Graph Grammar Modeling and Verification of Ad Hoc Routing Protocols (Extended Version)

Mayank Saksena, Oskar Wibling, and Bengt Jonsson

Dept. of Information Technology, P.O. Box 337, S-751 05 Uppsala, Sweden
{mayanks, oskarw, bengt}@it.uu.se

Abstract. We present a technique for modeling and automatic verification of network protocols, based on graph transformation. It is suitable for protocols with a potentially unbounded number of nodes, in which the structure and topology of the network is a central aspect, such as routing protocols for ad hoc networks. Safety properties are specified as a set of undesirable global configurations. We verify that there is no undesirable configuration which is reachable from an initial configuration, by means of symbolic backward reachability analysis.
In general, the reachability problem is undecidable. We implement the technique in a graph grammar analysis tool, and automatically verify several interesting nontrivial examples. Notably, we prove loop freedom for the DYMO ad hoc routing protocol. DYMO is currently on the IETF standards track, to potentially become an Internet standard.

1 Introduction

The verification of network protocols has been one of the most important driving forces for the development of model checking technology. Most approaches (e.g., [17, 13]) analyze finite-state models of protocols, but an increasing number of techniques are developed for analyzing parameterized or infinite-state models (see, e.g., [3, 26, 2]). In this paper, we consider verification of protocols for networks with a potentially unbounded number of nodes, possibly with a dynamically changing topology. This is a large class of protocols, including protocols for wireless ad hoc networks, many distributed algorithms, security protocols, telephone system services, etc. Global configurations of such protocols are naturally modeled using graphs, that are transformed by the dynamic behavior of the protocol, and therefore various forms of graph transformation systems have been used to model and analyze them [21, 8].

In this paper, we present a technique for modeling and verification of protocols using a variant of *graph transformation systems* (GTSs) [21, 8]. We use a general mechanism for expressing conditions on the applicability of a rule, in the form of *negative application conditions (NACs)*. Sets of global configurations are symbolically represented by *graph patterns* [8], which are graphs extended with NACs. Intuitively, a graph pattern represents the set of configurations that contain it as a subgraph, but none of the NACs. A safety property of a protocol is represented by a set of graph patterns that represent undesirable global configurations.

We consider the problem of verifying safety properties. This can be reduced to the problem whether an undesirable configuration can be reached, by a sequence of graph transformation steps, from some initial global configuration. We present a method for automatically checking such a reachability problem by backward reachability analysis. Backward reachability analysis is a powerful verification technique, which has generated decidability results for many classes of parameterized and infinite-state systems (e.g., [4, 3, 15]) and proven to be highly useful also for undecidable verification problems (e.g., [1]). By fixed point computation, we compute an over-approximation of the set of configurations from which a bad configuration can be reached, and check that this set contains no initial configuration. The central part of the backward reachability procedure is to compute the predecessors of a set of configurations in this symbolic representation. Since the reachability problem is undecidable in general, the fixed point computation is not guaranteed to terminate. However, we show that the techniques are powerful enough for verifying several interesting nontrivial examples, indicating that the approach is useful for network protocols where the dynamically changing topology of the network is a central aspect.

A main motivation for our work is to analyze protocols for wireless ad hoc networks, including the important class of *routing protocols*. We have implemented our technique, and successfully verified that the DYMO protocol [11] will never generate routing loops. Verifying loop freedom for ad hoc routing protocols has been the subject of much work [9, 14]; several previous protocol proposals have been incorrect in this respect [10, 5]. Our verification method handles a detailed ad hoc routing protocol model, with relatively little effort. In our work, we have also found GTSs to be an intuitive and visually clear form of modeling.

Related work. There have been several efforts to verify loop freedom of routing protocols for ad hoc networks. Bhargavan et al. [9] verified AODV [23] to be loop free, using a combination of SPIN for model checking a finite network model, and HOL theorem proving for generalizing the proof. In contrast, we prove the same property automatically for an arbitrary number of nodes. Our experience is that modeling using GTSs is more intuitive than to separately construct SPIN models and HOL proofs. Das and Dill [14] developed automatic predicate discovery for use in predicate abstraction, and proved loop freedom for a simplified version of AODV, excluding timeouts. The construction of an abstract system and discovery of relevant abstraction predicates require many calls to a theorem prover; our method does not need to interact with a theorem prover. We check the graphs directly for inconsistencies.

There have been several other approaches to modeling and analysis using variants of GTSs. König and Kozioura [21] over-approximate graph transformation systems using Petri nets, successively constructed using forward counter-example guided abstraction refinement. Their technique does not support the use of NACs. We have found NACs to be an advantage during modeling and verification. For example, our first approach at verifying the DYMO protocol was without NACs, resulting in a more complex model with features not directly related to the central protocol function.

Kastenberg and Rensink [20] translate GTSs to finite-state models in the GROOVE tool by putting an upper bound on the number of nodes in a network. Becker et al. [8] verified safety properties of mechatronic systems, modeled by GTSs that are similar to ours. However, they only check that the set of non-bad configurations is an inductive invariant. That worked for their application, but for verifying safety properties in general it requires the user to supply an inductive invariant. Bauer and Wilhelm [7, 6] use *partner abstraction* to verify dynamic communication systems; two nodes are not distinguished if they have the same labels and the sets of labels of their adjacent nodes are equal, respecting edge directions. That abstraction is not suited for ad hoc protocols, because nodes do not have dedicated roles.

Backward reachability analysis has also been used to verify safety properties in many parameterized and infinite-state system models, with less general connection patterns than those possible in GTSs. Examples include totally homogeneous topologies in which nodes can not identify different partners, resulting in Petri nets with variants (e.g., [15]), systems with linear structure and some extensions (e.g., [1]), and systems with binary connections between nodes, tailored for modeling telephone services [19].

Organization of paper. We give a brief outline of the DYMO protocol in Section 2. The graph transformation system formalism and the backward reachability procedure are presented in Sections 3 and 4. In Section 5 we describe how we modeled DYMO, and present our verification results in Section 6. Finally, Section 7 concludes the paper.

2 DYMO

We are interested in modeling and verification of ad hoc routing protocols. These protocols are used in networks that vary dynamically in size and topology. Every network node that participates in an ad hoc routing protocol acts as a router, using forwarding information stored in a routing table. The purpose of the ad hoc routing protocol is to dynamically update the routing tables so that they reflect the current network topology. DYMO [11] is one of two ad hoc routing protocols currently considered for standardization by the IETF MANET group [24]. The latest DYMO version at the time of writing is specified in version 10 of the DYMO Internet draft [12]. This is the version we have used as basis for our modeling. The following is a simplified description of the main properties of DYMO. The reader is referred to the Internet draft for the details.

In our protocol model, each DYMO network node A has an address, a routing table and a sequence number. The sequence number of A is included in routing messages originating from A, as a measure of freshness, and is incremented for each such message. The routing table of A contains the following fields for each destination node D.

- $\texttt{RouteNextHopAddress}_A(D)$ is the node to which A currently forwards packets, destined for node D.
- $\texttt{RouteSeqNo}_A(D)$ is the sequence number that node A has recorded for the route to destination D. It is a measure of the freshness of a route; a higher number means more recent routing information. Note that this sequence number concerns the route to D from A, and is not related to the sequence number of A.
- $\texttt{RouteHopCnt}_A(D)$ is the recorded distance from A to node D, in terms of number of hops.
- $\texttt{Broken}_A(D)$ is an indicator of whether or not the route from A to D can be used. The protocol has a mechanism to detect when a link on a route is broken [12]. Information regarding broken links is propagated through route error messages (`RERR`).

When a network node A wants to send a packet to another network node D, it first checks its routing table to see if it has an entry with $\texttt{Broken}_A(D) = \mathit{false}$. If that is the case, it forwards the packet to node $\texttt{RouteNextHopAddress}_A(D)$. Otherwise, node A needs to find a route to D, which it does by issuing a route request (`RREQ`) message. The route request is flooded through the network. It contains the addresses of nodes A and D, the sequence number of A, and a hop counter. The hop counter contains the value 1 when the `RREQ` is issued; each retransmitting node then increases it by one. Node A increases its own sequence number after each issued route request.

When the destination of a route request, D, receives it, it generates a route reply message (`RREP`). The route reply contains the same fields as the request. Route replies are not flooded, but instead routed through the network using available routing table entries. `RREP`s and `RREQ`s are collectively referred to as routing messages (RMs).

Whenever a network node A receives an RM, the routing table of A is compared to the RM. If A does not have an entry pertaining to the originator of the RM, then the information in the RM is inserted into the routing table of A. Otherwise, the information in the RM replaces that of the routing table if the information is more recent, or equally recent but better, in terms of distance to the originator. The routing table update rules are detailed in Section 5.

3 Modeling Using Graph Transformation Systems

We model systems as transition systems of a particular form, in which configurations are hypergraphs, and transitions between configurations are specified by graph rewriting rules. Constraints on configurations are represented by so-called *patterns*, which are hypergraphs extended with a mechanism to describe the absence of certain hyperedges: *negative application conditions (NACs)*. Our definitions are similar to the ones used by, e.g., Becker et al. [8], but with a more general facility for expressing NACs.

Assume a finite set Λ of *labels*. A *hypergraph* is a pair $\langle N, E \rangle$, where N is a finite set of *nodes*, and $E \subseteq \Lambda \times N^*$ is a finite set of *hyperedges*. A hyperedge is a pair $(\lambda, \overrightarrow{n})$, where $\lambda \in \Lambda$ is its *label* and $\overrightarrow{n} \in N^*$. The length of $\overrightarrow{n}$ is called the *arity* of the hyperedge. A hyperedge is essentially a relation on nodes, and can be visualized as a box labeled λ, with connections to each node $n \in \overrightarrow{n}$.

A *pattern* is a tuple $\varphi = \langle N_\varphi, E_\varphi, \mathcal{G}_\varphi^- \rangle$, where $\langle N_\varphi, E_\varphi \rangle$ is a hypergraph, and $\mathcal{G}_\varphi^-$ is a set of NACs. Each *NAC* is a hypergraph $G^- = \langle N^-, E^- \rangle$, where N^- is a finite set of *negative nodes* disjoint from N_φ, and $E^- \subseteq \Lambda \times (N_\varphi \cup N^-)^*$ is a finite set of *negative hyperedges*. Further, for any two NACs $G_1^- = \langle N_1^-, E_1^- \rangle$, $G_2^- = \langle N_2^-, E_2^- \rangle$, we have the constraint that $N_1^- \cap N_2^- = \emptyset$. We refer to N_φ and E_φ as *positive* nodes and edges of φ. We define $\mathcal{N}(E) = \{n \in \overrightarrow{n} \mid (\lambda, \overrightarrow{n}) \in E\}$.

Example. Figure 1 shows a pattern — the left-hand side of one of the DYMO model routing table update rules. The pattern models a network node receiving routing information for a node to which it currently has no route. In the pattern, positive nodes are drawn as circles and negative nodes as double circles. Nodes have numeric names for identification. Positive and negative edges are drawn as boxes and double boxes. Edge connections are numbered, to indicate their order. The pattern contains a single NAC, consisting of the negative edges labeled `RouteEntry` and `RouteAddress` along with their connected nodes. Without the possibility to express non-existence, we would need to model traversal through the entries to conclude the absence of an entry. In more detail, the pattern consists of a network node A (node 3) and a routing message (node 1). A has a routing table (node 4) that contains no routing table entry pointing to network node D (node 6). The message has originator D, a hop count (node 7), a sequence number (node 5) and an IP source (node 2).

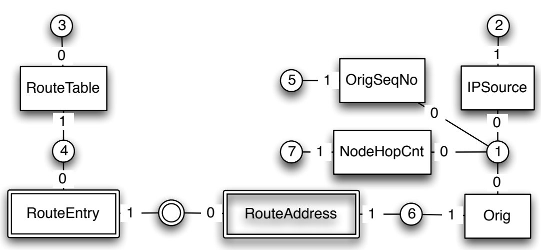


Fig. 1. A pattern containing a NAC.

A hypergraph $g = \langle N_g, E_g \rangle$ is *subsumed by* a pattern $\varphi = \langle N_\varphi, E_\varphi, \mathcal{G}_\varphi^- \rangle$, written $g \preceq \varphi$, if there exists an injection $h : N_\varphi \to N_g$ satisfying:

1. for each $(\lambda, \overrightarrow{n}) \in E_\varphi$ we have $(\lambda, h(\overrightarrow{n})) \in E_g$ and
2. there exists no $\langle N^-, E^- \rangle \in \mathcal{G}_\varphi^-$ and no injection $k : N^- \to N_g$ such that $(\lambda, (h \cup k)(\overrightarrow{n})) \in E_g$ for each $(\lambda, \overrightarrow{n}) \in E^-$, where $(h \cup k)$ is defined as h on N_φ and as k on N^-.

Intuitively, a pattern $\varphi = \langle N_\varphi, E_\varphi, \mathcal{G}_\varphi^- \rangle$ is a constraint, saying that a hypergraph must contain $\langle N_\varphi, E_\varphi \rangle$ as a subgraph, which does not have a "match" for any NAC in $\mathcal{G}_\varphi^-$.

Above we let $f((n_1, \ldots, n_k)) = (f(n_1), \ldots, f(n_k))$ for a function on nodes applied to a vector of nodes. If an injection h satisfying the above conditions exists, we say that $g \preceq \varphi$ is *witnessed by* h, written $g \preceq_h \varphi$.

For a pattern φ we use $[\![\varphi]\!]$ to denote the set of hypergraphs g such that $g \preceq \varphi$. For a set of patterns Φ, we let $[\![\Phi]\!] = \cup\{[\![\varphi]\!] \mid \varphi \in \Phi\}$. We call pattern $\varphi = \langle N_\varphi, E_\varphi, \mathcal{G}_\varphi^- \rangle$ *consistent* if $\langle N_\varphi, E_\varphi \rangle \preceq \varphi$. Informally, φ is consistent if none of its NACs contradicts its positive nodes and edges. An inconsistent pattern ψ represents an empty set, as $g \preceq \psi$ is not satisfied by any g.

A pattern φ is subsumed by the pattern ψ, denoted $\varphi \preceq \psi$, if $[\![\varphi]\!] \subseteq [\![\psi]\!]$. The relation $\preceq$ on patterns can be checked according to the following Proposition.

Proposition 1. *Given patterns $\varphi = \langle N_\varphi, E_\varphi, \mathcal{G}_\varphi^- \rangle$ and $\psi = \langle N_\psi, E_\psi, \mathcal{G}_\psi^- \rangle$ which are consistent, we have that $\varphi \preceq \psi$ iff there exists an injection $h : N_\psi \to N_\varphi$ such that $\langle N_\varphi, E_\varphi \rangle \preceq_h \langle N_\psi, E_\psi, \emptyset \rangle$ and for each NAC $\langle M^-, F^- \rangle \in \mathcal{G}_\psi^-$ there is a NAC $\langle N^-, E^- \rangle \in \mathcal{G}_\varphi^-$ and an injection $k : N^- \to M^-$ such that*

- *$(\mathcal{N}(E^-) \setminus N^-) \subseteq h(N_\psi)$, and*
- *for each $(\lambda, \overrightarrow{n}) \in E^-$, we have $(\lambda, (h^{-1} \cup k)(\overrightarrow{n})) \in F^-$.* □

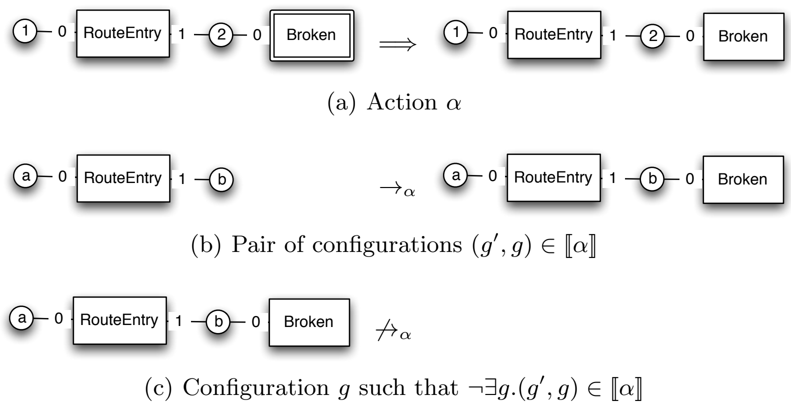
Intuitively, $\varphi \preceq \psi$ if and only if the positive part of ψ is a subgraph of the positive part of φ, and for each NAC in $\mathcal{G}_\psi^-$, there is a corresponding NAC in $\mathcal{G}_\varphi^-$ which is a subgraph of the former NAC.

In our system model, configurations are represented by hypergraphs. Transitions are specified by *actions*, which are (hypergraph) rewrite rules.

Definition 1. *An* action *is a pair $\langle L, R \rangle$, where $L = \langle N_L, E_L, \mathcal{G}_L^- \rangle$ is a pattern and $R = \langle N_R, E_R \rangle$ is a hypergraph with $N_L \subseteq N_R$ (i.e., actions can create nodes, but not delete them). The action $\alpha = \langle L, R \rangle$ denotes the set $[\![\alpha]\!]$ of pairs of configurations (g', g), with $g' = \langle N_{g'}, E_{g'} \rangle$, $g = \langle N_g, E_g \rangle$ and $N_{g'} \subseteq N_g$ such that there is an injection $h : N_R \to N_g$ satisfying:*

- *$g' \preceq L$ is witnessed by the restriction of h to N_L*
- *$N_g = N_{g'} \cup h(N_R)$*
- *$E_g = (E_{g'} \setminus h(E_L)) \cup h(E_R)$.* □

Example. Figure 2(a) shows an action $\alpha = \langle L, R \rangle$. The pattern L is to the left of the arrow ($\Longrightarrow$) and R to the right. The action does not create any nodes, i.e., $N_L = N_R$. Figure 2(b) shows a pair $(g', g) \in [\![\alpha]\!]$, i.e., g' can be rewritten via α to g. The subsumption $g' \preceq L$ is witnessed by the injection $h = \{1 \mapsto a, 2 \mapsto b\}$. The injection h satisfies $N_g = N_{g'} \cup h(N_R) = \{a, b\}$ and $E_g = (E_{g'} \setminus h(E_L)) \cup h(E_R) = h(E_R)$. Figure 2(c) shows a configuration g' such that there is no g with $(g', g) \in [\![\alpha]\!]$, since $g' \not\preceq L$. In other words, g' cannot be rewritten via α.



(a) Action α

(b) Pair of configurations $(g', g) \in [\![\alpha]\!]$

(c) Configuration g such that $\neg\exists g.(g', g) \in [\![\alpha]\!]$

Fig. 2. Example of an action and its semantics.

Definition 2. *A* system model *is a pair $\langle\gamma_0, \mathcal{A}\rangle$ consisting of an initial configuration γ_0 together with a finite set of actions $\mathcal{A}$.* $\square$

For a set Γ of configurations and an action α, let $pre(\alpha, \Gamma) = \{g' \mid \exists g \in \Gamma.\ (g', g) \in [\![\alpha]\!]\}$, i.e., the configurations which in one step can be rewritten to Γ using α. Similarly, for a set of actions $\mathcal{A}$, let $pre^*(\mathcal{A}, \Gamma)$ denote the set of configurations which can reach a configuration in Γ by a sequence of rewritings using actions in $\mathcal{A}$.

4 Symbolic Verification

We formulate a verification scenario as the problem whether a set of configurations, represented by a set of patterns, is reachable. More precisely, given a system model $\langle\gamma_0, \mathcal{A}\rangle$, and a set of patterns Φ, the *reachability problem* asks whether there is a sequence of transitions from γ_0 to some configuration in $[\![\Phi]\!]$.

In our approach, we analyze a reachability problem using *backward reachability analysis*, in which we compute an over-approximation of the set $pre^*(\mathcal{A}, [\![\Phi]\!])$ of configurations, and check whether it includes γ_0. We clarify why and when the computation is not exact in the *Approximation* paragraph below. In general, the reachability problem is undecidable, and our analysis is not guaranteed to terminate. However, the technique is sufficiently powerful to verify several nontrivial network protocols (see Section 6).

We attempt to compute $pre^*(\mathcal{A}, [\![\Phi]\!])$ by standard fixed point iteration, using predecessor computation, as shown in Procedure 1. In the procedure, V and W are sets of patterns whose predecessors already have (V) and have not (W) been computed. In each iteration of the while loop, we choose a pattern φ from W. If $\gamma_0 \in [\![\varphi]\!]$ then we have found a (possibly spurious) path from γ_0 to $[\![\Phi]\!]$. Otherwise, we check whether φ is redundant, meaning that it is subsumed by some other pattern which will be or has been explored. If not, we add to W a

Procedure 1 Backward Reachability Analysis

Require: System model $\langle \gamma_0, \mathcal{A} \rangle$ and a set Φ of (bad) patterns
Ensure: If terminates; answers whether a configuration in $[\![\Phi]\!]$ is reachable from γ_0

1 $V := \emptyset$, $W := \Phi$
2 **while** $W \neq \emptyset$ **do**
3   **choose** $\varphi \in W$
4     $W := W \setminus \{\varphi\}$
5   **if** $\gamma_0 \in [\![\varphi]\!]$ **then**
6   **return** "Reachable"
7   **if** $\forall \psi \in (V \cup W).\ \neg(\varphi \preceq \psi)$ **then**
8         $V := V \cup \{\varphi\}$
9   **for each** $\alpha \in \mathcal{A}$ **do**
10             $W := W \cup \textsc{Pre}(\alpha, \varphi)$
11 **return** "Unreachable"

set of patterns over-approximating $pre(\mathcal{A}, [\![\varphi]\!])$. As a further optimization, not shown in Procedure 1, at line 7 we also remove patterns from V and W that are subsumed by φ; keeping V and W small speeds up the procedure.

The central part of Procedure 1 is the (nontrivial) computation of predecessors of a pattern; it is done as in Procedure 2, whose description follows. Procedure 2 terminates on any input, as all loops are finite.

Procedure 2 $\textsc{Pre}(\alpha, \varphi)$

Require: Action $\alpha = \langle L, R \rangle$, pattern $\varphi = \langle N_\varphi, E_\varphi, \mathcal{G}^-_\varphi \rangle$
Ensure: Φ is a set of patterns satisfying $pre(\alpha, [\![\varphi]\!]) \subseteq [\![\Phi]\!]$

$\Phi := \emptyset$
Rename all nodes in φ, so that they are disjoint from the nodes in L and R
for each partial injection $h : N_R \to N_\varphi$ **do**
    Rename each node $h(n)$ in the range of h to n
    if $\exists\, n \in \mathcal{D}omain(h) - N_L\ .\ \mathcal{E}_+(n, \varphi) \not\subseteq \mathcal{E}_+(n, R)\ \vee$
    $\mathcal{I}nconsistent(\varphi + R)$ **then**
        skip
    else
        $\varphi' := (\varphi \ominus_\alpha R) + L$
        for each $G^- \in \mathcal{G}^-_\varphi$ **do**
            if $\mathcal{I}nconsistent((L \ominus_E R) + G^-)$ **then**
                $\varphi' = \varphi' - G^-$
        if $\neg\mathcal{I}nconsistent(\varphi')$ **then**
            $\Phi := \Phi \cup \varphi'$
return Φ

Let a *partial injection*, or *matching*, from a set N to a set N' be an injection from a nonempty subset of N to N'. For two patterns $\varphi = \langle N_\varphi, E_\varphi, \mathcal{G}^-_\varphi \rangle$ and $\psi = \langle N_\psi, E_\psi, \mathcal{G}^-_\psi \rangle$, we use $\varphi + \psi$ to denote $\langle N_\varphi \cup N_\psi, E_\varphi \cup E_\psi, \mathcal{G}^-_\varphi \cup \mathcal{G}^-_\psi \rangle$.

When adding patterns, if the node and edge sets are not disjoint, the result is a "merge". No automatic renaming is assumed.

We use the following two subtraction operations in Procedure 2. First, for a pattern $\varphi = \langle N_\varphi, E_\varphi, \mathcal{G}^-_\varphi \rangle$, and an action $\alpha = \langle L, R \rangle$, let $\varphi \ominus_\alpha R$ be the pattern $\psi = \langle N_\psi, E_\psi, \mathcal{G}^-_\varphi \rangle$, with $E_\psi = E_\varphi \setminus E_R$ and $N_\psi = N_\varphi \setminus (N_R \setminus N_L)$. Second, for a pattern $\varphi = \langle N_\varphi, E_\varphi, \mathcal{G}^-_\varphi \rangle$, and a hypergraph $g = \langle N_g, E_g \rangle$, let $\varphi \ominus_E g$ be the pattern $\psi = \langle N_\psi, E_\psi, \mathcal{G}^-_\varphi \rangle$, with $E_\psi = E_\varphi \setminus E_g$ and $N_\psi = \mathcal{N}(E_\psi)$.

For a NAC G^-, we use $\varphi + G^-$ to denote $\langle N_\varphi,\, E_\varphi,\, \mathcal{G}^-_\varphi \cup G^- \rangle$ and $\varphi - G^-$ to denote $\langle N_\varphi,\, E_\varphi,\, \mathcal{G}^-_\varphi \setminus G^- \rangle$. If $n \in N_\varphi$, let $\mathcal{E}_+(n, \langle N_\varphi, E_\varphi, \mathcal{G}^-_\varphi \rangle)$ denote the set of edges in E_φ connected to n.

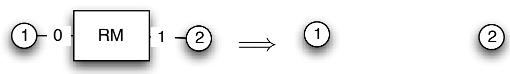
Procedure 2 first renames the nodes (line 2) to avoid unintended node collisions between φ and α. Thereafter, the loop starting at line 3 performs a sequence of operations for each possible matching between some nodes of N_R and N_φ.

On line 4 each node $h(n)$ in the range of h is renamed to n, in order to "merge" R and φ according to h. Since nodes that are created by α must also have all their edges created by α, we should discard matchings which violate this (line 5). On line 5 we also discard inconsistent matchings. The check $\mathcal{I}nconsistent(\varphi)$ is true iff pattern φ is not consistent.

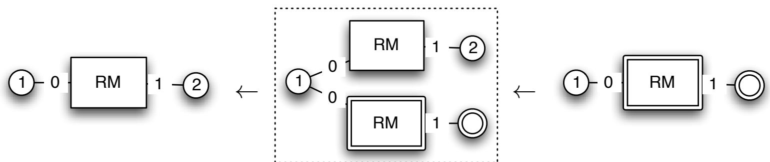
On line 8 the action α is "executed" backwards to obtain a pattern φ' that is a potential predecessor of φ. Using the special subtraction $\ominus_\alpha$ nodes and edges created by α are removed from φ. On lines 9–11, we remove all NACs from φ' which contradict subgraphs removed by α. The backward execution (on line 8) and the NAC removal may both introduce approximation (see the paragraph below). Since by definition α cannot remove nodes, we use the special subtraction $\ominus_E$ which ignores nodes not connected to edges. On line 12, we discard the resulting predecessor pattern if it is inconsistent — this can happen if a NAC in L contradicts a positive subgraph of φ'. Finally, if we reach line 13, we have found a predecessor pattern, which is added to Φ.

Approximation. The predecessor computation in Procedure 2 may introduce an approximation at line 8 or line 11. If α removes a hyperedge $(\lambda, \overrightarrow{n})$ of arity zero or one, and this edge is included in φ, then $pre(\alpha, [\![\varphi]\!])$ should contain two copies of $(\lambda, \overrightarrow{n})$, representing "two or more" occurrences of $(\lambda, \overrightarrow{n})$. However, the use of sets to contain the hyperedges of patterns results in $\textsc{Pre}(\alpha, \varphi)$ containing only one copy of $(\lambda, \overrightarrow{n})$, after line 8, representing "one or more" occurrences of $(\lambda, \overrightarrow{n})$.

Further, if α removes a subgraph which is forbidden by φ, then $pre(\alpha, [\![\varphi]\!])$ should say that there is exactly one subgraph of this form. However, patterns cannot always express "exactly one" occurrence of a subgraph. In this situation, Procedure 2 therefore lets the resulting pattern say that "there is at least one occurrence" of this subgraph. As an example, consider the simple situation in Figure 3, where α, shown in Figure 3(a), removes an `RM`-edge between two nodes, and φ, the rightmost pattern in Figure 3(b), says that there is no `RM`-edge. The exact predecessor of φ is: "there is exactly one `RM`-edge between two nodes". However, the resulting predecessor (the leftmost pattern in Figure 3(b)) represents that there is *at least* one `RM`-edge connected to graph node 1. To illustrate the effect of lines 9–11 of Procedure 2, an intermediate pattern, where the contradiction has not yet been resolved, is shown in Figure 3(b).



(a) Idealized action α



(b) Predecessor computation showing intermediate pattern

Fig. 3. Approximation due to upwards-closure.

Optimizations. To make the analysis more efficient, we have (implemented) two mechanisms for the user to specify simple type constraints. One is to annotate nodes with types that are respected in the analysis, with the semantics that nodes may only "match" nodes of same type. Another is to add patterns that describe multiplicity constraints on edges. For example, our DYMO models use "a network node can have at most one routing table", by specifying a pattern where a node has two routing tables as "impossible".

We need to model integer-valued variables, as DYMO uses sequence numbers and hop counts. This is done by representing integers as nodes, and greater than ($>$) and equality ($=$) relations as edges between these nodes. We do not represent concrete integer values. Hence, we cannot compare integers which are not connected by a relational edge. We have extended our tool to handle the transitive closure of $>$ and $=$, as part of the predecessor computation. For each predecessor pattern generated, the closure of all transitive numerical relations present in the pattern is computed. New relational edges are then added to the pattern accordingly. The reason is that our syntactic subsumption check cannot deduce such semantic information about relations. The check for created nodes on line 5 of Procedure 2 was also extended to take into account the transitivity of numerical relations.

5 Modeling and Verification of DYMO

In this section we describe how we modeled the DYMO protocol (more precisely, the latest version at the time of writing, version 10 [12], and version 5). See our project home page [16] for the complete models. In total, our DYMO v10 model consists of one initial graph ("an empty network") and 77 actions. Of these, 38 actions model routing table update rules, similar to the one in Figure 4 below. We have only used unary and binary hyperedges in our models, although our implementation supports hyperedges of any arity.

Modeling network topology and message transmission. We represent arbitrary network topologies by letting the initial system configuration be an empty network (i.e., an empty graph), and including an action for creating an arbitrary network node; thus any initial topology can be formed. We do not explicitly model connectivity in the network. Instead all nodes can potentially react on all messages in the network; this reaction on a message can be postponed indefinitely, corresponding to a node being out of range or otherwise incapable of receiving the message. Messages can also be non-deterministically removed, corresponding to message loss. In our modeling of message transmission, messages are left in the network after a node has handled them (until they are potentially dropped): this accounts for messages being duplicated.

Handling of timeouts and hop limits. DYMO uses timeouts to determine if a RREQ should be retransmitted, if a link is broken, or if a routing table entry should be removed. We over-approximate timeouts as "event x can happen at any time", which covers all possibilities for a timeout. It is known from previous work on the AODV protocol [9], that if entries are removed from the routing table, loops may form. The reason is that obsolete information can then be accepted. In DYMO, routing table entries are invalidated (set to broken) after some time, and later removed; temporary loops are thus tolerated. We exclude removal of routing table entries from our analysis; they can only be invalidated. In practice, we thus verify loop-freedom under the assumption that routing table entries are kept "long enough".

We do not model DYMO hop limits [12], used to limit packet traversal. However, since we include actions for arbitrary dropping of RMs and RERRs, we implicitly cover all possible hop limit settings.

Routing table update rules. The DYMO specification [12] prescribes when a node should update its own routing table upon receiving routing data, i.e., when received routing data should replace existing data. Existing data is represented by a routing table entry, with fields `RouteSeqNo`, `RouteHopCnt`, and `Broken`. Received data is represented by a routing message with fields `OrigSeqNo`, `NodeHopCnt` and message type `RM` – either a route request (`RREQ`) or a route reply (`RREP`). The table entry should be updated in the following cases:

1. $\texttt{OrigSeqNo} > \texttt{RouteSeqNo}$
2. $\texttt{OrigSeqNo} = \texttt{RouteSeqNo} \wedge \texttt{NodeHopCnt} < \texttt{RouteHopCnt}$
3. $\texttt{OrigSeqNo} = \texttt{RouteSeqNo} \wedge \texttt{NodeHopCnt} = \texttt{RouteHopCnt} \wedge \texttt{RM} = \texttt{RREP}$
4. $\texttt{OrigSeqNo} = \texttt{RouteSeqNo} \wedge \texttt{NodeHopCnt} = \texttt{RouteHopCnt} \wedge \texttt{Broken}$

The rules say that an update is allowed if (1) the message has a higher sequence number for the destination, or (2) the message has the same sequence number, but a shorter route, or (3) the message has the same routing metric value, and the message is a route reply, or (4) the table entry is broken. See Figure 4 for an illustration of how we model the update rules. The figure corresponds to rule (2). In our framework, we have to model each combination of network nodes used in the rules, such as when `IPSource` equals `Orig`, or `RouteNextHopAddress` equals `RouteAddress`, etc., as separate actions; however, we have tool support for doing this.

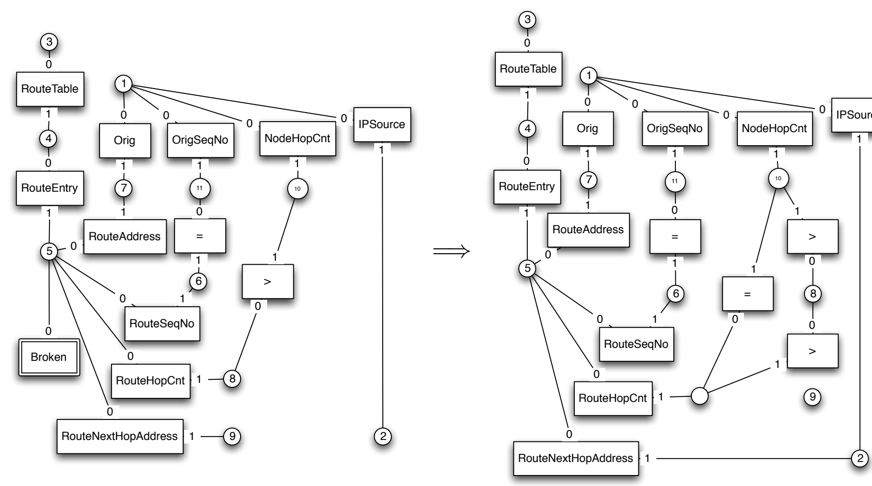


Fig. 4. Action modeling a routing table update.

Formalizing the non-looping property. A central property of ad hoc routing protocols is that they never cause routing loops, as a routing loop prevents a packet from reaching its intended destination. A routing loop is a nonempty finite sequence of nodes $n_1, \ldots, n_k$ such that for some destination D it holds that for all $i : 1 \leq i \leq k$ node $n_{(i+1)(mod\ k)}$ is the next hop towards D from node n_i, and $n_i \neq D$.

We define the ordering $<_D$ on nodes in a configuration as: $n <_D n'$ iff $\texttt{RouteSeqNo}_n(D) > \texttt{RouteSeqNo}_{n'}(D) \vee (\texttt{RouteSeqNo}_n(D) = \texttt{RouteSeqNo}_{n'}(D) \wedge \texttt{RouteHopCnt}_n(D) < \texttt{RouteHopCnt}_{n'}(D))$. There can be no routing loops towards a destination D, if each hop from a node n towards D goes to a node n' with $n' <_D n$. Since $<_D$ is a partial order, any routing path towards D can contain a node at most once. The same ordering was used in the proof of loop freedom for AODV in [9]. The following property, LP, implies the pairwise ordering along routing paths; if LP is invariant for DYMO, there are no routing loops.

$$\begin{array}{c|c} \begin{array}{c} \forall A, B, D \\ A \neq B, B \neq D, \\ A \neq D \end{array} & \quad \texttt{RouteNextHopAddress}_A(D) = B \implies B <_D A \end{array} \qquad (LP)$$

By negating the loop property (LP), we obtain a characterization of the bad system configurations. Loops may thus form if the sequence number strictly decreases, or the sequence number stays the same but the hop count does not decrease, between a node A and its next hop B on a route towards a destination node D. In our verification of DYMO, we verify unreachability for a set of six bad patterns. Three represent a disjunct of $(\neg LP)$ under quantification; two represent a network node with a routing table entry pointing to the node itself; and one pattern represents that a node has a next hop (which is not D) towards some destination D, but the next hop has no entry for D. As an example, a pattern representing one of the disjuncts of $(\neg LP)$ is shown in Figure 5.

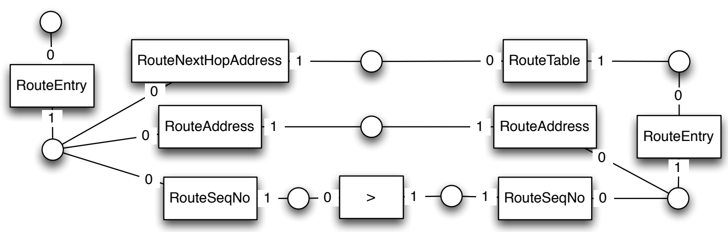


Fig. 5. Graph pattern representing a set of bad system configurations in DYMO.

6 Experimental Results

We have modeled and verified the DYMO protocol as described in Sections 5 and 4. Recall that the analysis is under an assumption of routing table entries not being removed. The analysis has been performed using our tool GBT (Graph Backwards Tool). GBT and the models are available at our project home page [16]. The tool uses the `.dot` format for describing hypergraphs and patterns (input and output). If the initial configuration can be reached, an error trace, showing a sequence of actions leading to one of the bad patterns, is provided. Note that this trace may be spurious, due to over-approximation.

We have verified the latest DYMO version at the time of writing, namely version 10 of the Internet draft [12], as well as an older draft (version 5). Our results are presented in Table 1. In the “dest. reply” models, only the destination node replies to an RREQ, whereas in “interm. reply”, intermediate nodes may also reply (in case they have a fresh enough route, see [12]). Column *Actions* contains the number of actions in the model. *Checked* contains the total number of non-impossible patterns generated by the predecessor computation, plus the ones given as input. *Covered* contains the patterns which were subsumed (see Section 4). *Left* contains the patterns left after the analysis has finished; none of them contain the initial graph. *Time* contains the total verification time (GBT start to end) on a machine with an AMD Opteron 2220 2.8 GHz processor.

Table 1. Measurement results from using GBT.

Protocol	Actions	Checked	Covered	Left	Verified	Time
DYMO draft 10						
- dest. reply	56	160681	160671	10	Yes	52 min 42 s
- interm. reply	77	254620	254610	10	Yes	1h 59 min
DYMO draft 05	50	119506	119496	10	Yes	39 min 20 s
Pub/priv srv I	12	367	360	7	Yes	0.18 s
Pub/priv srv II	13	493	484	9	Yes	0.25 s
Firewall I	6	128	125	3	Yes	0.09 s
Firewall II	6	128	125	3	Yes	0.09 s

In Table 1 we have also included GBT verification results for the "Public/private servers" and "Firewall" examples, used by König and Koziοura [21]. These examples required modifications to work with our tool. The abstraction introduced from using sets to contain the hyperedges of patterns required us to add a zero arity edge to the right hand side of two actions in "Public/private servers II". The transitivity handling in our tool was also extended to include communication channels.

7 Conclusions and Future Work

We have described and implemented a general framework for modeling and verification of protocols using a variant of graph transformation systems, and applied it to automatically prove loop freedom of the DYMO v10 ad hoc routing protocol. We expect that several of the actions used in our DYMO model need only small modifications to work for other ad hoc routing protocols categorized as reactive (i.e., on-demand). The reason is that reactive ad hoc routing protocols generally use the same kind of flooding route discovery mechanism; examples include AODV[23], DSR[18], and LUNAR[25] (see [22] for an extensive list).

As GTSs with NACs make up quite a generic modeling framework, there should be possibilities for interesting case studies, and further development. Directions for future work include further optimizations of the predecessor computation, e.g., by early detection of unfruitful matchings. We are currently working on a new DYMO model, to investigate the effect on run-time performance when using hyperedges of arity greater than two. Termination of the reachability analysis can be obtained by bounding and truncating the generated patterns, at the cost of over-approximation, e.g., by enforcing a maximum size. The possibility of spurious counter-examples, due to approximations in the predecessor computation, motivates looking at counter-example guided abstraction refinement.

Acknowledgments. We would like to thank Barbara König for valuable help on the Augur tool and related issues. We also thank Parosh Abdulla, Joachim Parrow, and the anonymous referees for their many helpful comments.

References

1. P. Abdulla, G. Delzanno, N. Ben Henda, and A. Rezine. Regular model checking without transducers. In *Proc. TACAS '07,* 13^{th} *Int. Conf. on Tools and Algorithms for the Construction and Analysis of Systems*, 2007.
2. P. Abdulla, B. Jonsson, M. Nilsson, and M. Saksena. A survey of regular model checking. In *Proc. CONCUR 2004,* 15^{th} *Int. Conf. on Concurrency Theory.* Springer Verlag, 2004.
3. P. A. Abdulla, K. Čerāns, B. Jonsson, and T. Yih-Kuen. Algorithmic analysis of programs with well quasi-ordered domains. *Information and Computation*, 160:109–127, 2000.
4. P. A. Abdulla and B. Jonsson. Verifying programs with unreliable channels. *Information and Computation*, 127(2):91–101, 1996.
5. M. Abolhasan, T. Wysocki, and E. Dutkiewicz. A review of routing protocols for mobile ad hoc networks. *Ad Hoc Networks*, 2(1):1–22, January 2004.
6. J. Bauer. *Analysis of Communication Topologies by Partner Abstraction.* PhD thesis, Universität des Saarlandes, 2006.
7. J. Bauer and R. Wilhelm. Static Analysis of Dynamic Communication Systems. In *14th International Static Analysis Symposium.* Springer, 2007.
8. B. Becker, D. Beyer, H. Giese, F. Klein, and D. Schilling. Symbolic invariant verification for systems with dynamic structural adaptation. In *Proc. ICSE '06,* 28^{th} *Int. Conf. on Software Engineering*, pages 72–81. ACM Press, 2006.
9. K. Bhargavan, D. Obradovic, and C. A. Gunter. Formal verification of standards for distance vector routing protocols. *Journal of the ACM*, 49(4):538–576, 2002.
10. J. Broch, D. A. Maltz, D. B. Johnson, Y.-C. Hu, and J. Jetcheva. A performance comparison of multi-hop wireless ad hoc network routing protocols. In *Proceedings of MobiCom'98*, October 1998.
11. I. D. Chakeres and C. E. Perkins. DYMO - Dynamic MANET On-demand Routing Protocol home page. `http://www.ianchak.com/dymo/`, May 2007.
12. I. D. Chakeres and C. E. Perkins. Dynamic MANET On-demand (DYMO) Routing. Internet draft, July 2007. draft-ietf-manet-dymo-10.txt.
13. E. M. Clarke, O. Grumberg, and D. Long. Model checking and abstraction. In *Proc.* 19^{th} *ACM Symp. on Principles of Programming Languages*, 1992.
14. S. Das and D. L. Dill. Counter-example based predicate discovery in predicate abstraction. In *Proc. FMCAD '02,* 4^{th} *Int. Conf. on Formal Methods in Computer-Aided Design*, pages 19–32. Springer, 2002.
15. J. Esparza, A. Finkel, and R. Mayr. On the verification of broadcast protocols. In *Proc. LICS' 99* 14^{th} *IEEE Int. Symp. on Logic in Computer Science*, 1999.
16. GBT - Graph Backwards Tool project home page. `http://www.it.uu.se/research/group/mobility/adhoc/gbt`.
17. G. Holzmann. The model checker SPIN. *IEEE Trans. on Software Engineering*, SE-23(5):279–295, May 1997.
18. D. B. Johnson, D. A. Maltz, and J. Broch. DSR: The dynamic source routing protocol for multi-hop wireless ad hoc networks. In *Ad Hoc Networking*, chapter 5, pages 139–172. Addison-Wesley, 2001.
19. B. Jonsson and L. Kempe. Verifying safety properties of a class of infinite-state distributed algorithms. In *Proc.* 7^{th} *Int. Conf. on Computer Aided Verification*, pages 42–53. Springer Verlag, 1995.
20. H. Kastenberg and A. Rensink. Model checking dynamic states in GROOVE. In *SPIN Workshop*, pages 299–305. Springer, 2006.

21. B. König and V. Kozioura. Counterexample-guided abstraction refinement for the analysis of graph transformation systems. In *Proc. TACAS '06, 12^{th} Int. Conf. on Tools and Algorithms for the Construction and Analysis of Systems*, pages 197–211. Springer, 2006.
22. List of ad-hoc routing protocols - Wikipedia, the free encyclopedia. `http://en.wikipedia.org/wiki/Ad_hoc_routing_protocol_list`.
23. C. E. Perkins and E. M. Belding-Royer. Ad-hoc on-demand distance vector routing. In *Proc. 2^{nd} Workshop on Mobile Computing Systems and Applications (WMCSA '99)*, pages 90–100. IEEE Computer Society, 1999.
24. The official IETF MANET working group web page. `http://www.ietf.org/html.charters/manet-charter.html`.
25. C. Tschudin, R. Gold, O. Rensfelt, and O. Wibling. LUNAR: a lightweight underlay network ad-hoc routing protocol and implementation. In *Proc. Next Generation Teletraffic and Wired/Wireless Advanced Networking (NEW2AN)*, February 2004.
26. L. D. Zuck and A. Pnueli. Model checking and abstraction to the aid of parameterized systems (a survey). *Computer Languages, Systems & Structures*, 30(3–4):139–169, 2004.

A Proofs

We prove that our backward reachability analysis (Procedure 1) is correct.

In practice, the analysis has to use the syntactic characterization of $\preceq$ to check subsumption of patterns, as stated in Proposition 1. Thus, we will have to prove Proposition 1 .

The underlying assumption in Procedure 1 is that we can throw away patterns φ which are subsumed by some previously seen pattern ψ (see line 7). The motivation is that, since any hypergraph in $[\![\varphi]\!]$ is contained in $[\![\psi]\!]$, it should suffice to take predecessors of ψ . We prove that, indeed, our predecessor computation on patterns (Procedure 2) is adequate — meaning that $\textsc{Pre}(\alpha, \varphi)$ subsumes $\mathit{pre}\,(\alpha, [\![\varphi]\!])$.

Notations and Conventions We will use the following notations and conventions in the proofs.

If an injection h satisfying the conditions of Proposition 1 exists for patterns φ and ψ , we say that $\varphi \preceq \psi$ is *witnessed by* h, written $\varphi \preceq_h \psi$.

An injection $h : N \to N'$ can be applied on any pattern, hypergraph, edge or NAC in a distributed fashion, with the convention that $h(n) = n$ for nodes $n \notin N$. For example, $h(\mathcal{G}^-_\varphi)$ is the set $\{h(G^-_\varphi) \mid G^-_\varphi \in \mathcal{G}^-_\varphi\}$ etc.

For a set of NACs $\mathcal{G}^-_\varphi$ we let $\mathcal{N}(\mathcal{G}^-_\varphi) = \{\mathcal{N}(E^-) \mid \langle N^-, E^-\rangle \in \mathcal{G}^-\}$.

Two sets of NACs $\mathcal{G}^-_\varphi$ and $\mathcal{G}^-_\psi$ are *isomorphic*, written $\mathcal{G}^-_\varphi \simeq \mathcal{G}^-_\psi$, if there exists a bijection $h : \mathcal{N}(\mathcal{G}^-_\psi) \to \mathcal{N}(\mathcal{G}^-_\varphi)$ such that $h(\mathcal{G}^-_\psi) = \mathcal{G}^-_\varphi$ and $h^{-1}(\mathcal{G}^-_\varphi) = \mathcal{G}^-_\psi$.

Let $\mathcal{N}$ be the set of all nodes (negative and positive).

The *identity* between sets of nodes is the bijection $Id : \mathcal{N} \to \mathcal{N}\,;\ h(n) = n$.

A.1 Pattern Subsumption

Proposition 1. *Given patterns* $\varphi = \langle N_\varphi, E_\varphi, \mathcal{G}^-_\varphi\rangle$ *and* $\psi = \langle N_\psi, E_\psi, \mathcal{G}^-_\psi\rangle$ *which are consistent, we have that* $\varphi \preceq \psi$ *iff there exists an injection* $h : N_\psi \to N_\varphi$, *such that (1)* $\langle N_\varphi, E_\varphi\rangle \preceq_h \langle N_\psi, E_\psi, \emptyset\rangle$ *and (2) for each NAC* $\langle M^-, F^-\rangle \in \mathcal{G}^-_\psi$ *there is a NAC* $\langle N^-, E^-\rangle \in \mathcal{G}^-_\varphi$ *and an injection* $k : N^- \to M^-$ *such that*

- $(\mathcal{N}(E^-) \setminus N^-) \subseteq h(N_\psi)$, *and*
- *for each* $(\lambda, \overrightarrow{n}) \in E^-$, *we have* $(\lambda, (h^{-1} \cup k)(\overrightarrow{n})) \in F^-$.

Proof. ("$\Longrightarrow$") Assume that $\varphi \preceq \psi$, i.e., $[\![\varphi]\!] \subseteq [\![\psi]\!]$.

Since φ is consistent, we know that $\langle N_\varphi, E_\varphi\rangle \in [\![\varphi]\!]$, and by assumption, $\langle N_\varphi, E_\varphi\rangle \in [\![\psi]\!]$, so $\langle N_\varphi, E_\varphi\rangle \preceq \psi$. There thus exists h witnessing $\langle N_\varphi, E_\varphi\rangle \preceq \langle N_\psi, E_\psi, \emptyset\rangle$.

Now suppose that every h witnessing $\langle N_\varphi, E_\varphi\rangle \preceq \langle N_\psi, E_\psi, \emptyset\rangle$ fails to satisfy the condition (2). Thus, for every h there exists $G^-_h \in \mathcal{G}^-_\psi$ such that for each $G^-_\varphi \in \mathcal{G}^-_\varphi$ there is no k with $(h^{-1} \cup k)(G^-_\varphi) \subseteq G^-_h$.

Let $H = \{h \mid \langle N_\varphi, E_\varphi\rangle \preceq_h \langle N_\psi, E_\psi, \emptyset\rangle\}$, which contains at least one element, as shown above. For each $h \in H$, we define a hypergraph G^+_h , where $G^+_h =$

$\langle N_h^+, E_h^+\rangle$. Intuitively, G_h^+ is a *positive interpretation* of $G_h^- = \langle N_h^-, E_h^-\rangle$ where the negative edges are interpreted as positive edges with the same labels. More precisely, $N_h^+ = h^+(N_h^-) \cup (h \cup h^+)(\mathcal{N}(E_h^-))$ and $E_h^+ = (h \cup h^+)(E_h^-)$ with the injection $h^+ : N_h^- \to N_h^+$ and $(N_h^+ \cap N_\varphi) = \emptyset$.

Now consider the hypergraph $g_\wr = \langle N_\varphi, E_\varphi\rangle \cup \bigcup\{G_h^+ \mid h \in H\}$. Intuitively, $g_\wr$ consists of the positive part of φ and for each $h \in H$ a part which contradicts a NAC in ψ but not a NAC in φ .

Now note that $g_\wr \preceq \varphi$. In fact, $g_\wr \preceq_h \varphi$ for any $h \in H$ since we do not contradict $\mathcal{G}_\varphi^-$ by our assumption. Moreover, no matter which $h \in H$ we choose, we get $\neg(g_\wr \preceq_h \psi)$, since we map to a part which contradicts $G_h^- \in \mathcal{G}_\psi^-$. But H contains all possible witnesses of $\langle N_\varphi, E_\varphi\rangle \preceq_h \langle N_\psi, E_\psi, \emptyset\rangle$. Hence, $g_\wr \in [\![\varphi]\!] \setminus [\![\psi]\!]$, contradicting $[\![\varphi]\!] \subseteq [\![\psi]\!]$. Therefore, there must exist an h satisfying (1) which also satisfies (2).

("$\Longleftarrow$") Given an injection $h : N_\psi \to N_\varphi$ and a set of injections $k_{G^-} : N^- \to M^-$ — one for each NAC $\langle M^-, F^-\rangle \in \mathcal{G}_\psi^-$ as defined above — satisfying (1) and (2). We show that $[\![\varphi]\!] \subseteq [\![\psi]\!]$.

Consider any $g = \langle N_g, E_g\rangle \in [\![\varphi]\!]$. There exists an injection $h_g : N_\varphi \to N_g$ witnessing $g \preceq \varphi$. Consider also the *composed* injection $h' = h_g \circ h : N_\psi \to N_g$; $n \mapsto h_g(h(n))$. It witnesses $g \preceq \psi$ as we will see.

We have $h'(E_\psi) \subseteq E_g$ since for each $(\lambda, \overrightarrow{n}) \in E_\psi$ there is a corresponding edge $h((\lambda, \overrightarrow{n})) \in E_\varphi$ and again $h_g(h((\lambda, \overrightarrow{n}))) \in E_g$.

Now suppose that $g \not\preceq \psi$ because the second condition (on page 5) fails. Then there exists some NAC $G^- = \langle M^-, F^-\rangle \in \mathcal{G}_\psi^-$ which contradicts g . More precisely, there exists an injection $k' : M^- \to N_g$ such that for each $(\lambda, \overrightarrow{n}) \in F^-$ we have $(\lambda, (h' \cup k')(\overrightarrow{n})) \in E_g$.

But then there is a corresponding NAC $\langle N^-, E^-\rangle \in \mathcal{G}_\varphi^-$ which also contradicts g . To see this, we use the injection $k_{G^-} : N^- \to M^-$ with the properties defined above, and the injection $k'' = k' \circ k_{G^-} : N^- \to N_g$; $n \mapsto k'(k_{G^-}(n))$. Now for each $(\lambda, \overrightarrow{n}) \in E^-$ we have $(\lambda, ((h' \circ h^{-1}) \cup k'')(\overrightarrow{n})) \in E_g$. But this contradicts $g \preceq \varphi$. Hence, $g \preceq \psi$ and $\varphi \preceq \psi$.

□

A.2 Correctness of the Predecessor Calculation

We prove the correctness of Procedure 2. First we establish a property of the symbolic predecessor computation, which will be used in the proof.

Symbolic Predecessor Computation We prove a useful property of the symbolic predecessor computation.

Proposition 2. *Given patterns $\varphi = \langle N_\varphi, E_\varphi, \mathcal{G}_\varphi^-\rangle$, $\psi = \langle N_\psi, E_\psi, \mathcal{G}_\psi^-\rangle$ and an action $\alpha = \langle L, R\rangle$. If $\varphi \preceq_h \psi$ and $h(\mathcal{G}_\psi^-) \simeq \mathcal{G}_\varphi^-$, then for each $\varphi' \in \textsc{Pre}(\alpha, \varphi)$ there exists some $\psi' \in \textsc{Pre}(\alpha, \psi)$ such that $\varphi' \preceq \psi'$.*

Proof. Assume that $\varphi \preceq_h \psi$ and $h(\mathcal{G}^-_\psi) \simeq \mathcal{G}^-_\varphi$. We show that for each predecessor φ' of φ there exists a predecessor ψ' of ψ such that $\varphi' \preceq \psi'$ as illustrated below.

$$\begin{tikzcd}
\varphi' \arrow[d, phantom, sloped, "\preceq"] & \varphi \arrow[l, "\textsc{Pre}(\alpha)"'] \arrow[d, phantom, sloped, "\preceq_h"] \\
\exists\, \psi' & \psi \arrow[l, "\textsc{Pre}(\alpha)"']
\end{tikzcd}$$

After renaming the nodes so that the sets of nodes are disjoint, we compute $\textsc{Pre}(\alpha, \varphi)$ and $\textsc{Pre}(\alpha, \psi)$ according to Procedure 2.

We consider all predecessors of φ . Let thus $h_{R\varphi} : N_R \to N_\varphi$ be the chosen partial injection on line 3 which causes the predecessor φ' of φ to be output. We show that a corresponding injection gives us the desired predecessor ψ' — namely the following:

$$h_{R\psi} = h^{-1} \circ h_{R\varphi} : N_R \to N_\psi \,;\, n \mapsto h^{-1}(h_{R\varphi}(n))\,.$$

We now argue line by line, of Procedure 2, that subsumption is preserved during the predecessor computation for our choices of injections. Initially, we have $\varphi \preceq_h \psi$. Clearly, after the renaming done on line 4 we still have subsumption. The following gives us a simpler correspondence between φ and ψ .

Lemma 1. *If $\varphi \preceq_h \psi$ then $\widehat{\varphi} \preceq_{Id} \widehat{\psi}$, where $\widehat{\varphi} = h^{-1}_{R\varphi}(\varphi)$ and $\widehat{\psi} = h^{-1}_{R\psi}(\psi)$ are the patterns obtained after the renaming.*

Proof. Since $\varphi \preceq_h \psi$ we have $h^{-1}(\varphi) \preceq_{Id} \psi$. Since the sets N_ψ and N_R are disjoint, we can apply $h^{-1}_{R\psi}$ to both sides, obtaining $h^{-1}_{R\psi} \circ h^{-1}(\varphi) \preceq_{Id} \widehat{\psi}$. By definition $h^{-1}_{R\psi} = (h^{-1} \circ h_{R\varphi})^{-1} = (h^{-1}_{R\varphi} \circ h)$ so the left hand side becomes $\widehat{\varphi}$. □

For readability, we abuse notation slightly, and continue to call the renamed patterns φ and ψ . Let thus $\varphi := h^{-1}_{R\varphi}(\varphi)$ and $\psi := h^{-1}_{R\psi}(\psi)$. By Lemma 1 we thus have $\varphi \preceq_{Id} \psi$.

We continue with the test on line 5 . It suffices to show that if φ is not skipped, i.e. both clauses are false, then ψ is not skipped either.

Lemma 2. *If the first clause of the test on line 5 is false for φ then it is also false for ψ .*

Proof. That the first clause is false, means that for the quantified n , $\mathcal{E}_+(n, \varphi) \subseteq \mathcal{E}_+(n, R)$. Since $\varphi \preceq_{Id} \psi$ we get $N_\psi \subseteq N_\varphi$ and $E_\psi \subseteq E_\varphi$, and the statement follows.

□

Lemma 3. *If the second clause of the test on line 5 is false for φ then it is also false for ψ .*

Proof. *That the second clause is false for* φ *means that* $\neg\mathcal{I}nconsistent(\varphi + R)$ *. This means that there is no NAC in* $\mathcal{G}_\varphi^-$ *which contradicts* R *. The statement follows since* $\varphi \preceq_{Id} \psi$ *as for Lemma 2.* □

Next we show that the computation on line 8 preserves subsumption.

Let $\varphi' = (\varphi \ominus_\alpha R) + L$ and $\psi' = (\psi \ominus_\alpha R) + L$.

Lemma 4. *If* $\varphi \preceq_{Id} \psi$ *then* $\varphi' \preceq_{Id} \psi'$ *.*

Proof. *Suppose that* $\varphi \preceq_{Id} \psi$ *.*

$$(\varphi \ominus_\alpha R) + L = \langle N_\varphi \setminus (N_R \setminus N_L) \cup N_L, E_\varphi \setminus E_R \cup E_L, \mathcal{G}_\varphi^- \cup \mathcal{G}_L^- \rangle$$

$$(\psi \ominus_\alpha R) + L = \langle N_\psi \setminus (N_R \setminus N_L) \cup N_L, E_\psi \setminus E_R \cup E_L, \mathcal{G}_\psi^- \cup \mathcal{G}_L^- \rangle$$

We check the conditions of Proposition 1 after line 8.

1. *This condition is true, because the same edges are removed from and added to* φ *and* ψ *.*
2. *This condition holds, because the same NACs are added to* φ *and* ψ *.*

□

We show that the NAC handling done on lines 9–11 also preserves subsumption. We will use φ'' and ψ'' to denote the results from executing lines 9–11.

Lemma 5. *If* $\varphi' \preceq_{Id} \psi'$ *then* $\varphi'' \preceq_{Id} \psi''$ *.*

Proof. *Because the sets of NACs are isomorphic, whenever a NAC is removed from* φ' *it is also removed from* ψ' *. Hence subsumption is preserved.* □

For the inconsistency check on line 12 of Procedure 2 we conclude that since we have $\varphi'' \preceq_{Id} \psi''$, if ψ'' is inconsistent, then so is φ'' . Hence, if φ'' passes the test, it is not inconsistent and neither is ψ'' .

Let us return to the original notations, used in the statement. We simply rename our patterns: $\varphi' := \varphi''$ and $\psi' := \psi''$. Since φ' is a predecessor of φ , it will pass this last inconsistency check. It follows that so will ψ' . Hence, we have found our ψ' with $\psi' \in \textsc{Pre}(\alpha, \psi)$ such that $\varphi' \preceq \psi'$, concluding the proof. □

Main Proof Now we continue with the main proof. We will use the following correspondence between graph and pattern subsumption.

Lemma 6. *Given a hypergraph* $g = \langle N_g, E_g \rangle$ *and a consistent pattern* $\varphi = \langle N_\varphi, E_\varphi, \mathcal{G}_\varphi^- \rangle$*.*

$$g \preceq_h \varphi \quad \Longleftrightarrow \quad \varphi_g = \langle N_g, E_g, h(\mathcal{G}_\varphi^-) \rangle \preceq_h \varphi \text{ and } \varphi_g \text{ is consistent} .$$

Proof. *(*$\Longrightarrow$*). Suppose* $g \preceq_h \varphi$ *. Then* g *does not contradict any NAC of* φ *. Hence,* $\langle N_g, E_g, h(\mathcal{G}_\varphi^-) \rangle$ *is consistent and subsumed by* φ *(witnessed by* h *and the identity mapping between the NACs).*

($\Longleftarrow$*). Suppose* $\varphi_g = \langle N_g, E_g, h(\mathcal{G}_\varphi^-) \rangle \preceq_h \varphi$ *and* φ_g *is consistent. Since* φ_g *is consistent, we get* $g = \langle N_g, E_g \rangle \preceq_{Id} \varphi_g$ *. Furthermore, since* $\varphi_g \preceq \varphi$ *we have* $g \preceq_{Id} \varphi_g \preceq \varphi$ *. Hence,* $g \preceq \varphi$ *.* □

Finally, we are ready to prove correctness.

Proposition 3. *Given an action* $\alpha = \langle L, R\rangle$, *and a consistent pattern* $\varphi = \langle N_\varphi, E_\varphi, \mathcal{G}^-_\varphi\rangle$.

$$\Phi = \textsc{Pre}(\alpha, \varphi) \text{ satisfies } pre\,(\alpha, [\![\varphi]\!]) \subseteq [\![\Phi]\!]\,.$$

Proof. Our proof strategy is depicted in Figure 1. We consider any pair of graphs $(g', g) \in [\![\alpha]\!]$ where $g \preceq \varphi$. We first show that there exist patterns φ_g and φ' as shown in the figure — i.e., such that $g \preceq \varphi_g$, $\varphi_g \preceq \varphi$, $\varphi' \in \textsc{Pre}(\alpha, \varphi_g)$ and $g' \preceq \varphi'$. Once this has been established, we get by Proposition 2 that there exists a pattern $\psi \in \textsc{Pre}(\alpha, \varphi)$ such that $\varphi' \preceq \psi$. Finally, we get $g' \preceq \psi$ by transitivity.

$$\begin{tikzcd}
g' \arrow[d, phantom, sloped, "\preceq"] & g \arrow[l, "pre\,(\alpha)"'] \arrow[d, phantom, sloped, "\preceq"] \\
\exists\,\varphi' \arrow[d, phantom, sloped, "\preceq"] & \exists\,\varphi_g \arrow[l, "\textsc{Pre}(\alpha)"'] \arrow[d, phantom, sloped, "\preceq"] \\
\exists\,\psi & \varphi \arrow[l, "\textsc{Pre}(\alpha)"']
\end{tikzcd}$$

Fig. 1. Proof strategy. We show that the patterns preceded by "$\exists$" exist. The proposition statement then follows by transitivity of $\preceq$.

By Lemma 6 we get that $\varphi_g = \langle N_g, E_g, \mathcal{G}^-_\varphi\rangle \preceq \varphi$ and, by consistency, that $g \preceq \varphi_g$. Now, by Proposition 2, we get that for any $\varphi' \in \textsc{Pre}(\alpha, \varphi_g)$ there exists $\psi \in \textsc{Pre}(\alpha, \varphi)$ with $\varphi' \preceq \psi$. It now suffices to show that there exists a predecessor φ' of φ_g such that $g' \preceq \varphi'$.

Suppose that the injection $h : N_R \to N_g$ relates g' and g according to Definition 1. Hence we have $g = \langle N_g, E_g\rangle$ with $N_g = N_{g'} \cup h(N_R)$ and $E_g = E_{g'} \setminus h(E_L) \cup h(E_R)$. We show that, if we choose this *same* injection on line 3 in the computation of $\textsc{Pre}(\alpha, \varphi)$, we obtain an adequate φ'. Let us, then, go through lines 4–13 of Procedure 2 using the injection $h : N_R \to N_g$ from above.

- *Line 4.* After the renaming we obtain $\widehat{\varphi_g} = \langle h^{-1}(N_{g'}) \cup N_R\,,\ h^{-1}(E_{g'}) \setminus E_L \cup E_R\,,\ h^{-1}(\mathcal{G}^-_\varphi)\rangle$.
- *Line 5, first clause.* This clause is false, since the edges of $\widehat{\varphi_g}$ are $h^{-1}(E_{g'}) \setminus E_L \cup E_R$. Thus $\mathcal{E}_+(n, \widehat{\varphi_g})$ is clearly a subset of $\mathcal{E}_+(n, E_R)$ for $n \in (N_R \setminus N_L)$ (in fact, the sets are equal).
- *Line 5, second clause.*

  $$\widehat{\varphi_g} + R = \langle h^{-1}(N_{g'}) \cup N_R \cup N_R,\ h^{-1}(E_{g'}) \setminus E_L \cup E_R \cup E_R,\ h^{-1}(\mathcal{G}^-_\varphi)\rangle = \widehat{\varphi_g}$$

  Since $\widehat{\varphi_g}$ is consistent, so is $\widehat{\varphi_g} + R$.

– *Line 8.* We here obtain the pattern

$$\varphi' = (\widehat{\varphi_g} \ominus_\alpha R) + L = \langle h^{-1}(N_{g'}) \, , \, h^{-1}(E_{g'}) \, , \, h^{-1}(\mathcal{G}^-_\varphi) \cup \mathcal{G}^-_L \rangle \, .$$

– *Lines 9–11.* Suppose that $g' \not\preceq \varphi'$. Since condition 1 of subsumption is satisfied (as defined on page 5), the reason must be that condition 2 is not. But since by definition $g' \preceq_h L$, the violated NAC must be in $\mathcal{G}^-_\varphi$. Moreover, since $\varphi + R$ is consistent, the part of φ' which contradicts the NAC cannot be in R. The only remaining alternative is that something (positive) in $L \ominus_E R$ contradicts the NAC. Hence, condition 2 is met, and $g' \preceq \varphi'$, if all contradictions of this form are resolved. This is precisely what is done on lines 9–11.
– *Line 12.* Finally, φ' is consistent, since $g' \preceq \varphi'$, and the test on this line is passed. □