Analysis of infinite-state graph transformation systems by cluster abstraction*

Peter Backes and Jan Reineke

Universität des Saarlandes, Saarbrücken, Germany
{rtc,reineke}@cs.uni-saarland.de

Abstract. Analysis of distributed systems with message passing and dynamic process creation is challenging because of the unboundedness of the emerging communication topologies and hence the infinite state space. We model such systems as graph transformation systems and use abstract interpretation to compute a finite overapproximation of the set of reachable graphs. To this end, we propose cluster abstraction, which decomposes graphs into small overlapping clusters of nodes. Using `astra`, our implementation of cluster abstraction, we are for the first time able to prove several safety properties of the merge protocol. The merge protocol is a coordination mechanism for car platooning where the leader car of one platoon passes its followers to the leader car of another platoon, eventually forming one single merged platoon.

Keywords: graph transformation, abstract interpretation, parameterized verification, shape analysis, distributed message-passing systems

1 Introduction

Distributed message-passing systems such as car platoons and drone swarms consist of an unbounded and dynamically changing number of agents. These agents act in a coordinated fashion using wireless ad-hoc networks to achieve common goals. For this purpose, they assume different roles in a logical communication topology that is established on top of the physical communication medium. These communication topologies, which consist of unidirectional channels between pairs of agents, are formed by distributed protocols that all agents execute concurrently.

The purpose of our analysis is to determine the emerging topologies, which can then be used to evaluate safety properties, ensuring that the system will never reach a state with an undesired topology.

We model such systems by graph transformation systems, i.e., graphs modified by transformation rules. Graph transformation is a lingua franca with a

* This work was partially supported by the German Research Council (DFG) as part of the Transregional Collaborative Research Center "Automatic Verification and Analysis of Complex Systems" (SFB/TR 14 AVACS). See http://www.avacs.org/ for more information.

broad range of applications in systems modeling, all of which become potential use cases for our method. Many domain-specific models can be translated automatically into graph transformation systems.

In the graph transformation framework, we represent agents as labeled nodes and communication channels and message queues as labeled, directed edges of a graph. We model the dynamics of the system, like agents sending and receiving messages, detecting each other's presence and setting up and closing communication channels, as transformation rules that are applied to the graphs. Those rules match subgraph patterns in a graph, optionally restricted by application conditions, and replace them by modified subgraphs.

The main challenges with respect to the analysis of the systems under consideration are the unboundedness of the graphs, caused by the unboundedness of the number of agents, and the concurrency of the computations of the participating agents. In particular, the state space of such systems is infinite, and naive state-space exploration cannot be used for our purpose. Instead, we use abstract interpretation, overapproximating the graphs by abstract representations of bounded size.

To compute this overapproximation, we lift rule application to the abstract level, reducing the infinite concrete state space to a finite abstract one: We match the rules on the abstract representation, partly undo the abstraction, just enough to apply the rule, and restore abstraction on the result. By fixed-point iteration, we end up with one final abstract topology, an overapproximation of all graphs the system may produce.

The crucial idea of our abstraction is to decompose graphs into overlapping, simultaneously evolving *clusters*, one per node of the graph—*cluster abstraction*. Each cluster consists of a *core node*, corresponding to the specific node under consideration, and *peripheral nodes*, corresponding to the immediate neighborhood of the core node, i.e., its adjacent nodes. We keep the edges between peripheral nodes and the core node, as well as the core node itself, completely concrete. The neighborhood of a node may be unbounded, e.g., in some protocols a leader may have an unbounded number of followers. To arrive at a finite abstract domain, we use approximated counting: two or more neighborhood nodes that are similar become one summary node in the periphery. By a three-valued abstraction, we preserve information about the neighborhood edges where possible.

We have implemented cluster abstraction in a tool called `astra`. In addition to benchmarks from the literature, ranging from red-black trees to firewalls, we successfully apply `astra` to the merge protocol. The merge protocol is a coordination mechanism for car platooning that could not be fully analyzed with previous approaches.

Outline. In Section 2, we describe the graph transformation framework our work is based upon. Section 3 introduces cluster abstraction and the computation of the corresponding abstract transformer. In Section 4 we present our tool implementation `astra` and experimental results. After discussing related work in Section 5, we conclude the paper in Section 6.

2 Background

2.1 Graph Preliminaries

Our framework is based on directed graphs with edge and node labels. We allow several edges between the same pair of nodes, but only as long as their direction or edge label differ.

Definition 1 (Graph). *Let $\mathcal{V}$ be a set of node names, $\mathcal{N}$ a set of node labels and $\mathcal{E} = \{\beta_1, \ldots, \beta_{|\mathcal{E}|}\}$ a set of edge labels. A* graph *G is a tuple $(V_G, E_G^{\beta_1}, \ldots, E_G^{\beta_{|\mathcal{E}|}}, \ell_G)$ where $V_G \subseteq \mathcal{V}$ is the set of nodes, $\ell_G : V_G \to \mathcal{N}$ is the node label assignment and $E_G^\beta \subseteq V_G \times V_G$ is the set of edges with label $\beta \in \mathcal{E}$.*

For simplicity, we assume a globally unique set $\mathcal{V}$ of node names, plus a globally unique set of node labels $\mathcal{N}$ and edge labels $\mathcal{E}$. Note the difference between node names and node labels: Nodes may share the same node label and nodes from different graphs may share the same node name, but nodes from the same graph always have different node names. We use mappings over node names to relate nodes of different graphs. We denote the set of graphs as $\mathcal{G}$.

Graph morphisms map the nodes of one graph to the nodes of another graph such that the node labels agree and all edges are preserved. The existence of a graph morphism means that one graph is a subgraph of another.

Definition 2 (Partial and total graph morphism, subgraph relation). *Let G and H be graphs. An injective partial function $h : V_G \rightharpoonup V_H$ is a* partial graph morphism *iff $\ell_G \cap (\mathrm{def}(h) \times \mathcal{N}) = h \circ \ell_H$ and for all $\beta \in \mathcal{E}$, $h(E_G^\beta) \subseteq E_H^\beta$. We call h a* (total) graph morphism *iff it is a total function, i.e., $h : V_G \to V_H$. If an injective graph morphism exists, G is a* subgraph *of H, denoted by $G \lesssim_h H$.*

For the purpose of abstraction, we will later need to consider *spokes* between nodes, not merely individual edges. Spokes represent the configuration of edges, i.e., direction and edge label of edges between two given nodes.

Definition 3 (Spoke). *Let G be a graph and $v, v' \in V_G$. Then the* spoke *between v and v' in G is the pair $SP_G(v, v') := (\{\beta \in \mathcal{E} \mid (v, v') \in E_G^\beta\}, \{\beta \in \mathcal{E} \mid (v', v) \in E_G^\beta\})$ We denote the set of all spokes $2^\mathcal{E} \times 2^\mathcal{E}$ by $\mathcal{SP}$. An alternative notation for the empty spoke $(\emptyset, \emptyset)$ shall be $\emptyset$.*

2.2 Graph Transformation Systems

Graph transformation systems rewrite graphs according to transformation rules, starting with some initial graph. Rule application can be restricted via negative application conditions. In this paper, we consider negative application conditions specified by partner constraints. A partner constraint prohibits incident edges with a specific direction and label to an adjacent node with a specific label.

Definition 4 (Partner constraint). *A* partner constraint *is a tuple $(d, \beta, l) \in \mathcal{PC} = \{in, out\} \times \mathcal{E} \times \mathcal{N}$ where d is a direction, β an edge label and l a node label.*

Transformation rules consist of a left hand side graph matched against the host graph, a right hand side graph by which the left hand side graph is replaced, and a mapping that describes node correspondence between the left and the right hand side graph. Additionally, for each left hand side node, an optional set of partner constraints can be specified.

Definition 5 (Graph transformation rule). *A* graph transformation rule *is a tuple* (L, h, p, R) *where* L *(the* left hand side*) and* R *(the* right hand side*) are graphs,* $h : V_L \rightharpoonup V_R$ *is an injective partial mapping from the left to the right hand side and* $p : V_L \rightharpoonup 2^{\mathcal{PC}}$ *specifies the partner constraints.*

For simplicity, in the following, we assume one globally unique set of graph transformation rules $\mathcal{R}$ and an initial graph I, which, together with node and edge labels, form the graph transformation system $\mathcal{S} := (\mathcal{N}, \mathcal{E}, I, \mathcal{R})$. We further assume for simplicity that in each rule, either all or none of its right hand side nodes are newly created.

For a rule to match, its left hand side must be a subgraph of the host graph and all negative application conditions need to be satisfied: We check each partner constraint against the matched node and its neighborhood.

Definition 6 (Match, partner constraint satisfaction). *Let* $r = (L, h, p, R)$ *be a rule,* G *a graph and* $m : V_L \to V_G$*. Then* m *is a* match *from* r *to* G *iff* $L \lesssim_m G$ *such that the partner constraints* p *are satisfied: For each* $v \in \mathrm{def}(p)$ *and* $\beta \in \mathcal{E}$*, we have* $p(v) \cap E = \emptyset$*, where*

$$\begin{aligned} E = \{&(out, \beta, \ell_G(u')) \mid (m(v), u') \in E_G^\beta\} \\ \cup \{&(in, \beta, \ell_G(u')) \mid (u', m(v)) \in E_G^\beta\})\} \end{aligned}$$

Rule application requires that the left hand side matches the host graph. A result graph is the host graph with labels of matched nodes changed as specified by h, nodes and edges of the left hand side removed and nodes and edges of the right hand side added as specified by the rule. Added nodes may be assigned any unused node name, thus the result is not unique. We obtain a mapping from the unchanged nodes of the host graph to the result graph as a byproduct. A graph is directly derived from a host graph according to some rule iff there is any way to apply the rule and obtain this graph as the result.

Definition 7 (Rule application, direct derivation). *Let* $r = (L, h, p, R)$ *be a rule,* G, H *graphs,* $m : V_L \to V_G$ *an injective graph morphism and* $D := m(V_L \setminus \mathrm{def}(h))$ *the set of deleted nodes. Then* H *is a result of the* application *of* r *to* G *with respect to* m*, written* $G \overset{r,m}{\rightsquigarrow} H$*, iff there is an injective mapping* $m' : V_R \setminus h(V_L \setminus \mathrm{def}(m)) \to V_H$ *such that* $m = h \circ m'$*,* $V_H \cap D = \emptyset$ *and*

$$\begin{aligned} \ell_H &= (\ell_G \setminus (D \times \mathcal{N}) \cup (m'^{-1} \circ \ell_R) \\ V_H &= (V_G \setminus D) \cup m'(V_R) \\ E_H^\beta &= ((E_G^\beta \setminus m(E_L^\beta)) \cap (V_H \times V_H)) \cup m'(E_R^\beta) \end{aligned}$$

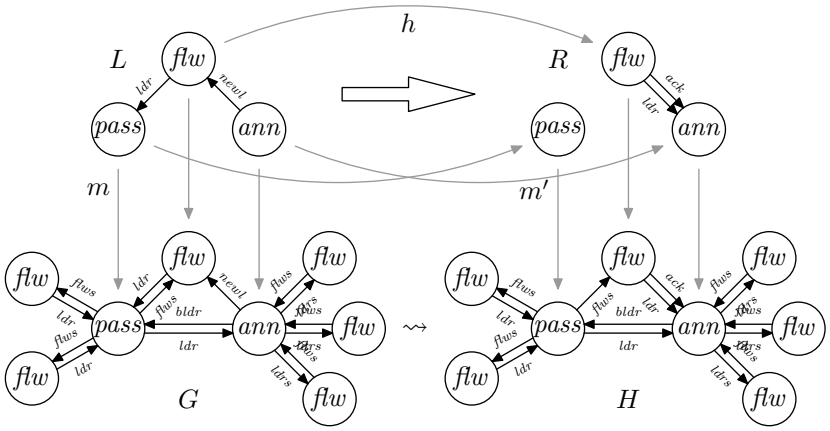

Fig. 1. An example of rule application: A rule $(L, h, \emptyset, R)$ transforming graph G into graph H, as it occurs in the merge protocol [1].

The direct derivation relation $\overset{r}{\rightsquigarrow}$ *is a relation over* $\mathcal{G} \times \mathcal{G}$ *where* $G \overset{r}{\rightsquigarrow} H$ *iff there is a match* m *such that* $G \overset{r,m}{\rightsquigarrow} H$.

In this paper, we are interested in reachability properties, i.e., is a graph with a particular property reachable or not? Therefore, we define the semantics of the graph transformation system simply as the set of reachable graphs.

Definition 8 (Graph transformation system semantics). *The* semantics of a graph transformation system $\mathcal{S}$ *is the smallest set such that* $I \in [\![\mathcal{S}]\!]$ *and, if there are graphs* $G \in [\![\mathcal{S}]\!]$ *and* H *and a rule* $r \in \mathcal{R}$ *such that* $G \overset{r}{\rightsquigarrow} H$*, then* $H \in [\![\mathcal{S}]\!]$.

2.3 The Merge Protocol

Our main benchmark is a graph transformation system modeling the *merge protocol* [2,1]. This protocol is used in car platooning, where autonomous cars on highways form platoons driving at constant speed and distance to save fuel. Its purpose is to allow (1) two cars to form a platoon with the car in front becoming the platoon leader and the other becoming its follower, (2) a car joining an existing platoon as a new follower and (3) merging of two platoons, with the leader on the back handing over all its followers to the leader in front, eventually itself becoming one of the followers.

What makes the merge protocol so difficult to analyze is the vast range of topological configurations all present and evolving at the same time, caused by the protocol's massively distributed nature. For example, a car may receive at any time a request to form a platoon, at the same time receive a request to merge with another platoon, all while being in the middle of any intermediate step of a merge operation, or sending such a request itself—and this happening with an arbitrary number of cars in different contexts at once. This is different from the

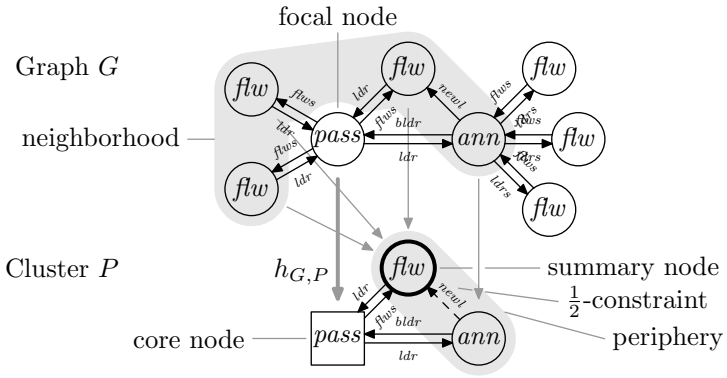

Fig. 2. An example of local abstraction: Graph G with the *pass*-labeled node as focal node is abstracted into cluster P. The periphery of P is an abstraction of the neighborhood of the focal node in G.

typical setting of shape analysis, i.e., the static analysis of heap-manipulating programs, where data structures typically have a regular global structure modified only at some select points, those referenced by pointers from the stack. On the other hand, shape analyses are often employed to prove *global* invariants about the heap structure, such as the sortedness of a binary tree, whereas in the analysis of the merge protocol, our goal is to show that undesired *local* configurations never occur.

3 Analysis

3.1 Cluster Abstraction

In the graph transformation systems we consider, unbounded numbers of nodes may be created dynamically. Thus, the state space of such systems is infinite in size, making exact analysis by concrete state-space exploration impossible. To overcome this challenge, we employ a bounded abstraction: each concrete graph of arbitrary size is represented by an abstract graph of bounded size, reducing the infinite state space to a finite one.

We apply *local abstraction* to each node of a given graph, obtaining a bounded set of *clusters*. Local abstraction focusses on one specific node in the graph, henceforth called the *focal node*. It abstracts from all nodes in the graph except for the focal node and its immediate neighborhood, referred to as the *periphery* in the abstraction. The neighborhood consists of the incident edges and the adjacent nodes of the focal node. In addition, neighborhood nodes are merged into summary nodes if they are connected to the focal node by the same spoke (see Definition 3). Further, edges among neighborhood nodes are abstracted into three-valued *constraints*. This yields a *cluster*, which consists of the core node and its periphery. The core node shall have the unique name $core \in \mathcal{V}$. Figure 2 illustrates local abstraction, which is formally defined later.

Local abstraction asymmetrically preserves information about one specific node and some information about its neighborhood only, none about the rest of the graph. To capture the structure of the entire graph, we apply local abstraction to all of its nodes. As the neighborhoods of nodes are overlapping, this preserves some information about the global graph structure.

While a concrete graph may contain an arbitrary number of nodes, the set of distinct clusters is bounded. Thus the abstraction is bounded.

The process described above yields a set of clusters that may contain clusters that differ only with respect to constraints between peripheral nodes. To reduce analysis complexity, such clusters are merged by loosening the constraints.

Definition 9 (Cluster). *A* cluster P *is a tuple* $(G_P, S_P, C_P^{\beta_1}, \ldots, C_P^{\beta_{|\mathcal{E}|}})$ *where* $G_P = (V_P, E_P^{\beta_1}, \ldots, E_P^{\beta_{|\mathcal{E}|}}, \ell_P)$ *is a graph,* $\{core\} \subseteq V_P \subseteq \{core\} \cup \mathcal{SP} \times \mathcal{N}$ *are the node names,* $S_P \subseteq D_P = V_P \setminus \{core\}$ *is a set of* summary nodes*, with* D_P *the set of* peripheral nodes*,* $C_P^\beta : ((D_P \times D_P) \setminus \{(v,v) \mid v \in D_P \setminus S_P\}) \to \{0, 1, \frac{1}{2}\}$ *are the* peripheral constraints*,* $E_P^\beta \subseteq (\{core\} \times V_P) \cup (V_P \times \{core\})$ *for any* $\beta \in \mathcal{E}$ *and* $SP_P(core, v) \neq \emptyset$ *for all* $v \in D_P$*. We denote the set of all clusters by* $\mathcal{P}$*.*

Given a graph G and one of its nodes v, local abstraction yields a cluster P with a core node that corresponds to the focal node v. P has one peripheral node per uniquely connected neighborhood node of v, that is, with a unique configuration of neighborhood node label plus non-empty spoke.

The edges connecting the neighborhood nodes are abstracted as follows: If, in G, there are β-labeled edges from all source nodes V_1 to all target nodes V_2, both sets each corresponding to a (possibly summary) node in P, then there is a peripheral 1-constraint in P that involves two nodes corresponding to V_1 and V_2. If there are some, but not all such β-labeled edges, we use a $\frac{1}{2}$-constraint instead. And if there are no such β-labeled edges at all, a 0-constraint. Note that peripheral constraints do not contain information about self-loops of the corresponding concrete nodes.

The byproduct of local abstraction is a mapping $h_{G,P}$. It maps nodes of G to corresponding nodes in P, if any. $h_{G,P}$ is not necessarily injective: If the abstraction contains a summary node, then all corresponding concrete nodes will be mapped to it.

Definition 10 (Local abstraction, induced mapping). *The* local abstraction *of a graph* G *with respect to a* focal node $v \in V_G$*, denoted by* $\alpha(G, v)$*, is the cluster* P *that satisfies the following conditions:*

- $V_P = h_{G,P}(V_G)$
- $E_P^\beta = h_{G,P}(E_G^\beta \cap ((\{v\} \times V_G) \cup (V_G \times \{v\})))$
- $S_P = \{u \in D_P \mid |{h_{G,P}}^{-1}(\{u\})| \geq 2\}$
- $C_P^\beta(u_1, u_2) = \begin{cases} 0 & : \forall v_1 \neq v_2 : (h_{G,P}(v_1), h_{G,P}(v_2)) = (u_1, u_2) \Rightarrow (v_1, v_2) \notin E_G^\beta \\ 1 & : \forall v_1 \neq v_2 : (h_{G,P}(v_1), h_{G,P}(v_2)) = (u_1, u_2) \Rightarrow (v_1, v_2) \in E_G^\beta \\ \frac{1}{2} & : \textit{else} \end{cases}$
- $\ell_P = {h_{G,P}}^{-1} \circ \ell_G$

where $h_{G,P} : V_G \rightharpoonup V_P$ *is the* induced mapping *of concrete nodes from* G *to abstract nodes in* P*, defined as*

$$h_{G,P} = \{(v, core)\} \cup \{(u, u') \in (V_G \setminus \{v\}) \times (\mathcal{SP} \times \mathcal{N}) \mid SP_G(v, u) \neq \emptyset \text{ and } u' = (SP_G(v, u), \ell_G(u))\},$$

The information order compares the information content of two peripheral constraints. It expresses that a $\frac{1}{2}$-constraint is less precise than both a 0 and a 1 constraint.

Definition 11 (Information order). *For* $l_1, l_2 \in \{0, 1, \frac{1}{2}\}$*, we write* $l_1 \sqsubseteq l_2$ *iff* $l_1 = l_2$ *or* $l_2 = \frac{1}{2}$*.*

Using information order, we define a partial order on clusters P and P' that considers P to be less than or equal to P' if P and P' are equal except for peripheral constraints, and each constraint of P is less than or equal (with respect to the information order) to the corresponding constraint of P'.

Definition 12 (Cluster order). *Let* P *and* P' *be clusters. We write* $P \sqsubseteq P'$ *iff* $G_P = G_{P'}$*,* $S_P = S_{P'}$ *and* $C^\beta_P(v, v') \sqsubseteq C^\beta_{P'}(v, v')$ *for any* $\beta \in \mathcal{E}$ *and* $v, v' \in V_P$*. We say that* P' *is an upper bound of* P*.*

Note that both information order and cluster order are partial orders, so the notion of least upper bounds is applicable to them. A least upper bound exists for clusters as long as they differ in peripheral constraints only. It yields a cluster with peripheral constraints that are just weak enough to be consistent with both clusters. In effect, a constraint becomes $\frac{1}{2}$ whenever it differs in the two clusters (or is already $\frac{1}{2}$).

Our abstract domain consists of sets of clusters, such that no pair of clusters is comparable according to the cluster order:

Definition 13 (Abstract topology). *An* abstract topology *is a set* $S \subseteq \mathcal{P}$*, where for no pair* $P_1 \neq P_2 \in S$ *there is a mutual upper bound* $P' \in \mathcal{P}$*.*

To obtain such an abstract topology from the clusters produced by local abstraction, we impose an order on sets of clusters, with an induced least upper bound. Cluster set S is less than or equal to cluster set S' according to this induced order iff for each cluster P in S, S' contains a cluster P', such that $P \sqsubseteq P'$ according to the cluster order.

Definition 14 (Cluster set order). *Let* S, S' *be sets of clusters. We write* $S \sqsubseteq S'$ *iff for each* $P \in S$*, there is a* $P' \in S'$ *such that* $P \sqsubseteq P'$*.*

We split the set of clusters into singleton sets, each containing one of the clusters. Then we consider the least upper bound over all of those singleton sets. This means joining any clusters that can be joined and taking the union for those that cannot. At the end, this yields the abstract topology we were looking for. We call this abstract topology the topologization of the cluster set under consideration.

Definition 15 (Topologization). *The* topologization *of a set of clusters $S \subseteq \mathcal{P}$ is the abstract topology $\bigsqcup S = \bigsqcup\{\{P\} \mid P \in S\}$).*

For each equivalence class of clusters from S identical except for peripheral constraints, topologization yields a single, joined, less precise cluster in the resulting topology. Note that we overload the $\bigsqcup$ operator, denoting topologization if applied to a set of clusters, and denoting the least upper bound on cluster sets if applied to sets of sets of clusters. Note further that, given $S \sqsubseteq S'$, we have $\bigsqcup S \sqsubseteq \bigsqcup S'$ and $S \sqsubseteq \bigsqcup S'$, but not necessarily $\bigsqcup S \sqsubseteq S'$.

The full abstraction of a graph is the topologization of the set of clusters obtained by local abstraction of each node of the graph. Each of these nodes corresponds to the core node of one of the clusters in the resulting abstract topology. Conversely, we define topology concretization.

Definition 16 (Cluster abstraction and concretization). *Let $\mathfrak{G} \subseteq \mathcal{G}$. Then the* cluster abstraction *of $\mathfrak{G}$ is the abstract topology $\alpha(\mathfrak{G}) = \bigsqcup\{\alpha(G, v) \mid v \in V_G \wedge G \in \mathfrak{G}\}$. An abstract topology S represents the set of concrete graphs $\gamma(S) = \{G \in \mathcal{G} \mid \alpha(\{G\}) \sqsubseteq S\}$.*

3.2 Abstract Transformer

Thus far, we considered how to apply rules on concrete graphs and how to abstract a graph into an abstract topology. Now, we discuss the application of rules on abstract topologies instead of concrete graphs. We obtain an abstract topology capturing the graphs we would obtain in the concrete. We sacrifice some precision in the abstract transformation to allow for a tractable and efficient implementation.

Rule application to all graphs from the cluster concretization induces an abstract derivation relation between clusters for a given rule and abstract topology. The relation holds if the core nodes of source and target cluster relate to corresponding nodes in the respective host and result graph.

Definition 17 (Induced abstract derivation). *The* induced abstract derivation *is a relation $\overset{r,S}{\Rightarrow} \subseteq \mathcal{P} \times \mathcal{P}$ where $P' \overset{r,S}{\Rightarrow} Q$ iff there are graphs G, H, a match $m : V_L \to V_G$ from r to G and a node $v \in V_G$, such that G is in the cluster concretization of S, $P \sqsubseteq P'$, $G \overset{r,m}{\rightsquigarrow} H$ and $\alpha(H, v) = Q$, where $r = (L, h, p, R)$, $P = \alpha(G, v)$ with induced mapping $h_{G,P} : V_G \to V_P$ and $m \circ h_{G,P} \neq \emptyset$.*

The induced abstract topology is the topology we obtain if we apply full abstraction to the initial graph and then iteratively compute abstract topologies until we reach a fixpoint: We apply any rule in any possible way to any graph from the cluster concretization of the abstract topology from the previous iteration, add the resulting clusters to those that already existed, and take the least upper bound on the cluster set thus obtained.

Definition 18 (Induced abstract topology). *The* induced abstract topology *is the set $[\![\mathcal{S}]\!]^\sharp = [\![\mathcal{S}]\!]^\sharp_n$ where $n = \min\{i \in \mathbb{N} \mid [\![\mathcal{S}]\!]^\sharp_i = [\![\mathcal{S}]\!]^\sharp_{i+1}\}$ and $[\![\mathcal{S}]\!]^\sharp_i$ defined recursively as follows:*

– $[\![\mathcal{S}]\!]^\sharp_0 = \alpha(\{I\})$
– $[\![\mathcal{S}]\!]^\sharp_i = \bigsqcup([\![\mathcal{S}]\!]^\sharp_{i-1} \cup \{Q \in \mathcal{P} \mid \exists P \in \mathcal{P}, r \in \mathcal{R} : P \overset{r,[\![\mathcal{S}]\!]^\sharp_{i-1}}{\Rightarrow} Q\})$

Note that the existence of the induced abstract topology follows from the fact that $[\![\mathcal{S}]\!]^\sharp_i \sqsubseteq [\![\mathcal{S}]\!]^\sharp_{i+1}$ and the finiteness of the domain.

Proposition 1. *The induced abstract topology overapproximates the graph transformation system semantics, i.e.,* $[\![\mathcal{S}]\!] \subseteq \gamma([\![\mathcal{S}]\!]^\sharp)$.

Induced abstract derivation, and, consequently, the induced abstract topology involves rule application to an infinite number of graphs. For an implementation, we need to reduce this to a finite number. That this is possible follows from the fact that our domain is finite.

We capture the characteristics of a sufficient, yet finite subset using the notion of abstract matches. While a concrete match relates a left hand side L of a rule to the nodes of a host graph G, the abstract match relates it to a cluster P. The core node of P has a corresponding node in a host graph G. This node has a corresponding focal node in the result graph H. (Recall that we do not permit node deletion.) Local abstraction on the result graph will yield the relevant cluster Q. Q primarily depends on P and the node and edge modifications as stipulated by the rule. Thus, the main components of an abstract match are P and the relation $h_{L,P}$ between the left hand side and the matched nodes of P.

However, indirectly, and perhaps contrary to intuition, Q also depends on some nodes and edges of the host graph G that are *neither* matched *nor* determined by P:

– For each match to a summary node, only one concrete instance will be matched. Thus, Q may depend on the number of additional unmatched instances (captured by *mater* in the following definition). We need to distinguish only the cases of zero, one, and more than one instances, since the latter will always become a summary node after abstraction.
– A $\frac{1}{2}$-constraint in P may become a 0 and 1 constraint in Q, and sometimes remain as is: (a) If two matched peripheral nodes have an unmatched $\frac{1}{2}$-constraint in between, the corresponding concrete edge will be either present or absent in G, captured by *cc*. (b) The concrete edge corresponding to a $\frac{1}{2}$-constraint between a pair of unmatched peripheral nodes will be either present or absent in G. Concrete edges incident to residual materializations of a summary node v with $mater(v) \geq 1$ may be present for all, none or some of the corresponding concrete node pairs. Both cases are captured by *dd*. (c) The same possibilities exist for edges between an unmatched peripheral node and a matched node. The mapping *dc* specifies these edges. In this case, the matched node does not even have to be in P, since it might just be about to become connected to the focal node through application of the rule.

In addition, the match requires that a closure exists, that is, we have a graph G from the cluster concretization for which the match holds.

Definition 19 (Abstract match). *Let $r = (L, h, p, R)$ be a rule and S be an abstract topology. An abstract match from r to S is a tuple $(P, h_{L,P}, mater, dc, cd, dd)$ where*

- *$P \in \mathcal{P}$ is the matched cluster,*
- *$h_{L,P} : V_L \rightharpoonup V_P$ maps the left hand side to the nodes of P,*
- *$mater : D_P \to \{0, 1, 2\}$ specifies the residual materialization count of summary nodes in P,*
- *$dc : (V_L \times D_P \times \{-1, 1\} \times \mathcal{E}) \to \{0, 1, \frac{1}{2}\}$ specifies the materialization of edges from peripheral to matched nodes and vice versa,*
- *$dd : (D_P \times D_P \times \mathcal{E}) \to \{0, 1, \frac{1}{2}\}$ specifies the peripheral edge materialization*
- *$cc : V_L \times V_L \to 2^{\mathcal{E}}$ specifies the materialization of edges among matched nodes*

and the following conditions are satisfied:

- *$P \sqsubseteq P'$ for some $P' \in S$*
- *$h_{L,P}(V_L) \neq \emptyset$*
- *$|{h_{L,P}}^{-1}(core)| < 2$*
- *the following conditions hold for $matched : D_P \to \mathbb{N}$, the induced number of matches, defined as $matched(u) := |{h_{L,P}}^{-1}(\{u\})|$:*

$$matched(u) = 0 \Rightarrow mater(u) = \begin{cases} 2 & \text{if } u \in S_P \\ 1 & \text{otherwise} \end{cases}$$

$$matched(u) = 1 \Rightarrow mater(u) \in \begin{cases} \{1, 2\} & \text{if } u \in S_P \\ \{0\} & \text{otherwise} \end{cases}$$

$$matched(u) > 1 \Rightarrow u \in S_{P_v}$$

- *there is a graph G, a match $m : V_L \to V_G$ from r to G and a node $v \in V_G$ such that $\alpha(G, v) = P$ with induced mapping $h_{G,P} : V_G \rightharpoonup V_P$ and*
  - *$m \circ h_{G,P} = h_{L,P}$,*
  - *$mater(u) = \min\{|{h_{G,P}}^{-1}(\{u\}) \setminus m(V_L)|, 2\}$,*
  - *for all $u \in V_L \setminus m^{-1}(\{v\})$, $u' \in D_P$, $\beta \in \mathcal{E}$, and $d \in \{-1, 1\}$,*

$$dc(u, u', d, \beta) = \begin{cases} 0 & : \forall v' \notin m(V_L) : h_{G,P}(v') = u' \Rightarrow (m(u), v') \in (E_G^\beta)^d \\ 1 & : \forall v' \notin m(V_L) : h_{G,P}(v') = u' \Rightarrow (m(u), v') \notin (E_G^\beta)^d \\ \frac{1}{2} & \text{otherwise,} \end{cases}$$

  - *for all $u, u' \in D_P$, for all $\beta \in \mathcal{E}$,*

$$dd(u, u', \beta) = \begin{cases} 0 & : \forall v_1 \neq v_2 \notin m(V_L) : (h_{G,P}(v_1), h_{G,P}(v_2)) = (u, u') \\ & \qquad \Rightarrow (v_1, v_2) \in E_G^\beta \\ 1 & : \forall v_1 \neq v_2 \notin m(V_L) : (h_{G,P}(v_1), h_{G,P}(v_2)) = (u, u') \\ & \qquad \Rightarrow (v_1, v_2) \notin E_G^\beta \\ \frac{1}{2} & \text{otherwise,} \end{cases}$$

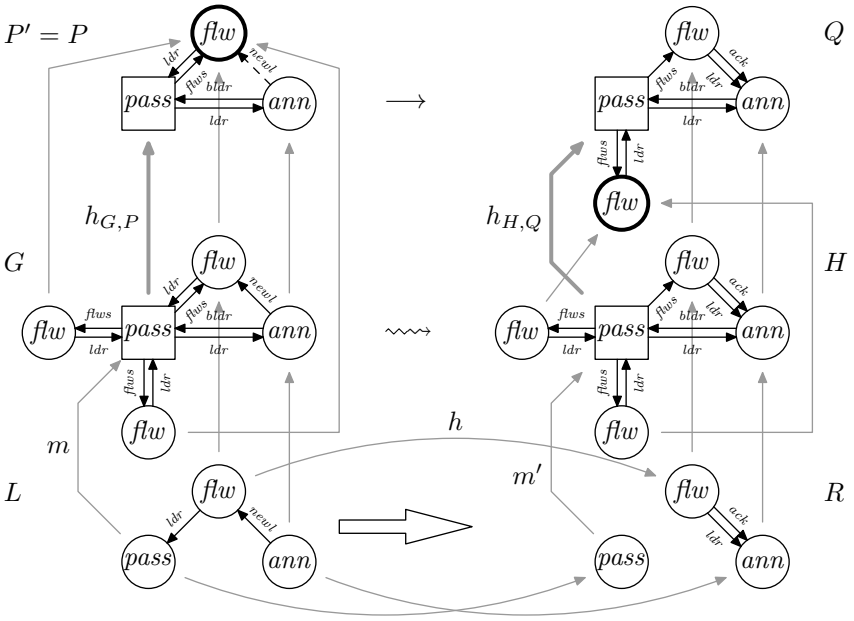

Fig. 3. An example of the abstract transformer.

- *for all* $u, u' \in V_L$,

$$(cc(u,u'), cc(u',u)) = SP_G(m(u), m(u')),$$

- $G \in \gamma(S)$, *and*
- *the partner constraints* p *are satisfied.*

Since the number of abstract matches is finite, the definition is constructive and a computation method directly follows from it, except for the non-trivial closure check. However, the fact that we are looking for an overapproximation allows us to weaken this check, including the option to ignore it completely. This includes the check that partner constraints are satisfied. Note that at least the partner constraints for ${h_{G,P}}^{-1}(\mathit{core})$ can be checked without knowledge of the entire graph G.

From the abstract matches, we generate partial concretizations. These are graphs with focal node and neighborhood, just sufficient to capture all potential changes to the cluster caused by rule application and local abstraction of the result. We do not need to consider the full graph, since this is taken care of by symmetry: The additional nodes it contains will be covered by other abstract matches with those nodes as the core node of a cluster. Those, in turn, have their own partial concretizations to account for the impact of the rule application.

Note that edges specified by *dd* and *dc* will never be modified by a rule, for that would require its adjacent nodes both to be matched, which is, by definition, not the case. The set A, in the following definition, splits the unmatched peripheral nodes into two subsets such that those in the set will have respective edges for the $\frac{1}{2}$ case and the complementary nodes will not.

Definition 20 (Partial concretization). *The* partial concretization *function γ maps abstract matches $(P, h_{L,P}, mater, dc, dd, cc)$ to tuples $(G, m, h_{G,P})$ where G is a graph, $m : V_L \rightharpoonup V_G$ is an injective partial graph morphism from the left hand side to this graph and $h_{G,P} : V_G \rightharpoonup V_P$ is a mapping to the abstraction P, all defined as follows:*

- $V_G = \{core\} \cup (V_L \setminus {h_{L,P}}^{-1}(\{core\})) \cup \{(u, n) \in D_P \times \mathbb{N} \mid 1 \leq n \leq mater(u)\}$
- $h_{G,P}(u) = \begin{cases} core & \text{if } u = core \\ v & \text{if } u = (v, n) \\ h_{L,P}(u) & \text{if } u \in V_L \setminus {h_{L,P}}^{-1}(\{core\}) \end{cases}$
- $m = \{({h_{L,P}}^{-1}(core), core)\} \cup \{(u, u) \mid u \in (V_L \setminus {h_{L,P}}^{-1}(\{core\}))\}$
- $E_G^\beta = \{(u, u') \in A \times A \mid dd(h_{G,P}(u), h_{G,P}(u'), \beta) \geq \frac{1}{2}\}$
  $\cup \{(u, u') \in V_G \times V_G \mid dd(h_{G,P}(u), h_{G,P}(u'), \beta) = 1\}$
  $\cup \{(u, u') \in m(V_L) \times m(V_L) \mid \beta \in cc(m^{-1}(u), m^{-1}(u'))\}$
  $\cup \{(u, u') \in m(V_L) \times A \mid dc(m^{-1}(u), h_{G,P}(u'), 1, \beta) = \frac{1}{2}\}$
  $\cup \{(u, u') \in m(V_L) \times V_G \mid dc(m^{-1}(u), h_{G,P}(u'), 1, \beta) = 1\}$
  $\cup \{(u, u') \in A \times m(V_L) \mid dc(m^{-1}(u'), h_{G,P}(u), -1, \beta) = \frac{1}{2}\}$
  $\cup \{(u, u') \in V_G \times m(V_L) \mid dc(m^{-1}(u'), h_{G,P}(u), -1, \beta) = 1\}$
  $\cup {h_{G,P}}^{-1}(E_P^\beta)$
  where $A = (D_P \times \{1\}) \cap V_G$
- $\ell_G = (m^{-1} \circ \ell_L) \cup (h_{G,P} \circ \ell_P)$

The abstract transformer describes how clusters are affected by rule application. It presupposes the existence of an abstract match, constructs the corresponding partial concretization, applies the rule, and constructs the modified cluster by local abstraction of the focal node. See Figure 3.2 for an example.

Definition 21 (Abstract transformer). *Let $r = (L, h, p, R)$ be a rule and S be an abstract topology. The* abstract transformer *(or* direct derivation*) is a relation $\overset{r,S}{\to} \subseteq \mathcal{P} \times \mathcal{P}$ where $P' \overset{r,S}{\to} Q$ iff there is a graph H and an abstract match $\hat{m} = (P, h_{L,P}, mater, dc, dd, cc)$ from r to S such that $P \sqsubseteq P'$, $\gamma(\hat{m}) = (G, m, h_{G,P})$, $G \overset{r,m}{\rightsquigarrow} H$ and $Q = \alpha(H, core)$*

The graph morphism m may be partial, i.e., some nodes of the left hand side may map to none of the nodes in G. Not even the focal node needs to be covered. In those cases, we waive the totality requirement that rule application puts on m, thereby modifying only those parts of the partial concretization that are matched. We obtain an abstract topology that overapproximates the system by abstracting the start graph and applying the abstract transformer in a fixpoint iteration.

Definition 22 (Derived abstract topology). *The* derived abstract topology *is the set $[\mathcal{S}]^\sharp = [\mathcal{S}]^\sharp_n$, where $n = \min\{i \in \mathbb{N} \mid [\mathcal{S}]^\sharp_i = [\mathcal{S}]^\sharp_{i+1}\}$ and $[\mathcal{S}]^\sharp_i$ is defined recursively as follows:*

– $[\mathcal{S}]_0^\sharp = \alpha(\{I\})$
– $[\mathcal{S}]_i^\sharp = \bigsqcup([\mathcal{S}]_{i-1}^\sharp \cup \{Q \in \mathcal{P} \mid \exists P \in [\mathcal{S}]_{i-1}^\sharp, r \in \mathcal{R} : P \overset{r,[\mathcal{S}]_{i-1}^\sharp}{\rightarrow} Q\}$
$\cup \{\alpha(\{R\}) \mid (\emptyset, \emptyset, \emptyset, R) \in \mathcal{R}\})$

Note that we assumed the absence of rules with non-empty left hand side that create new nodes. Because of this, we do not need to take care of new clusters that occur as a byproduct of the modification of an existing cluster. Instead, for each rule with empty left hand side, we add the clusters obtained by local abstraction for each right hand side node. This takes place unconditionally, pointing towards the equivalence of node creation and initial graphs in our domain.

Theorem 1. *The derived abstract topology overapproximates the induced abstract topology, i.e.,* $[\![\mathcal{S}]\!]^\sharp \sqsubseteq [\mathcal{S}]^\sharp$.

Corollary 1 (Soundness). *The derived abstract topology overapproximates the graph transformation system semantics, i.e.,* $[\![\mathcal{S}]\!] \subseteq \gamma([\mathcal{S}]^\sharp)$.

Proof. This follows immediately from Proposition 1, Theorem 1, and the monotonicity of cluster concretization.

4 Experimental Evaluation

4.1 Implementation

We implemented cluster abstraction in our tool `astra` 2.0. The implementation differs from theory in minor respects: (a) Partial concretization materializes clusters over the entire left hand side of a rule at once, exploiting symmetry and allowing us to properly check all partner constraints. (b) We do a rudimentary check for the existence of a closure, by checking whether peripheral constraints of unmatched nodes are satisfiable. (c) To cover cases with unmatched core nodes, for each match, we iterate over all possibilities in which one additional cluster can be attached in the periphery. (d) After each iteration, we apply a reduction step, eliminating any cluster whose existence can be ruled out easily, and concretizing $\frac{1}{2}$-constraints if more precise information is available. (e) In various places, we use overapproximation ad hoc in order to improve analysis time.

4.2 Selection of Benchmarks

With `astra` 1.0, we already succeeded to analyze a part of the *merge protocol* [1] with star abstraction [3], a precursor to the method described in this paper. (In a nutshell, cluster abstraction with all peripheral constraints being $\frac{1}{2}$.) It was sufficient to analyze platoon formation and car joining, but not platoon merging, for which state space explosion occurred: Follower handover requires ternary predicates, while star abstraction only preserves binary predicates. This causes a cascade of spurious abstract states, with the analysis eventually spending its time enumerating an intractable number of combinatorial possibilities. The main goal of `astra` 2.0 was to analyze the full protocol. We did this for two

Table 1. Benchmark analysis statistics. cl. = clusters, a.r. = active rules, i.e., applied at least once, m. = abstract matches, rule app. = rule applications, it. = iterations, vfy. = safety property verified. *safety property not expressible as forbidden subgraphs

Benchmark	# cl.	# a.r.	# m.	# rule app.	# it.	time	vfy.
Synchronous merge	873	34	9674	349774	17	0m 14.057s	**yes**
Asynchronous merge	3069	36	44553	36114603	21	14m 27.977s	**yes**
AVL trees	1876	302	114284	2221151967	38	757m 9.273s	**yes**
Firewall	31	4	139	1371	5	0m 0.012s	**yes**
Firewall 2	96	9	786	45525	7	0m 0.330s	no
Public/private servers 2	239	26	1633	102250	10	0m 1.030s	**yes**
Dining philosophers	41	8	40	179	7	0m 0.006s	no*
Resources	32	7	100	207	4	0m 0.007s	**yes**
Mutual exclusion	308	9	2419	1237361	17	0m 56.060s	**yes**
Red-black trees	263	38	8769	24855500	11	10m 3.145s	**yes**
Singly-linked lists	7	2	15	13	3	0m 0.000s	**yes**
Circular buffers	152	2	798	241234	17	2m 43.441s	no*
Euler walks	18	6	47	134	3	0m 0.008s	no*

versions. In addition, we analyzed the *AVL tree* benchmark from [4] and various other benchmarks from the related work: *Firewall, public/private servers, dining philosophers, resources, mutual exclusion*, and *red-black trees* are benchmarks from the AUGUR package [5]; *singly-linked lists, circular buffers* and *Euler walks* for GROOVE are from [6].

The AUGUR package comes with additional benchmarks that we did not analyze: *connections, leader election protocol* and the *Needham–Schroeder protocol* all make use of numerical attributes, which are not yet supported by our tool. *External-internal processes* is merely a stripped-down version of *public and private server 2*. *Public and private server* contains a subset of the rules from *public and private server 2*. The same holds for the finite-state version of *dining philosophers* versus the infinite-state version, which we analyze. *Red-black trees converted* is a tweaked version of *red-black trees* to ease analysis with AUGUR.

We could analyze the GROOVE benchmarks without modifications. The AUGUR benchmarks, on the other hand, had to be translated from the tool's hyperedge-based approach to one based on nodes and edges. In addition, we had to make a structure-preserving change to the *public/private server* grammar (replacing a specific edge with two edges connected by a node) in order to prevent combinatorial explosion that would otherwise have defied analysis. For *red-black* and *AVL trees*, we manually added invariants about the uniqueness of some labels over the entire graph. These invariants trivially follow from the respective graph transformation systems and it would in principle be easy to find them automatically. However, uniqueness is not expressible in our abstraction, because clusters always represent an arbitrary number of concrete instances.

We checked the safety properties by adding rules specifying respective forbidden subgraphs, producing a node with an error label if found. This approach

could not be taken for *dining philosophers*, *circular buffers* and *Euler walks*, since the respective safety properties quantify over an unbounded number of nodes and hence cannot be formulated as forbidden subgraphs.

4.3 Analysis Results

`astra` was able to analyze all benchmarks. See Table 1 for the number of iterations required for reaching the fixed point, the number of clusters in the final result and the processor time taken. We ran all analyses on an Intel Core 2 Quad CPU Q9550 (2.83GHz) with 4 GB of memory under Linux 3.15, though only 9 MB were used at the peak for the largest benchmark, *asynchronous merge*. Execution time given is the time in user mode as reported by `time`(1).

In all but one of the cases with safety properties expressible as forbidden subgraphs, verification succeeded. Verification failed for *firewall 2* because the abstraction was unable to distinguish locations in front of and behind the firewall.

5 Related Work

Petri graphs are unfoldings of graph transformation systems, abstracted by a cutoff after a defined depth [7]. Reachability can be checked with existing techniques for Petri nets. As we have seen, we were able to analyze a subset of their benchmarks. Once they support negative application conditions (which they currently list as future work), it will be interesting to investigate whether their tool AUGUR [8] can analyze our main target, the merge protocol.

Bauer et al.'s partner abstraction [9] considers connected components instead of overlapping clusters and folds nodes according to neighborhood node and edge labels. In practice, it requires the system to obey friendliness properties that hold only for a simplified merge protocol where processes know each other's state [4]. Rensink and Distefano [10] consider an abstraction similar in design and limitations. Ideas from both approaches were combined and extended in neighborhood abstraction [11]. No friendliness restriction applies, but lacking Bauer's decomposition into components, the GROOVE implementation runs out of memory even on Bauer's simplified merge protocol [6].

Environment abstraction [12] abstracts a system into one process and its environment, i.e., the set of states of all the other processes plus relations to them. Cherem and Rugina [13] propose a local abstraction for shape analysis that tracks individual heap cells and their immediate neighborhood. Bauer et al.'s daisy patterns [14] and our star abstraction [3] are graph abstractions based on the same idea, the former abstracting the transformation rules in addition to the graph. All these abstractions are less precise than cluster abstraction, since none of them tracks peripheral node relationships.

Saksena et al. [15] verify graph transformation systems by symbolic backward reachability analysis. Starting with the undesirable configurations, they compute, by backward rule application in a fixed point iteration, an overapproximation of the set of reachable predecessor configurations, checking whether an initial configuration is among them. While not guaranteed to terminate, their method succeeds in proving loop freedom of an ad hoc routing protocol.

Berdine et al. [16] show that shape analysis of concurrent programs via canonical abstraction [17] leads to state-space explosion even for a toy example. The complexity of expressing cluster abstraction via canonical abstraction confirms this: at least, one abstraction predicate would be needed for each spoke, which is exponential in the number of edge labels. Berdine et al.'s own solution allows efficient analysis of an unbounded number of threads manipulating an unbounded shared heap. However, their abstraction is unable to express direct relations between the state of the threads. Manevich et al. [18] decompose the heap into a bounded number of overlapping components as specified by user-defined location selection predicates. In contrast, our method decomposes the graph by local abstraction of each of the unbounded number of nodes.

Zufferey et al. [19] provide an abstraction for depth-bounded systems (systems with a bound on the longest acyclic path), an expressive class of well-structured transition systems. Unfortunately, the merge protocol does not belong to this class unless one uses a simplified version similar to Bauer's.

6 Conclusions and Future Work

We have seen an abstraction for the analysis of the set of reachable graphs generated by infinite-state graph transformation systems. Using `astra`, our implementation of *cluster abstraction*, we were for the first time able to analyze the full merge protocol. In addition, our method has proven robust and precise enough to allow for the analysis of various benchmarks from the literature.

Future work: (1) We are going to check safety properties that cannot be expressed as forbidden subgraphs, such as quantification over an unbounded number of nodes. (2) We shall explore suitable approximations for the closure check, to preserve more of the global graph structure during rule application. (3) We are going to investigate opportunities to adjust the precision of our analysis. Especially, structure-preserving changes to the graph transformation system before the analysis seem to be an interesting way to give direction to the abstraction. For example, adding edges to the right hand side of rules with a new label that never occurs on a left hand side can keep nodes in the periphery of some clusters, thereby increasing precision. (4) If some cluster may occur at most once, we would like to retain this information. (5) We would like to allow integer values as node and edge attributes, in addition to regular labels. Lifted to the abstraction, it extends clusters by overapproximated values for those attributes, based on abstract domains on integers. (6) Based on a suitable fragment of μ-calculus, we plan to support abstract model checking on an abstract labeled transition systems of clusters, preserving some non-trivial relationships for the transitions, such as size invariants on summary nodes. We plan to extend this to model checking over an abstract labeled transition system, based on a suitable fragment of μ-calculus.

Acknowledgments. We thank Reinhard Wilhelm for many valuable discussions about this work and the anonymous referees for their useful comments.

References

1. Backes, P., Reineke, J.: A graph transformation case study for the topology analysis of dynamic communication systems. In: TTC'10. Volume WP10-03 of CTIT Workshop Proceedings., Enschede, University of Twente (2010) 107–118
2. Hsu, A., Eskafi, F., Sachs, S., Varaiya, P.: Design of platoon maneuver protocols for IVHS. Technical report, Institute of Transportation Studies, UC Berkeley (1991)
3. Backes, P., Reineke, J.: Abstract topology analysis of the join phase of the merge protocol [using astra]. In: TTC'10. Volume WP10-03 of CTIT Workshop Proceedings., Enschede, University of Twente (2010) 127–133
4. Backes, P.: Topology analysis of dynamic communication systems. Diploma thesis, Saarland University (March 2008)
5. Kozyura, V., König, B.: Augur 2—A tool for the analysis of (attributed) graph transformation systems using approximative unfolding techniques. (April 2008)
6. Zambon, E.: Abstract graph transformation : theory and practice. PhD thesis, University of Twente (2013)
7. Baldan, P., König, B.: Approximating the behaviour of graph transformation systems. In Corradini, A., Ehrig, H., Kreowski, H.J., Rozenberg, G., eds.: ICGT'02. Volume 2505 of LNCS. (January 2002) 14–29
8. König, B., Kozioura, V.: Augur 2—a new version of a tool for the analysis of graph transformation systems. In Bruni, R., Varró, D., eds.: GT-VMT'06. Volume 2011 of ENTCS. (2008) 201–210
9. Bauer, J., Wilhelm, R.: Static analysis of dynamic communication systems by partner abstraction. In Nielson, H.R., Filé, G., eds.: SAS'07. Volume 4634 of LNCS. (2007) 249–264
10. Rensink, A., Distefano, D.: Abstract graph transformation. In: SVV'05. Volume 157 of ENTCS. (May 2006) 39–59
11. Boneva, I., Kreiker, J., Kurbán, M., Rensink, A., Zambon, E.: Graph abstraction and abstract graph transformations (amended version). Technical Report TR-CTIT-12-26, University of Twente, Enschede, the Netherlands (October 2012)
12. Clarke, E., Talupur, M., Veith, H.: Environment abstraction for parameterized verification. In Emerson, E.A., Namjoshi, K.S., eds.: VMCAI'06. Volume 3855 of LNCS. (2006) 126–141
13. Cherem, S., Rugina, R.: Maintaining doubly-linked list invariants in shape analysis with local reasoning. In Cook, B., Podelski, A., eds.: VMCAI'07. Volume 4349 of LNCS. (2007) 234–250
14. Bauer, J., Boneva, I., Rensink, A.: Graph abstraction by daisy patterns. Privately circulated (May 2009)
15. Saksena, M., Wibling, O., Jonsson, B.: Graph grammar modeling and verification of ad hoc routing protocols. In Ramakrishnan, C.R., Rehof, J., eds.: TACAS'08. Volume 4963 of LNCS. (2008) 18–32
16. Berdine, J., Lev-Ami, T., Manevich, R., Ramalingam, G., Sagiv, M.: Thread quantification for concurrent shape analysis. In Gupta, A., Malik, S., eds.: CAV'08. Volume 5123 of LNCS. (2008) 399–413
17. Sagiv, M., Reps, T., Wilhelm, R.: Parametric shape analysis via 3-valued logic. ACM Trans. Program. Lang. Syst. **24**(3) (May 2002) 217–298
18. Manevich, R., Lev-Ami, T., Sagiv, M., Ramalingam, G., Berdine, J.: Heap decomposition for concurrent shape analysis. In Alpuente, M., Vidal, G., eds.: SAS'08. Volume 5079 of LNCS. (2008) 363–377
19. Zufferey, D., Wies, T., Henzinger, T.A.: Ideal abstractions for well-structured transition systems. In Kuncak, V., Rybalchenko, A., eds.: VMCAI'12. Volume 7148 of LNCS. (January 2012) 445–460