notes in Computer Science, pages 446–453, 2004.

[43] Dániel Varró. Automated formal verification of visual modeling languages by model checking. *Journal of Software and Systems Modelling*, 2003. Accepted to the Special Issue on Graph Transformation and Visual Modelling Techniques.

[44] Oliver T. W. Yu and Victor C. M. Leung. Adaptive resource allocation for prioritized call admission over an ATM-based wireless PCN. *IEEE Journal on Selected Areas in Communications*, 15:1208–1224, 1997.

[45] M. Zonoozi and P. Dassanayake. User mobility modeling and characterization of mobility patterns. *IEEE Journal on Selected Areas in Communications*, 15:1239–1252, 1997.