

A Modal-Logic Based Graph Abstraction

Jörg Bauer², Iovka Boneva¹, Marcos E. Kurbán³, and Arend Rensink¹

¹ Institut für Informatik, Technical University of Munich,
85748 Garching bei München, Germany
`joba@model.in.tum.de`

² Formal Methods and Tools Group, EWI-INF, University of Twente
PO Box 217, 7500 AE, Enschede, The Netherlands
`{bonevai,rensink}@cs.utwente.nl`

³ Former member of Formal Methods and Tools Group,
EWI-INF, University of Twente

Abstract. Infinite or very large state spaces often prohibit the successful verification of graph transformation systems. Abstract graph transformation is an approach that tackles this problem by abstracting graphs to abstract graphs of bounded size and by lifting application of productions to abstract graphs. In this work, we present a new framework of abstractions unifying and generalising existing takes on abstract graph transformation. The precision of the abstraction can be adjusted according to the properties to be verified facilitating abstraction refinement. We present a modal logic defined on graphs, which is preserved *and* reflected by our abstractions. Finally, we demonstrate the usability of the framework by verifying a graph transformation model of a firewall.

1 Introduction

Formal verification of graph transformation systems aims at statically proving or inferring properties of a graph transformation system, where such properties are typically given in some form of temporal logic. It is crucial to distinguish verification and simulation, the latter being very useful only for debugging, whereas verification establishes a property for *all* computations of a graph transformation system. Problems do arise when approaching this task. One such problem is the possibly infinite behaviour of a system which in most cases makes it impossible to study the whole behaviour of the system. Another problem is space: even for a finite state space, each state can be quite big to represent.

Some approaches to formal verification of graph transformation systems include [1,2,3,4,5,6,7,8]. They can be characterised as to which approach to graph transformation is used for modelling, which verification technique is applied, and which applications are tackled. The technique presented in [1] feeds finite-state graph transformation systems, given as a double pushout system, to an off-the-shelf model checker to verify reactive systems. However, we face the more general problem of unbounded systems. The approaches presented in [2,3] both use backwards reachability analysis for hyperedge replacement grammars trying to reach an initial graph by backwards search from a forbidden configuration.

The technique is applied to mechatronic systems and ad-hoc network routing, respectively, but, unfortunately, is not guaranteed to terminate. An approximation of the behaviour of a graph transformation system in terms of Petri net unfoldings was used in [4] to verify properties of data structures residing in the run-time heap of programs with dynamically allocated heap memory.

In this work, we present a new take on *abstract graph transformation* as introduced independently by [6] and [7]. Abstract graph transformation relies on *abstract interpretation* [9] of graph transformation systems, that is, given some equivalence relation, graphs are quotiented into abstract graphs of bounded, finite size. Application of productions is then lifted to work on abstract graphs. The abstraction first introduced in [10] summarises nodes with similar kind and number of incident edges, while the abstraction of [7] considers similar adjacent nodes. These two abstractions are generalised in this work and put into a unifying framework. To this end, we introduce the notion of *neighbourhood abstraction* as a part of a general abstraction framework. For this abstraction, nodes are summarised if they have similar neighbourhood up to some *radius i* , parameter of the abstraction. This enables abstractions with different precisions. Additionally, the number of possible abstract graphs obtained by neighbourhood abstraction is bounded. We introduce a logic accompanying our abstractions: given a formula our abstraction guarantees that *a*) if the formula holds for the original graph, then it holds for the abstracted graph (preservation); and *b*) if the formula holds for the abstracted graph, then it holds for the original one too (reflection).

Contributions

- Our abstraction framework unifies and generalises previous approaches on abstract graph transformation. For this particular technique, it supposedly establishes the most general treatment of *local* abstractions, that is, abstractions based on equivalence relations, where equivalence is determined by local information on nodes. In contrast, equivalences used in shape analysis [11] of heap programs often consider global properties like reachability.
- Technically, the most surprising result comprises the definition of a modal logic, properties of which are preserved *and* reflected by our abstraction. While preservation is necessary for the soundness of analyses based on our abstraction, reflection is a rather unusual and strong result.
- Our framework allows for automated *abstraction refinement*. If a property cannot be established given a certain neighbourhood size, one may automatically increase this size to obtain more precise results. The only other approach allowing for automated refinement is [5].
- A *canonical representation* of abstract graphs reduces otherwise costly isomorphism checks to simple equality tests.
- While, certainly, our method has its limitations, it works well for an important class of systems, dynamic communication systems. They are characterised by a dynamically changing number of communicating objects and a dynamically changing communication topology. Important examples include ad-hoc network protocols, traffic control- or mechatronic systems. Our

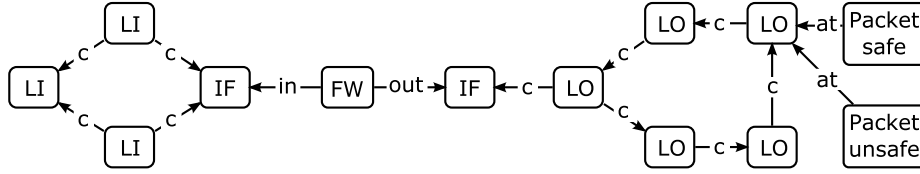


Fig. 1. Two networks delimited by a firewall

method is not suited for the analysis of graphs occurring in runtime heaps. The latter almost always require reachability information to be taken into account, something our approach fails to handle satisfactorily.

Outline. To start with, we shall present our case study, a firewall system, in Sect. 1.1. Section 2 introduces graphs and the general abstraction mechanism, as well as so called neighbourhood abstraction. In Sect. 3, we present a modal logic that is preserved *and* reflected by neighbourhood abstraction. In Sect. 4, a canonical representation of abstract graphs obtained by neighbourhood abstraction is defined, which is crucial for the representation of graphs in the actual implementation of the transformation. Before we conclude in Sect. 6, we briefly describe, in Sect. 5, how all ingredients can be combined for defining a fully automatic method for system verification.

1.1 Case Study: Firewalls

Figure 1 shows a graph model of two networks delimited by a firewall (FW). It has an internal and an external interface (IF), connected respectively to a network of *in*-locations (LI) to be protected by the firewall, and a network of *out*-locations (LO). Arbitrarily many packets flowing through the network can be created at locations – *safe* ones at any location and *unsafe* ones only at out-locations. The flow is bi-directional despite the drawing of directed *c*-edges. The full set of rules implementing such a firewall are given in [12]. A property we want to verify is that unsafe packets never reach in-locations.

2 Graphs and Graph Abstraction

We consider finite graphs whose edges are labelled from a finite set of labels, **Lab**. We mimic node labels by labelling special edges whose target is a special node \perp . Formally, a *graph* G is a tuple $(N_G, E_G, \text{src}_G, \text{tgt}_G, \text{lab}_G)$ where N_G is a finite set of nodes, E_G is a finite set of edges disjoint from N_G , $\text{src}_G : E_G \rightarrow N_G$ and $\text{tgt}_G : E_G \rightarrow N_G \cup \{\perp\}$ with $\perp \notin (N_G \cup E_G)$ associate with each edge its source and target nodes, and $\text{lab}_G : E_G \rightarrow \mathbf{Lab}$ labels edges. Let G and H be graphs. A *graph morphism* $f : G \rightarrow H$ is a function from $N_G \cup E_G \cup \{\perp\}$ to $N_H \cup E_H \cup \{\perp\}$ such that $f(\perp) = \perp$ and $f^{-1}(\perp) = \{\perp\}$; f maps nodes to nodes and edges to edges, i.e. $f(N_G) \subseteq N_H$, $f(E_G) \subseteq E_H$; f is compatible with source and target mappings, i.e. $\text{src}_H \circ f = f \circ \text{src}_G$, and $\text{tgt}_H \circ f = f \circ \text{tgt}_G$,

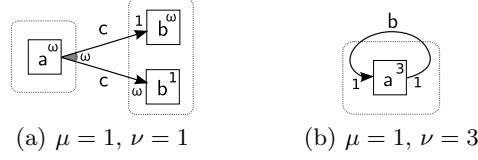


Fig. 2. Examples of abstract graphs

and f preserves labels, $f \circ \text{lab}_G = \text{lab}_H$. A morphism f is called *injective* (resp. *surjective*, resp. *bijective*) if it defines an injective (resp. surjective, resp. bijective) map. A bijective morphism is also called an *isomorphism*. We extend lab_G to a node to determine its set of labels, i.e. $\text{lab}_G(v) = \{\mathbf{a} \in \text{Lab} \mid \exists e \in E_G : \text{src}_G(e) = v, \text{tgt}_G(e) = \perp, \text{lab}_G(e) = \mathbf{a}\}$. We write $v \triangleright_G^{\mathbf{a}}$ and $v \triangleleft_G^{\mathbf{a}}$ for the set of \mathbf{a} -outgoing edges and \mathbf{a} -incoming edges of node v , respectively, i.e. $v \triangleright_G^{\mathbf{a}} = \{e \in E_G \mid \text{src}_G(e) = v, \text{lab}_G(e) = \mathbf{a}\}$ and symmetrically for $v \triangleleft_G^{\mathbf{a}}$. For a set of nodes V , $V \triangleright_G^{\mathbf{a}}$ (resp. $V \triangleleft_G^{\mathbf{a}}$) is the extension of $\triangleright_G^{\mathbf{a}}$ (resp. $\triangleleft_G^{\mathbf{a}}$) on sets. Finally, for X, Y nodes or sets of nodes, we denote $X \triangleright \triangleright_G^{\mathbf{a}} Y$ the set of \mathbf{a} -labelled edges between X and Y , i.e. $X \triangleright \triangleright_G^{\mathbf{a}} Y = X \triangleright_G^{\mathbf{a}} \cap Y \triangleleft_G^{\mathbf{a}}$. When the graph G is clear from the context, we may omit the subscript G . For brevity, in the sequel of the paper we ignore the node \perp and simply talk about node labels.

A *multiplicity* approximates the cardinality of a finite set. For any natural $\mu > 0$, let \mathbf{M}_μ be the set $\{0, 1, 2, \dots, \mu, \omega\}$ where $\omega \notin \mathbb{N}$. The μ -multiplicity of a set U is denoted $|U|_\mu$ and defined by: $|U|_\mu = \text{Card}(U)$ if $\text{Card}(U) \leq \mu$, and $|U|_\mu = \omega$ otherwise. We write \mathbf{M}_μ^+ for the set $\mathbf{M}_\mu \setminus \{0\}$. The usual ordering \geq is extended to elements of \mathbf{M}_μ by $\omega \geq \lambda$ for all λ in \mathbf{M}_μ . Sums over multiplicities are defined as expected writing \sum^μ for μ -bounded sums. For the sequel, we fix two naturals, $\nu, \mu > 0$, to denote multiplicities of sets of nodes (ν) and sets of edges (μ), i.e. ν and μ are parameters of graph abstractions.

2.1 Abstract Graphs and Abstraction

In this section, we discuss the notion of *abstract graphs*. Abstract graphs, such as the ones of Fig. 2, represent sets of (concrete) graphs called their *concretisations*. Every node of an abstract graph is associated with a *node multiplicity* indicating the number of concrete graph nodes it represents. The dotted rectangles are delimiting groups of nodes induced by an equivalence relation, the *grouping relation*, on them. All edges have associated multiplicity information (from \mathbf{M}_μ) in their end points: *outgoing edges multiplicity*, when associated to the source of the edge, and *incoming edges multiplicity* when associated to the target. Sometimes, this multiplicity is shared by several edges, indicated by the grey arc relating them. Edge multiplicities indicate how many of the depicted edges should be there in a concretisation. Note that edges related in one of their end points all have their other end point in the same group of nodes, and all have the same label. Actually, this is the condition for relating edges. More precisely, edge multiplicities are associated with a triple composed of a node, a label and a group of nodes.

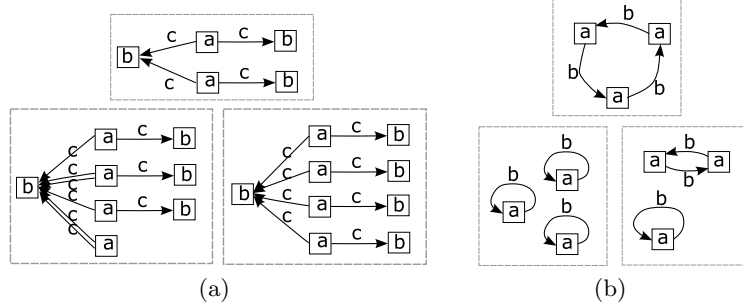


Fig. 3. Example concretisations of the abstract graphs on Fig. 2

Consider the abstract graph of Fig. 2(a). It represents a set of bipartite concrete graphs, such as the ones of Fig. 3(a), where **a**-nodes are connected to **b**-nodes by **c**-edges. Each of these graphs has at least two (as $\nu = 1$, ω stands for “two or more”) **a**-nodes and at least three (ω plus one) **b**-nodes. Moreover, every **a**-node has at least two (*i.e.* ω) outgoing **c**-edges going to **b**-nodes. All **b**-nodes except one have only one incoming edge; the remaining **b**-node has at least two incoming edges. The abstract graph of Fig. 2(b) represents a set of concrete graphs having three **a**-nodes connected to each other forming **b** cycles, such as in Fig. 3(b).

Let us fix some notations. Let A be a set and $\sim \subseteq A \times A$ be an equivalence relation over A . For $x \in A$, $[x]_{\sim}$ denotes the equivalence class of x induced by \sim , and A/\sim is the set of \sim -equivalence classes in A . If \sim and \sim' are two equivalence relations over A such that $\sim \subseteq \sim'$, then \sim is called a *refinement* of \sim' .

Definition 1 (abstract graph). An abstract graph S is a structure $(G_S, \sim_S, \text{mult}_{n,S}, \text{mult}_{out,S}, \text{mult}_{in,S})$ where

- $G_S = (N_S, E_S, \text{src}_S, \text{tgt}_S, \text{lab}_S)$ is a graph;
- $\sim_S \subseteq N_S \times N_S$ is an equivalence relation on N_S called the grouping relation;
- $\text{mult}_{n,S} : N_S \rightarrow \mathbf{M}_{\nu}^+$ is a node multiplicity function;
- $\text{mult}_{out,S} : N_S \times \text{Lab} \times N_S/\sim_S \rightarrow \mathbf{M}_{\mu}$ is an outgoing edges multiplicity function;
- $\text{mult}_{in,S} : N_S \times \text{Lab} \times N_S/\sim_S \rightarrow \mathbf{M}_{\mu}$ is an incoming edges multiplicity function.

Moreover, for any $v \in N_S$, $\mathbf{a} \in \text{Lab}$ and $C \in N_S/\sim_S$, we require $\text{mult}_{out,S}(v, \mathbf{a}, C) = 0$ iff $v \triangleright\triangleright_{G_S}^{\mathbf{a}} C = \emptyset$, and $\text{mult}_{in,S}(v, \mathbf{a}, C) = 0$ iff $C \triangleright\triangleright_{G_S}^{\mathbf{a}} v = \emptyset$.

Formally, the relation between concretisations of an abstract graph S and S is captured by *abstraction morphisms* respecting multiplicity.

Definition 2 (abstraction morphism, concretisation). Let G be a graph and S be an abstract graph. An abstraction morphism of G into S is a surjective graph morphism $s : G \rightarrow G_S$ such that the following conditions are met:

1. for all $w \in N_S$: $\text{mult}_{\text{n},S}(w) = |s^{-1}(w)|_\nu$;
2. for all $w \in N_S$, for all $\mathbf{a} \in \text{Lab}$, for all $C \in N_S / \sim_S$, and for all $v \in s^{-1}(w)$:

$$\text{mult}_{\text{out},S}(w, \mathbf{a}, C) = |v \triangleright \triangleright_G^{\mathbf{a}}(s^{-1}(C))|_\mu \text{ and } \text{mult}_{\text{in},S}(w, \mathbf{a}, C) = |(s^{-1}(C)) \triangleright \triangleright_G^{\mathbf{a}} v|_\mu.$$

If $s : G \rightarrow S$ is an abstraction morphism, then G is a concretisation of S . The set of all concretisations of S is written $\text{Concr}(S)$.

As another example, Fig. 4, page 329 shows an abstract graph for the firewall example from Fig. 1, with $\nu = \mu = 1$. The corresponding abstraction morphism summarises the two LI-neighbours of the internal interface to the unique LI-neighbour of the internal interface in the abstract graph. The three LO nodes that have only LO neighbours are summarised to a unique node. All other nodes in the abstract graph have multiplicity one and correspond to a unique node in the concrete graph.

Note that the requirements on outgoing (resp. incoming) edge multiplicities guarantee in particular that two different nodes v, v' of graph G can only be summarised by an abstraction morphism, if they have the same outgoing and incoming edges multiplicities with respect to a label and a group of nodes.

Construction of an Abstract Graph. Definitions 1 and 2 are declarative and do not give a hint on the effective construction of an abstract graph. This can be done as follows: Let G be a graph and $\sim, \equiv \subseteq N_G \times N_G$ be two equivalence relations such that \equiv refines \sim . Furthermore, assume that for any $v, v' \in N_G$, for any $C \in N_G / \sim$, and for any label \mathbf{a} : $v \equiv v'$ implies

$$|v \triangleright \triangleright_G^{\mathbf{a}} C|_\mu = |v' \triangleright \triangleright_G^{\mathbf{a}} C|_\mu \quad \text{and} \quad |C \triangleright \triangleright_G^{\mathbf{a}} v|_\mu = |C \triangleright \triangleright_G^{\mathbf{a}} v'|_\mu$$

Then \equiv and \sim induce an *abstract graph*, $S = (N_S, \sim_S, \text{mult}_{\text{n}}, \text{mult}_{\text{out}}, \text{mult}_{\text{in}})$, and an *abstraction morphism*, $s : G \rightarrow S$ as follows:

- Extend the equivalence relation \equiv to edges as follows: $e \equiv e'$ if $\text{src}_G(e) \equiv \text{src}_G(e')$, $\text{tgt}_G(e) \equiv \text{tgt}_G(e')$ and $\text{lab}_G(e) = \text{lab}_G(e')$. Then $S_G = (N_S, E_S, \text{src}_S, \text{tgt}_S, \text{lab}_S)$ is the graph quotient of G w.r.t. \equiv , i.e. $N_S = N_G / \equiv$; $E_S = E_G / \equiv$; and for any edge $[e]_{\equiv}$ in E_S , $\text{src}_S([e]_{\equiv}) = [\text{src}_G(e)]_{\equiv}$, $\text{tgt}_S([e]_{\equiv}) = [\text{tgt}_G(e)]_{\equiv}$ and $\text{lab}_S([e]_{\equiv}) = \text{lab}_G(e)$. Note that, because of the definition of \equiv on edges $[e]_{\equiv}$ is well-defined.
- Mapping $s : N_G \cup E_G \rightarrow N_S \cup E_S$ is defined by $s(v) = [v]_{\equiv}$ and $s(e) = [e]_{\equiv}$ for any $v \in N_G$ and any $e \in E_G$ and extended to preserve \perp .
- $\sim_S \subseteq N_S \times N_S$ is the equivalence relation given by $[v]_{\equiv} \sim_S [v']_{\equiv}$ if $v \sim v'$ for all $v, v' \in N_G$.
- $\text{mult}_{\text{n}} : N_S \rightarrow \mathbf{M}_\nu^+$ is defined by $\text{mult}_{\text{n}}(w) = |s^{-1}(w)|_\nu$ for all w in N_S .
- $\text{mult}_{\text{out}}, \text{mult}_{\text{in}} : N_S \times \text{Lab} \times N_S / \sim_S \rightarrow \mathbf{M}_\mu$ are mappings defined by $\text{mult}_{\text{out}}([v]_{\equiv}, \mathbf{a}, C) = |v \triangleright \triangleright_G^{\mathbf{a}} C|_\mu$ and $\text{mult}_{\text{in}}([v]_{\equiv}, \mathbf{a}, C) = |C \triangleright \triangleright_G^{\mathbf{a}} v|_\mu$ for all $v \in N_G$, $\mathbf{a} \in \text{Lab}$, and $C \in N_S / \sim_S$.

It is obvious that S and s are indeed a well-defined abstract graph and abstraction morphism for two such equivalence relations. The complete formalisation

and proof of this construction is in [13]. Note that not all abstract graphs can be thus defined. Such abstract graphs necessarily have concretisations and cannot have parallel edges, which is not the case for general abstract graphs. Still, any abstract graph admitting concretisations and without parallel edges can be defined this way. For a graph G and equivalence relations \sim and \equiv as above we write $abstr_graph(G, \sim, \equiv)$ and $abstr_morph(G, \sim, \equiv)$ for the abstract graph and the corresponding abstraction morphism constructed as shown above.

2.2 Isomorphism of Abstract Graphs

Abstract graphs can be abstracted just like graphs, yielding “more abstract” graphs. In this section we describe this abstraction relation, which is composable. We then use it to define the notion of *isomorphism* between abstract graphs having the interesting property that isomorphic abstract graphs have the same concretisations.

Definition 3 (abstraction morphism between abstract graphs). *An abstraction morphism from an abstract graph S to an abstract graph T is a graph morphism $f : G_S \rightarrow G_T$ that complies to the following axioms:*

1. $\forall v, v' \in N_S: v \sim_S v'$ implies $f(v) \sim_T f(v')$;
2. $\forall w \in N_T: \mathbf{mult}_{n,T}(w) = \left(\sum_{v \in f^{-1}(w)} \mathbf{mult}_{n,S}(v) \right)$;
3. $\forall w \in N_T, \forall \mathbf{a} \in \mathbf{Lab}, \forall C \in N_T / \sim_T, \forall v \in f^{-1}(w)$, it holds

$$\mathbf{mult}_{out,T}(w, \mathbf{a}, C) = \sum_{D \in (f^{-1}(C)) / \sim_S}^{\mu} \mathbf{mult}_{out,S}(v, \mathbf{a}, D)$$

and similarly for the incoming edges multiplicity.

The axioms in the previous definition are well-defined. In the third axiom we sum up $\mathbf{mult}_{out,S}(v, \mathbf{a}, D)$ and $\mathbf{mult}_{in,S}(v, \mathbf{a}, D)$ for all $D \in (f^{-1}(C)) / \sim_S$. It is then necessary that all the triples (v, \mathbf{a}, D) belong to the domain of $\mathbf{mult}_{in,S}$, that is, it is necessary that any such D belongs to N_S / \sim_S . This is indeed the case thanks to the first axiom.

There is an analogy between an abstraction morphism between abstract graphs (Def. 3) and an abstraction morphism from a graph into an abstract graph (Def. 2). Namely, the second axiom in Def. 3 corresponds to the first axiom in Def. 2, but we sum up node multiplicities instead of simply counting nodes. Also the third axiom in Def. 3 and the second axiom in Def. 2 are analogous. Note that the composition of two abstraction morphisms yields another abstraction morphism [13].

We can now define two abstract graphs S and T to be *isomorphic* if there exists an isomorphism $f : G_S \rightarrow G_T$ such that both f and f^{-1} are abstraction morphisms. This leads us to the following interesting statement proven in [13].

Lemma 1. *If two abstract graphs S and T are isomorphic, then they have the same concretisations.*

Note that the inverse is not true. To this end, consider two abstract graphs S and T , where S has a single node of multiplicity two and no edges. T has two nodes, each of multiplicity one, and no edges. S and T both have a unique concretisation (up to graph isomorphism) which is the graph with two nodes and no edges, but S and T are clearly not isomorphic.

2.3 Neighbourhood Abstraction

So far, we have defined the notion of abstract graphs, concretisations, and abstraction morphisms. In our construction of abstract graphs, we assumed the existence of equivalence relations \equiv and \sim having particular properties. We shall now define an interesting choice of such relations inducing the notion of *neighbourhood abstractions*. This conveys the central idea of our work. In such an abstract graph, each node represents concrete graph nodes that have *similar neighbourhood, up to some “radius” i* . This i is a parameter of the precision of the neighbourhood abstraction. By gradually increasing i , we can obtain more precise abstractions, if the current one is too imprecise to verify the desired properties (abstraction refinement). We shall now define equivalence between nodes according to their neighbourhoods and, subsequently, neighbourhood abstraction.

Definition 4 (neighbourhood abstraction). *Let G be a graph. For each natural $i > 0$, we define the i -neighbourhood equivalence relation \equiv_i over N_G recursively by:*

- $v \equiv_0 v'$ if $\text{lab}_G(v) = \text{lab}_G(v')$;
- $v \equiv_{i+1} v'$ if $v \equiv_i v'$, and $|v \triangleright^a C|_\mu = |v' \triangleright^a C|_\mu$, and $|C \triangleright^a v|_\mu = |C \triangleright^a v'|_\mu$ for all label a in Lab and for all $C \in N / \equiv_i$.

The level i neighbourhood abstraction of G is $\text{abstr_graph}(G, \equiv_{i-1}, \equiv_i)$ and the corresponding abstraction morphism is $\text{abstr_morph}(G, \equiv_{i-1}, \equiv_i)$.

Two nodes are mapped to the same node of the abstract graph if they are neighbourhood equivalent up to some radius. The grouping relation is also given by neighbourhood equivalence, but using a smaller radius. Figure 4 shows the level 1 neighbourhood abstraction of the firewall configuration from Fig. 1 for $\mu = 1$ and $\nu = 1$.

It is obvious from the definition that the level $i + 1$ neighbourhood abstraction of a graph refines the level i neighbourhood abstraction. This is the basis of our abstraction refinement mechanism. The neighbourhood abstraction of a graph is defined syntactically, which ties it to its representation. To avoid this inconvenient situation, in the sequel by neighbourhood abstraction of a graph we mean the isomorphism class of the actual abstract graph.

Neighbourhood abstraction behaves well w.r.t. isomorphism (Lemma 2 below). In combination with Lemma 1 this shows that two graphs obtained by neighbourhood abstraction are isomorphic iff they have the same concretisations.

Lemma 2. *Let S and T be two abstract graphs obtained by neighbourhood abstraction. If S and T admit a common concretisation, then they are isomorphic.*

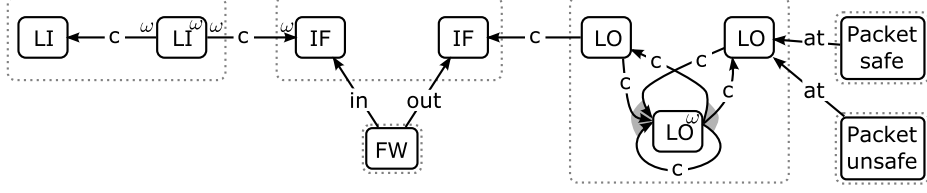


Fig. 4. The level one neighbourhood abstraction of the firewall example. Omitted node and edge multiplicities are equal to one.

3 A Modal Logic for Graphs and Abstract Graphs

So far, we have delineated a novel abstraction to be employed by abstract graph transformation, neighbourhood abstraction. For verification, we also need an accompanying logic to be defined now, which can be evaluated on both concrete and abstract graphs. A central theorem of our work states that this logic is preserved *and* reflected by neighbourhood abstraction.

Before we give the formal definition, let us look at properties for the firewall configuration of Fig. 1: (1) a packet cannot be safe and unsafe simultaneously; (2) an in-location cannot be directly connected to an out-location; (3) an unsafe packet never reaches an in-location; (4) a packet has at most one current position. These can be expressed in our logic as follows, where the $\rangle\text{at}\langle^\lambda$ operator is a forward existential modality, indicating the existence of *at least λ outgoing c -edge*; $\langle\text{at}\langle^\lambda$ is similar, but for incoming edges; \mathbf{t} stands for the true formula.

- | | |
|--|--|
| (1) $(\text{safe} \rightarrow \neg\text{unsafe}) \wedge (\text{unsafe} \rightarrow \neg\text{safe})$ | (3) $\neg(\text{LI} \wedge \langle\text{at}\langle^1 \cdot \text{unsafe})$ |
| (2) $\neg(\text{LI} \wedge \rangle c \langle^1 \cdot \text{LO}) \wedge \neg(\text{LO} \wedge \rangle c \langle^1 \cdot \text{LI})$ | (4) $\text{Packet} \rightarrow \neg\langle\text{at}\langle^2 \cdot \mathbf{t}$ |

3.1 Syntax and Semantics of the Logic

Logic formulae are defined by the following syntax (for $\mathbf{a} \in \text{Lab}$ and $\lambda \in \mathbf{M}_\mu$):

$$\phi ::= \mathbf{a} \mid \neg\phi \mid \phi \vee \phi' \mid \rangle\mathbf{a}\langle^\lambda \cdot \phi \mid \langle\mathbf{a}\langle^\lambda \cdot \phi$$

The *nesting depth* $d(\phi)$ of ϕ measures the maximal number of nested modalities. It is defined recursively as: $d(\mathbf{a}) = 0$, $d(\rangle\mathbf{a}\langle^\lambda \cdot \phi) = d(\langle\mathbf{a}\langle^\lambda \cdot \phi) = 1 + d(\phi)$, $d(\neg\phi) = d(\phi)$, $d(\phi \vee \phi') = d(\phi \wedge \phi') = \max(d(\phi), d(\phi'))$ for any \mathbf{a} in **Lab**. We write \mathcal{L}_i for the set of formulae with nesting depth at most i . Logic formulae are interpreted in graph nodes. For a graph G , a node v in N_G , and a formula ϕ , the *satisfaction relation* $G, v \models \phi$ is defined recursively on the structure of ϕ by:

- $G, v \models \mathbf{a}$ if $\mathbf{a} \in \text{lab}(v)$;
- $G, v \models \neg\phi$ if $G, v \not\models \phi$;
- $G, v \models \phi \vee \phi'$ if $G, v \models \phi$ or $G, v \models \phi'$;
- $G, v \models \rangle\mathbf{a}\langle^\lambda \cdot \phi$ if $|\{e \in v \triangleright^{\mathbf{a}} \mid G, \text{tgt}(e) \models \phi\}|_\mu \geq \lambda$;
- $G, v \models \langle\mathbf{a}\langle^\lambda \cdot \phi$ if $|\{e \in v \triangleleft^{\mathbf{a}} \mid G, \text{src}(e) \models \phi\}|_\mu \geq \lambda$.

If $G, v \models \phi$, we say that ϕ holds in node v . Intuitively, a formula of the form $\langle \mathbf{a} \rangle^\lambda \phi$ holds in a node v if the μ -bounded number of \mathbf{a} -labelled edges (e) connecting it to some node v' ($\text{src}_G(e) = v$ and $\text{tgt}_G(e) = v'$) in which ϕ holds is at least λ . Analogously, $\langle \mathbf{a} \rangle^\lambda \cdot \phi$ holds in v if the number of \mathbf{a} -labelled edges connecting some such v' to v is at least λ .

The satisfaction relation is defined for an abstract graph almost in the same way as for a (concrete) graph. The difference is the way it is defined for modalities. There, we no longer count individual edges, but sum up edge multiplicities instead.

- $S, v \models \langle \mathbf{a} \rangle^\lambda \cdot \phi$ if $\lambda \leq \sum_{C \in X} \text{mult}_{\text{out},S}(v, \mathbf{a}, C)$, and
- $S, v \models \langle \mathbf{a} \rangle^\lambda \cdot \phi$ if $\lambda \leq \sum_{C \in X} \text{mult}_{\text{in},S}(v, \mathbf{a}, C)$,

where $X = \{C \in N_S / \sim_S \mid \forall w \in C. S, w \models \phi\}$. In the firewall example, $\text{IF} \rightarrow \langle \mathbf{c} \rangle^1 \cdot (\text{LI} \vee \text{LO})$ holds in all nodes of the level 1 neighbourhood abstraction depicted. Note finally that counting makes our logic more expressive than the “usual” modal logic, while still strictly enclosed in first-order logic.

3.2 Preservation and Reflection

Let $s : G \rightarrow S$ be an abstraction morphism from the graph G to the abstract graph S . We say that s *preserves* a property p if whenever p holds in the node v of G , it also holds in the node $s(v)$ of S . Inversely, we say that s *reflects* p if whenever p holds in the node $s(v)$ of S , it also holds in the node v of G . One can also define in a similar manner preservation and reflection by an abstraction morphism between abstract graphs. Preservation and reflection are very important characterisations. If an abstraction preserves a set of safety properties, these properties can be verified on the abstract level. If an abstraction reflects a set of properties, then any characterisation of an abstract graph also holds for its concretisations. If both preservation and reflection hold, verifying a property on a graph is equivalent to verifying it on the abstract level. Neighbourhood abstraction features preservation and reflection of logic formulae with appropriate nesting depth, as stated in the following theorem.

Theorem 1 (Preservation and Reflection). *Let G be a graph and S the level i neighbourhood abstraction of G , for some $i \geq 1$, with corresponding abstraction morphism $s : G \rightarrow S$. Then s preserves and reflects $\mathcal{L}_i(\text{Lab})$.¹*

An important consequence of it is that neighbourhood abstraction can be parametrised by the properties we want to verify by choosing the level of abstraction

¹ Preservation of formulae with negation may seem in contradiction with the Morphism Preservation Theorem for finite structures [14] stating that a first order formula is preserved by morphism iff it is equivalent to an existential positive formula. Some modal logic formulae cannot be expressed in first-order logic without negation (e.g. $\neg \langle \mathbf{a} \rangle^\lambda \cdot \mathbf{\#}$). However, in our case, abstract graphs contain information on the interpretation of negated formulae, by means of the multiplicity functions explaining this apparent contradiction.

that preserves the properties one is interested in. The following lemma formalises the relation between the logic and neighbourhood equivalence:

Lemma 3. *Two nodes v, v' of a graph G are i -neighbourhood equivalent if, and only if, the same $\mathcal{L}_i(\text{Lab})$ formulae hold in v and in v' .*

For our running example, let G be the graph of Fig. 1 and S its level 1 neighbourhood abstraction of Fig. 4. Let $s : G \rightarrow S$ be the corresponding abstraction morphism, and let $\phi = \text{LO} \wedge \rangle \text{c} \rangle^1 \cdot \text{IF}$ and $\psi = \text{LO} \wedge \rangle \text{c} \rangle^1 \cdot \rangle \text{c} \rangle^1 \cdot \text{IF}$. In G , ϕ only holds for the LO-neighbour of the out interface, and in S , ϕ only holds for the corresponding abstract node. That is, ϕ of nesting depth 1 is preserved and reflected by s , whereas ψ , a formula of nesting depth 2, is not reflected. Indeed, in S , ψ holds in the LO-node with multiplicity ω but only in one of the pre-images of this node in G .

4 Canonical Representation of the Neighbourhood Abstraction

For abstract graph transformation, it is crucial to determine whether a newly computed abstract graph has been met before. To avoid expensive isomorphism checks on abstract graphs, we can benefit from a *canonical representation* of neighbourhood abstracted graphs. In effect, this reduces isomorphism checks to mere equality tests and is another important contribution we make.

Canonical names. Canonical names occur frequently in literature, e.g., in [11]. Here, a canonical name is a unique representation of an equivalence class w.r.t. a neighbourhood equivalence relation, which is independent of the underlying graph. For instance, each equivalence class for \equiv_0 contains only nodes having the same labels and is identified by this set of labels. It becomes the canonical name of this equivalence class. Each relation \equiv_i is equipped with a set NCan^i of canonical names.

Definition 5 (Canonical Name). *The set of level i node canonical names, NCan^i , is defined inductively for $i \geq 0$:*

$$\begin{aligned} \text{NCan}^0 &= {}_2\text{Lab} \\ \text{NCan}^{i+1} &= \text{NCan}^i \times (\text{NCan}^i \times \text{Lab} \rightarrow \mathbf{M}_\mu) \times (\text{NCan}^i \times \text{Lab} \rightarrow \mathbf{M}_\mu). \end{aligned}$$

The set ECan^i of level i edge canonical names is $\text{ECan}^i = \text{NCan}^i \times \text{Lab} \times \text{NCan}^i$. Let G be a graph. The mapping name_G^i maps nodes and edges of G to their level i canonical name as follows. For node v of G , $\text{name}_G^0(v) = \text{lab}_G(v)$, and $\text{name}_G^{i+1}(v) = (\text{name}_G^i(v), \text{out}, \text{in})$ where for $C \in \text{NCan}^i$ and for each $a \in \text{Lab}$ (N_C stands for the set of nodes v' such that $\text{name}_G^i(v') = C$),

$$\begin{aligned} \text{out}(C, a) &= |v \triangleright_G^a N_C|_\mu & \text{in}(C, a) &= |N_C \triangleright_G^a v|_\mu. \end{aligned}$$

For edge e of G , $\text{name}_G^i(e) = (\text{name}_G^i(\text{src}(e)), \text{lab}(e), \text{name}_G^i(\text{tgt}(e)))$.

Note that the number of different level i canonical names is finite. In combination with Lemma 4 below, we conclude that the number of level i neighbourhood abstractions is also finite up to isomorphism facilitating the verification of potentially infinite systems.

In the firewall example (Fig. 4), the different level zero canonical names are $C_1 = \{\text{FW}\}$, $C_2 = \{\text{IF}\}$, $C_3 = \{\text{LI}\}$, $C_4 = \{\text{LO}\}$, $C_5 = \{\text{Packet, safe}\}$ and $C_6 = \{\text{Packet, unsafe}\}$. The level one canonical name for the in-interface is $(C_2, \mathbf{0}, in)$, where $in = \{(C_1, in) \mapsto 1, (C_3, c) \mapsto \omega\}$, and $\mathbf{0}$ is the constant zero function. The canonical name for the bottom-most LO-node is (C_4, out, in) , where $out = \{(C_4, c) \mapsto 1\}$ and $in = \{(C_4, c) \mapsto 1\}$.

There is an obvious relation between canonical names and the neighbourhood equivalence expressed in the following central theorem. As a consequence of it and Lemma 3 we obtain that $v \equiv_i v'$ iff $\text{name}_G^i(v) = \text{name}_G^i(v')$, iff v and v' satisfy the same \mathcal{L}_i logic formulae. This closes the circle between neighbourhood equivalence, canonical names, and logical satisfaction.

Theorem 2. *For any $i \geq 0$, any graph G , any two nodes v, v' of G and any two edges e, e' of G , $v \equiv_i v'$ if, and only if, $\text{name}_G^i(v) = \text{name}_G^i(v')$, and $e \equiv_i e'$ if, and only if, $\text{name}_G^i(e) = \text{name}_G^i(e')$.*

Canonical Representation of the Neighbourhood Abstraction. Let G be a graph. Consider the triple $\mathcal{C}_G = (\text{name}^i(N_G), \text{name}^i(E_G), \text{mult})$, where $\text{mult} : \text{name}^i(N_G) \rightarrow \mathbf{M}_\nu^+$ is the function defined by $\text{mult}(C) = |\{v \in N_G \mid \text{name}_G^i(v) = C\}|_\nu$ for all $C \in \text{name}^i(N_G)$. Then \mathcal{C}_G is a canonical representation of the isomorphism class of the level i neighbourhood abstraction of G , as stated below:

Lemma 4. *Let G, H be graphs, and let $i \geq 1$. The level i neighbourhood abstractions of G and H are isomorphic if, and only if, \mathcal{C}_G and \mathcal{C}_H are equal.*

By \mathcal{C}_G and \mathcal{C}_H are equal, we mean component-wise equality, that is, equality of the sets of node and edge canonical names and equality of the node multiplicity functions that define them. Effectively, this allows us to reduce isomorphism checks on neighbourhood abstracted graphs to mere equality tests.

5 Towards an Automatic Verification Framework

We have defined a framework of neighbourhood abstractions having canonical representations and showed that an accompanying logic is preserved and reflected by it. We have not yet said how the application of a graph production rule is lifted to work on abstract graphs. Unfortunately, for lack of space, we need to refer the reader to [13] for a detailed treatment. In general, a rule application consists of three stages (which is typical of abstract graph transformation in general), where S is an abstract graph and (L, R, p) a production rule.

1. *Materialisation:* Transform the abstract graph S into the less abstract graph S' , such that there is an abstraction morphism $S' \rightarrow S$ and a matching $m : L \rightarrow S'$, the image of which is a concrete sub-graph of S' ; i.e. a sub-graph

in which all node and edge multiplicities are equal to one. S' is not unique, and the aim of materialisation is to construct the set of abstract graphs \mathcal{S} such that for all concretisations G of S , for all matchings $m : G \rightarrow S$, there exists $T \in \mathcal{S}$ and abstraction morphisms $s : G \rightarrow T$, $t : T \rightarrow S$ such that the image of $s \circ m$ is concrete in T .

2. *Update*: As we are dealing with concrete (sub-)graphs now, rule application is just as usual and applied to each element of materialisation.
3. *Normalisation*: The graphs obtained after updates need not necessarily be in canonical form. Therefore, we need to neighbourhood abstract them again.

We show in [13] that this abstract transformation mechanism is sound, in the sense that it over-approximates the concrete graph abstraction. If a graph G can be transformed by rule (L, R, p) yielding a graph H , then G 's abstraction S can be abstractly transformed by the same rule yielding H 's abstraction T . Note that general negative application conditions (NACs) cannot be handled, as satisfiability of such conditions is not preserved by the abstraction. However, transformation can be defined for simple NACs which satisfiability can be checked using the canonical name only, as for instance conditions testing for the absence of an adjacent edge or path of bounded length.

The overall verification of a system works as follows. Start with the neighbourhood abstraction of the initial graph of the system to be verified. Given the abstract transformation defined above, successively apply it to construct an abstract version of the graph transformation system which has neighbourhood abstractions as states and which is *guaranteed* to be *finite* regardless the original system. The canonical representation of the neighbourhood abstraction makes it easy to check whether a newly derived state has already been met before, which is a very costly operation in normal graph transformations. Moreover, the abstraction can be parametrised by the property one wants to verify, expressed in the modal logic. As the modal logic is preserved by the abstraction, one may now evaluate formulas on finite, abstract graphs to obtain information about the possibly infinite-state original system. Finally, note that our framework naturally gives rise to abstraction refinement: If the level i neighbourhood abstraction is not conclusive, then one can try level $i + 1$.

Running Example. In the abstract graph transformation system (GTS) induced by the level 1 abstraction of the firewall example, all reachable abstract graphs have one FW-node and two IF-nodes, to which the different possible configurations of the internal and external networks are connected. The number of abstract configurations is bounded, whereas the number of concrete configurations is infinite, because of the possibility of creating packets and connecting new locations. Consider now the four properties listed above. They are all safety properties, defining invariants that should hold in all state of the GTS. These properties indeed hold in all states of the abstract GTS. As the four formulae are of nesting depth one, by reflection of the logic we can deduce that they also hold in all states of the concrete GTS. For this particular example, the abstraction mechanism allows to *verify* the four properties using the level 1 neighbourhood abstraction.

Usability and Limitations. Our verification framework is fully automatic and parametrised by the properties of interest. Due to the local nature of neighbourhood abstractions, it works well on systems where updates are determined locally and where reachability is not important. Typical use cases include the firewall example as given here or wireless traffic control systems as, e.g. the ones investigated in [7]. Also an application to ad-hoc network routing is promising but has not yet been explored.

However, our technique is not so suited for systems, where reachability is of importance, which is often the case for verification of software with dynamically allocated data structures. For instance, in the case of linked lists, our abstraction (regardless of radius) cannot distinguish between circular and non circular lists, which in practice results in lots of spurious states and transitions in the abstract transition system (*i.e.* states and transitions that do not exist in the concrete system). Also, the rather high complexity of our approach might be prohibitive for really large examples. This is yet to be explored by careful experimentation.

6 Conclusion

We presented a framework of graph abstractions, called neighbourhood abstraction, which generalises previous approaches to abstract graph transformation, a method for formal verification of graph transformation systems. The abstraction is based on regrouping nodes with similar neighbourhood, and can be parametrised by the radius of the neighbourhood to be considered. It is guaranteed to yield systems of finite, bounded size facilitating their exploration. We also presented a modal logic that can be interpreted both on graphs and on abstract graphs. The logic and the abstraction are closely related, which makes it possible to parametrise the abstraction so that it preserves and reflects the valuation of formulas. We delineated the implementation of a fully automated verification framework based on our novel abstractions and facilitated by a canonical form of abstract graphs. Interestingly, this framework also allows for automated abstraction refinement. Our proposal is illustrated by an interesting and relevant graph transformation based model of a firewall system. Usability and limitations of our approach were clearly identified. Note that related work was already discussed in the Introduction and that proofs and some other tedious formalities were left out but can be found in [13].

Future Work. We plan to implement our technique within the GROOVE [15] framework, a standard tool for graph transformations. This will allow for a more thorough exploration of more examples and for a qualified judgement on practical scalability. We believe that our framework caters for all possible *local* abstractions, where locality refers to the portion of the graph used to determine the equivalence of nodes. On the other hand, this complicates the verification of heap-manipulating programs, where reachability, a more global property, is crucial. Therefore, we are working on abstractions taking reachability into account. This is similar to abstractions used in the work of Sagiv *et al.* on shape analysis (see [11] for an overview) of heap-manipulating programs. In this work,

the authors use logical structures to represent memory states of programs; abstract structures are 3-valued logical structures. Properties on these structures are defined using first-order logic with transitive closure (FO+TC) enabling the definition of reachability. It seems promising to explore the opportunities offered by FO+TC for abstract graph transformation too.

References

1. Heckel, R.: Compositional verification of reactive systems specified by graph transformation. In: Astesiano, E. (ed.) ETAPS 1998 and FASE 1998. LNCS, vol. 1382, pp. 138–153. Springer, Heidelberg (1998)
2. Becker, B., Beyer, D., Giese, H., Klein, F., Schilling, D.: Symbolic invariant verification for systems with dynamic structural adaptation. In: ICSE. ACM Press, New York (2006)
3. Saksena, M., Wibling, O., Jonsson, B.: Graph grammar modeling and verification of ad hoc routing protocols. In: Ramakrishnan, C.R., Rehof, J. (eds.) TACAS 2008. LNCS, vol. 4963. Springer, Heidelberg (2008)
4. Baldan, P., Corradini, A., König, B.: Verifying finite-state graph grammars: An unfolding-based approach. In: Gardner, P., Yoshida, N. (eds.) CONCUR 2004. LNCS, vol. 3170. Springer, Heidelberg (2004)
5. König, B., Kozioura, V.: Counterexample-guided abstraction refinement for the analysis of graph transformation systems. In: Hermanns, H., Palsberg, J. (eds.) TACAS 2006. LNCS, vol. 3920. Springer, Heidelberg (2006)
6. Rensink, A., Distefano, D.: Abstract Graph Transformation. In: International Workshop on Software Verification and Validation (SVV). ENTCS (2005)
7. Bauer, J.: Analysis of Communication Topologies by Partner Abstraction. PhD thesis, Universität des Saarlandes (2006)
8. Bauer, J., Wilhelm, R.: Static analysis of dynamic communication systems by partner abstraction. In: Riis Nielson, H., Filé, G. (eds.) SAS 2007. LNCS, vol. 4634. Springer, Heidelberg (2007)
9. Cousot, P., Cousot, R.: Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints. In: POPL, pp. 238–252 (1977)
10. Rensink, A.: Canonical Graph Shapes. In: Schmidt, D. (ed.) ESOP 2004. LNCS, vol. 2986, pp. 401–415. Springer, Heidelberg (2004)
11. Sagiv, M., Reps, T., Wilhelm, R.: Parametric Shape Analysis via 3-valued Logic. *ACM Transactions on Programming Languages and Systems* 24(3) (May 2002)
12. Bauer, J., Boneva, I., Rensink, A., Kurbán, M.E.: A modal-logic based graph abstraction, <http://www7.in.tum.de/~joba/icgt08.pdf>
13. Boneva, I., Rensink, A., Kurbán, M.E., Bauer, J.: Graph Abstraction and Abstract Graph Transformation. Technical report, University of Twente (2007)
14. Rossman, B.: Existential Positive Types and Preservation under Homomorphisms. In: 20th IEEE Symposium on Logic in Computer Science, pp. 467–476. CSP (2005)
15. Rensink, A.: The GROOVE Simulator: A Tool for State Space Generation. In: Pfaltz, J.L., Nagl, M., Böhlen, B. (eds.) AGTIVE 2003. LNCS, vol. 3062, pp. 479–485. Springer, Heidelberg (2004)